Matching Twigs in Probabilistic XML*

Benny Kimelfeld and Yehoshua Sagiv
The Selim and Rachel Benin School of Engineering and Computer Science
The Hebrew University of Jerusalem
Edmond J. Safra Campus Jerusalem 91904, Israel
{bennyk,sagiv}@cs.huji.ac.il

ABSTRACT

Evaluation of twig queries over probabilistic XML is investigated. Projection is allowed and, in particular, a query may be Boolean. It is shown that for a well-known model of probabilistic XML, the evaluation of twigs with projection is tractable under data complexity (whereas in other probabilistic data models, projection is intractable). Under query-and-data complexity, the problem becomes intractable even without projection (and for rather simple twigs and data).

In earlier work on probabilistic XML, answers are always complete. However, there is often a need to produce *partial answers* because XML data may have missing sub-elements and, furthermore, complete answers may be deemed irrelevant if their probabilities are too low. It is shown how to define a semantics that provides partial answers that are *maximal* with respect to a probability threshold, which is specified by the user. For this semantics, it is shown how to efficiently evaluate twigs, even under query-and-data complexity if there is no projection.

1. INTRODUCTION

Probabilistic data has been widely investigated due to many potential applications. Data models and query languages were proposed and algorithms for query evaluation were developed (Section 6 discusses related work). When querying probabilistic data, we have to compute the answers as well as the probability of each one. An alternative is to compute just those answers with a probability above some given threshold. In this paper, we study the evaluation of twig (i.e., tree) queries over probabilistic XML. Our data model for probabilistic XML is similar to that of [21].

Invariably, projection is always an obstacle. When applied to ordinary data, projection creates duplicates. Therefore, it can be handled straightforwardly as a post-processing step. However, projection introduces inherent complications when applied to probabilistic data. For example, one might think that since each duplicate has its own probability, we only need to take the sum in order to get the accumulated probability of a particular answer (e.g., tuple) that is represented by all its duplicates. However, this is incorrect, because the duplicates of a given answer are not disjoint events. They are also not independent events, which rules out other immediate solutions (e.g., using the complement probabilities). In [9] they show that, in probabilistic relational databases, evaluating projection is #P-hard even if it is only applied to tree queries with just three relations.

When querying XML using twig queries, projection means that an answer consists of the match of only a part of the query nodes. We present an algorithm for evaluating twig queries with projections (and in particular, Boolean queries) over probabilistic XML. Our algorithm is efficient under data complexity. We also show that, under query-and-data complexity, query evaluation is intractable even under strong restrictions (and even without projection).

Another important aspect of query evaluation over probabilistic XML is partial answers to queries. The phenomenon of missing sub-elements is pervasive in XML documents, and having probabilistic data only compounds it (because sub-elements with low probabilities might be deemed missing, for all practical purposes). Yet, existing approaches compute only answers that completely match the given query, even if these answers have low probabilities. At the same time, they ignore partial answers, even if they have high probabilities. This is a drawback when users are interested in partial (as well as complete) answers.

Our approach is to adapt existing work on maximal answers [4, 6, 17, 18, 19] to probabilistic XML. Intuitively, a *maximal answer* binds data items (i.e., values, objects or elements, depending on the data model) to a maximal subset of the query variables. In other words, one cannot extend a maximal answer by binding any additional variable.

Two issues arise when dealing with maximal answers. The first is efficiency. We do not want to generate all the partial answers and then choose the maximal ones. Thus, data complexity is not suitable for analyzing the running time of evaluation algorithms (because, when the size of the query is fixed, the time needed for generating all the partial answers and then choosing the maximal ones is greater by at most a polynomial factor than the time required for computing just the maximal answers). Therefore, we should analyze evaluation algorithms under query-and-data complexity.

The second issue is the notion of maximal answers over probabilistic data. If we use the ordinary definition (which applies to non-probabilistic data), then we are faced with

*This research was supported by The Israel Science Foundation (Grant 893/05).

VLDB '07, September 23-28, 2007, Vienna, Austria.

the problem discussed earlier. That is, we will get maximal answers with low probabilities and miss partial answers that have higher probabilities. Therefore, when a user specifies a probabilistic threshold, it should be used not just for filtering out answers (with low probabilities) in the last stage, but also for deciding whether an answer is maximal or not. Consequently, we define a maximal answer over probabilistic data as a partial answer that binds a subset V of the query variables, such that V is maximal with respect to the following condition. If we extend the answer by any possible binding to a variable that is not in V, then the probability of the answer will decrease below the threshold specified by the user.

We give an efficient algorithm, under query-and-data complexity, for computing maximal answers to projection-free twig queries over probabilistic XML. We also show how to use this algorithm to evaluate queries with projection. In addition, we discuss the issues of completeness constraints [19] and ranked enumeration.

2. DOCUMENTS, TWIGS AND MATCHES

This section describes our non-probabilistic data model.

2.1 Trees

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

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

If the trees t' and t satisfy $\mathcal{V}(t') \subseteq \mathcal{V}(t)$ and $\mathcal{E}(t') \subseteq \mathcal{E}(t)$, then t' is a *subtree* of t, denoted by $t' \preceq t$. If, in addition, t' and t have the same root, then t' is an *r-subtree* of t, denoted by $t' \preceq_r t$.

A subset V of the nodes of a tree t *induces* the subgraph g of t that consists of all the nodes of V and the edges connecting those nodes. Note that g is a subtree of t if it is connected.

Consider a node n of a tree t. We use t^n_Δ to denote the subtree of t that is rooted at n and is induced by all the descendants of n.

Given a nonempty subset N of the nodes of a tree t, we say that an r-subtree t' of t is *reduced* with respect to (abbr. w.r.t.) N if $N \subseteq \mathcal{V}(t')$ and $leaves(t') \subseteq N$ (i.e., t' contains all the nodes of N, but no proper r-subtree of t' has this property). Note that a tree t has exactly one r-subtree, denoted by $t^{|N}$, that is reduced w.r.t. N.

2.2 Documents and Twigs

In this paper, data are unranked XML documents and queries are twig patterns (possibly with projection).

Formally, an *XML document* (or just *document* for short) is a tree with *values* and *tags* attached to its nodes. We use d to denote documents, and u, v and w to denote nodes of documents. The value and tag of a node v are denoted by $val(v)$ and $tag(v)$, respectively.

A *twig pattern* (or just *twig* for short) is a tree with *child* edges and *descendant* edges. Each node n of the twig is associated with a unary predicate $cond_n(\cdot)$ over nodes of documents. As explained later, n can match a node v of a document only if $cond_n(v)$ is satisfied. We use n and m to denote nodes of twigs (in order to make a clear distinction from nodes of documents).

Figure 2 shows two twigs. Note that child and descendant edges are depicted by single and double lines, respectively. All the node predicates in Figure 2 have the simple form $tag(\cdot) = \mathtt{Y}$, where $\mathtt{Y}$ is a tag. The unary predicate $cond_n(v)$ is allowed to express any condition about the value and tag of node v, provided that it can be tested efficiently. For example, $(tag(\cdot) = \mathtt{A} \vee tag(\cdot) = \mathtt{B}) \wedge (val(\cdot) \geq 10 \vee val(\cdot) = 5)$.

2.3 Three Semantics for Queries

We now define three different semantics for applying twigs, namely, *complete*, *incomplete* and *Boolean*.

The Complete Semantics. When interpreting a twig under the complete semantics, we call it a *c-twig* and denote it by C. Under this semantics, a match of a twig in a document is a mapping from the nodes of the twig to those of the document, such that all the constraints of the twig are satisfied. Formally, a *complete match* (abbr. *c-match*) of a c-twig C in a document d is a mapping $\varphi : \mathcal{V}(C) \to \mathcal{V}(d)$ that satisfies the following.

1. Roots are matched, i.e., $\varphi(root(C)) = root(d)$.
2. For all nodes $n \in \mathcal{V}(C)$, the predicate $cond_n(\varphi(n))$ is satisfied.
3. For all child edges $(n_p, n_c) \in \mathcal{E}(C)$, the node $\varphi(n_p)$ of d is the parent of $\varphi(n_c)$.
4. For all descendant edges $(n_a, n_d) \in \mathcal{E}(C)$, the node $\varphi(n_a)$ of d is a proper ancestor of $\varphi(n_d)$.

$C(d)$ denotes the set of all the c-matches of the c-twig C in the document d, and it is the *value* of C w.r.t. d.

Let $X \subseteq \mathcal{V}(C)$. The query $\pi_X\, C$ is the *projection* of C onto X. The *projection* of a c-match φ onto X is the restriction of φ to the nodes of X and is denoted by $\varphi^{[X]}$. The set $\pi_X\, C(d)$, which is the *value* of $\pi_X\, C$ w.r.t. d, consists of all the mappings obtained by projecting the c-matches of $C(d)$ onto X, that is,

$$\pi_X\, C(d) = \left\{ \varphi^{[X]} \mid \varphi \in C(d) \right\}.$$

Given a mapping ϕ that is defined on X, we use $Img(\phi)$ to denote its image, that is,

$$Img(\phi) = \{\phi(n) \mid n \in X\}.$$

EXAMPLE 2.1. *Consider the document d_1 that is shown at the bottom of the middle part of Figure 1. Note that the tags appear inside the circles that depict the nodes. (For simplicity, the nodes of d_1 do not have values.) In this example, we view T_1 of Figure 2 as c-twig and, hence, denote it by C. Note that C has only child edges and its node predicates are of the form $tag(\cdot) = \mathtt{Y}$. There are two c-matches of C in d_1. The first is $\{n_0 \mapsto v_0, n_1 \mapsto v_5, n_2 \mapsto v_6\}$ (i.e., n_0 is mapped to v_0, etc.) and the second is $\{n_0 \mapsto v_0, n_1 \mapsto v_7, n_2 \mapsto v_8\}$.*

Next, consider the query $\pi_X\, C$, where $X = \{n_1\}$. The set $\pi_X\, C(d_1)$ consists of the mappings $\{n_1 \mapsto v_5\}$ and $\{n_1 \mapsto v_7\}$.

The Incomplete Semantics. Under the incomplete semantics, a twig is called an *i-twig* and is denoted by I. A match of an i-twig may be defined only over an r-subtree of the i-twig, rather than over the whole i-twig. Formally, an *incomplete match* (abbr. *i-match*) of an i-twig I in a document d is a mapping $\varphi : V \to \mathcal{V}(d)$, such that $V \subseteq \mathcal{V}(I)$, the nodes of V induce an r-subtree C of I and φ is a c-match of C in d. $Dom(\varphi)$ denotes the *domain* of φ, i.e., the set of nodes $\mathcal{V}(C)$. $I(d)$ denotes the set of all i-matches of I in d.

Note that although I is interpreted under the incomplete semantics, the notion of a c-match of I is well defined. In particular, every c-match of I is also an i-match.

Consider an i-twig I and a document d. Given two i-matches $\varphi_1, \varphi_2 \in I(d)$, we say that φ_2 *subsumes* φ_1, denoted by $\varphi_1 \sqsubseteq \varphi_2$, if $Dom(\varphi_1) \subseteq Dom(\varphi_2)$ and $\varphi_1(n) = \varphi_2(n)$ for all nodes $n \in Dom(\varphi_1)$. If, in addition, $\varphi_1 \neq \varphi_2$, then φ_2 *properly* subsumes φ_1, denoted by $\varphi_1 \sqsubset \varphi_2$. Finally, given a subset J of $I(d)$, the set of *maximal* i-matches of J is denoted by $\max(J)$ and is defined as the set of all i-matches $\varphi \in J$, such that no i-match of J properly subsumes φ.

EXAMPLE 2.2. *Consider again the document d_1 of Figure 1 and the twig T_1 of Figure 2 that we now interpret as an i-twig and denote by I. Clearly, both $\{n_0 \mapsto v_0, n_1 \mapsto v_5\}$ and $\{n_0 \mapsto v_0, n_1 \mapsto v_5, n_2 \mapsto v_6\}$ are i-matches of I in d_1. Moreover, the latter subsumes the former. In fact, in this particular example, the only maximal i-matches of I are c-matches. However, if we change the node predicate of n_2 to $tag(\cdot) = \mathtt{C}$, then the first of the above two i-matches is maximal whereas the second is not an i-match at all.*

The Boolean Semantics. When a twig is interpreted as a Boolean query, it is called a *b-twig* and is denoted by B (or R). Formally, given a document d, we define $B(d) =$ **true** if there is a c-match of B in d; otherwise, $B(d) =$ **false**.

3. PROBABILISTIC MODEL

3.1 Probabilistic XML

Probabilistic XML is a probability distribution over a space of documents (of the type defined in Section 2.2). When we want to emphasize that a document belongs to this space, we call it a *random* document. As in [21], we describe this space by means of a *probabilistic document* (abbr. *p-document*), which is a tree $\mathcal{P}$ that has two types of nodes. *Ordinary* nodes are regular XML nodes (with a tag and a value) and they may appear in random documents. *Distributional* nodes are only used for defining the probabilistic process of generating random documents (but they do not occur in those documents). A distributional node specifies a distribution over the subsets of its children.

Intuitively, a random document of a p-document $\mathcal{P}$ is generated by a two-step procedure. In the first step, we generate a random r-subtree s of $\mathcal{P}$ by applying the following top-down process, starting at the root. If we are at an ordinary node, we simply proceed to its children. When reaching a distributional node, we randomly choose a (possibly empty) subset of its children and proceed to each chosen child. We delete each unchosen child and its descendants. The result of the top-down process is a random r-subtree s of $\mathcal{P}$ (we call s a *sample* of $\mathcal{P}$). Note that s is not a document, because it contains distributional nodes of $\mathcal{P}$. So, in the second step, we remove all the distributional nodes. If we remove the parent of an ordinary node u, then the new parent of u is its lowest ancestor among all the ordinary nodes. Next, we formally define p-documents and their random documents.

A p-document $\mathcal{P}$ is a tree and its set of nodes $\mathcal{V}(P)$ is divided into two disjoint subsets, $\mathcal{V}^{ord}(\mathcal{P})$ and $\mathcal{V}^{dst}(\mathcal{P})$, that contain the ordinary and distributional nodes, respectively. A distributional node is neither the root nor a leaf of $\mathcal{P}$.

As in [21], $\mathcal{V}^{dst}(\mathcal{P})$ contains two types of distributional nodes. An *IDD* node has children that are probabilistically independent of each other, while the children of an *MXD* node are *mutually exclusive,* that is, at most one child can exist in the random r-subtree s described above.

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

$$\mathcal{P}(u, \emptyset) = 1 - \sum_{i=1}^{k} \mathcal{P}(u, v_i)\,.$$

If u is an IDD node, then we define

$$\mathcal{P}(u, \emptyset) = \prod_{i=1}^{k} \left(1 - \mathcal{P}(u, v_i)\right)\,.$$

EXAMPLE 3.1. *Consider the p-document $\mathcal{P}$ shown on the left side of Figure 1. Ordinary nodes are depicted as circles; their tags appear inside the circles and, for simplicity, they do not have values. Distributional nodes are shown as rectangular boxes with rounded corners. Thus, $\mathcal{V}^{ord}(\mathcal{P})$ consists of $v_0, v_1, \ldots, v_{10}$ and $\mathcal{V}^{dst}(\mathcal{P})$ comprises $u_1, \ldots, u_5$. MXD nodes are marked by filled circles. So, u_1 and u_5 are MXD nodes while u_2, u_3 and u_4 are IDD nodes. The probabilities $\mathcal{P}(\cdot, \cdot)$ are shown next to the corresponding edges.*

Consider the IDD node u_2. This node has two children v_1 and v_2 with the probabilities $\mathcal{P}(u_2, v_1) = 0.9$ and $\mathcal{P}(u_2, v_2) = 0.8$. Thus, $\mathcal{P}(u_2, \emptyset) = 0.1 \times 0.2 = 0.02$. Now, consider the MXD node u_1 that also has two children, namely, u_2 and v_3 with the probabilities $\mathcal{P}(u_1, u_2) = 0.5$ and $\mathcal{P}(u_1, v_3) = 0.3$. Hence, $\mathcal{P}(u_1, \emptyset) = 1 - 0.5 - 0.3 = 0.2$. Similarly, for the MXD node u_5, we have that $\mathcal{P}(u_5, \emptyset) = 1 - 0.4 - 0.5 - 0.1 = 0$.

In order to describe the discrete probability space of random documents, we first define the samples and their probabilities. Recall that the samples of a p-document $\mathcal{P}$ are the r-subtrees of $\mathcal{P}$ that may be obtained by invoking the random process described above. Formally, a *sample* of $\mathcal{P}$ is an r-subtree $s \preceq_r \mathcal{P}$ that satisfies the following two conditions.

1. (*Determinism Preservation*) Each node $v \in \mathcal{V}^{ord}(\mathcal{P}) \cap \mathcal{V}(s)$ has the same set of children in s and in $\mathcal{P}$.
2. (*Mutual Exclusion*) Each MXD node $u \in \mathcal{V}^{dst}(\mathcal{P}) \cap \mathcal{V}(s)$ has at most one child in s.

$\Omega(\mathcal{P})$ denotes the set of all samples of $\mathcal{P}$. Suppose that $s \in \Omega(\mathcal{P})$ and let $u_1, \ldots, u_l$ be the distributional nodes that

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

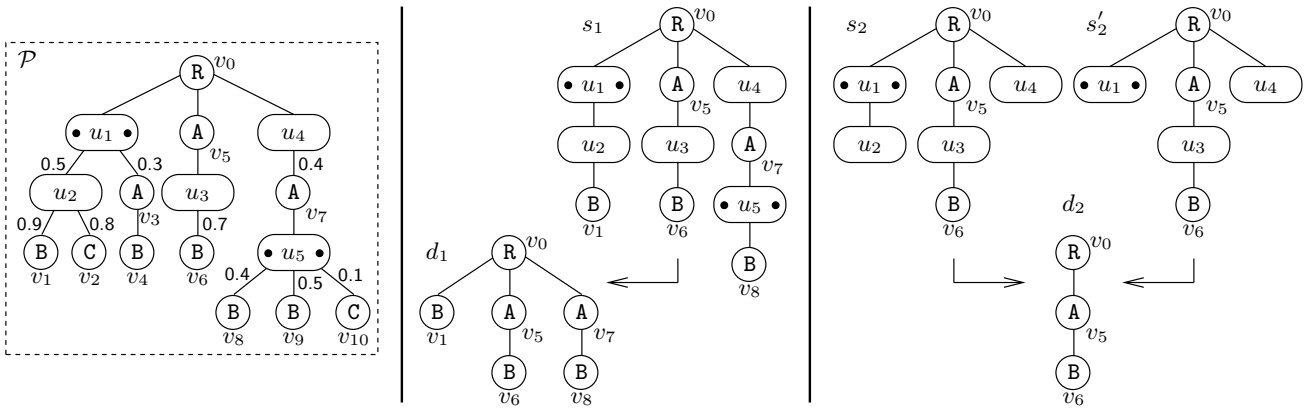

Figure 1: A p-document, samples and corresponding random documents

appear in s. The probability of s is $\prod_{i=1}^{l} p_i$, where p_i is the probability that the children of u_i are exactly those appearing in s. If u_i is an IDD node, then p_i is the product consisting of $\mathcal{P}(u_i, v)$ for each child v that appears in s and $1 - \mathcal{P}(u_i, v)$ for each child v that is not in s, that is,

$$p_i = \prod_{(u_i,v)\in\mathcal{E}(s)} \mathcal{P}(u_i, v) \prod_{(u_i,v)\in\mathcal{E}(\mathcal{P})\setminus\mathcal{E}(s)} (1 - \mathcal{P}(u_i, v)) .$$

If u_i is MXD, then $p_i = \mathcal{P}(u_i, v)$ if v is the single child of u_i in s; if u_i has no children in s, then $p_i = \mathcal{P}(u_i, \emptyset)$. Clearly, the probabilities of all the samples add up to 1.

EXAMPLE 3.2. *The p-document $\mathcal{P}$ and three of its samples (namely, s_1, s_2 and s_2') are shown in Figure 1. Note that s_2 is different from s_2', since the former contains the node u_2 and the latter does not.*

Each sample s naturally defines an ordinary document, denoted by $Doc(s)$. Formally, $Doc(s)$ is obtained from s by deleting all the the distributional nodes. If the parent of an ordinary node v is deleted, then v becomes a child of its lowest ordinary ancestor. Note that $Doc(\cdot)$ is not one-to-one; that is, two different samples may yield the same document. This follows from the fact that a distributional node can have a distributional child.

EXAMPLE 3.3. *Consider again the p-document $\mathcal{P}$ and the samples s_1, s_2 and s_2' from Example 3.2. Figure 1 also shows the documents d_1 and d_2. Note that $Doc(s_1) = d_1$ while $Doc(s_2) = Doc(s_2') = d_2$.*

We denote by $S[\mathcal{P}]$ the random variable that represents the sample obtained from $\mathcal{P}$. $D[\mathcal{P}]$ is the random variable $Doc(S[\mathcal{P}])$. In other words, $D[\mathcal{P}]$ represents a random document that is chosen from the distribution defined by $\mathcal{P}$. As noted above, a single document can be obtained from several different samples.

EXAMPLE 3.4. *Consider again Figure 1. Let us compute the probability of the document d_2, i.e.,* $\Pr(D[\mathcal{P}] = d_2)$. *It can be shown that s_2 and s_2' are the only samples that generate d_2, so*

$$\Pr(D[\mathcal{P}] = d_2) = \Pr(S[\mathcal{P}] = s_2) + \Pr\left(S[\mathcal{P}] = s_2'\right) .$$

For s_2 to be obtained, all of the following independent events should occur: (1) *u_1 chooses u_2,* (2) *u_2 chooses none of its children,* (3) *u_3 chooses v_6,* and (4) *u_4 does not choose v_7. Thus,*

$$\Pr(S[\mathcal{P}] = s_2) = 0.5 \times (0.1 \times 0.2) \times 0.7 \times 0.6 = 0.0042 .$$

Similarly,

$$\Pr\left(S[\mathcal{P}] = s_2'\right) = 0.2 \times 0.7 \times 0.6 = 0.084 .$$

Therefore,

$$\Pr(D[\mathcal{P}] = d_2) = 0.0042 + 0.084 = 0.0882 .$$

The method of Example 3.4 for computing the probability of a given document is inefficient because, in general, the number of samples that yield a specific document could be exponential in the size of $\mathcal{P}$. Nevertheless, this probability can be computed efficiently.

PROPOSITION 3.5. $\Pr(D[\mathcal{P}] = d)$ *can be computed in polynomial time in the size of $\mathcal{P}$.*

The algorithm for computing $\Pr(D[\mathcal{P}] = d)$ is omitted due to a lack of space. Moreover, this algorithm is not necessary for obtaining the main results of this paper.

3.2 Probabilistic Semantics of Queries

We consider a query Q and start with an intuitive explanation. To simplify the terminology, the discussion below assumes that Q is a c-twig, but in principle it also applies to i-twigs and b-twigs.

When dealing with ordinary XML, the input to Q is a document and the output is a set of c-matches, as explained in Section 2. In probabilistic XML, in contrast, the input is a p-document $\mathcal{P}$, which represents a probabilistic space that may have an exponential number of documents (in the size of $\mathcal{P}$). Consequently, it is intractable to apply Q to each document in this space. Instead, the common approach (e.g., [9, 21]) is to compute for each possible c-match ρ, the probability that ρ is produced when applying Q to a random document. Next, we describe the formal notation for expressing this probability.

Recall that the random variable $D[\mathcal{P}]$ represents a document generated by $\mathcal{P}$. So, for the given query Q, the random

variable $Q(D[\mathcal{P}])$ represents the application of Q (as defined in Section 2) to the random document $D[\mathcal{P}]$. Therefore, $\Pr(\rho \in Q(D[\mathcal{P}]))$ denotes the probability that the c-match ρ is produced when applying the query Q to a random document. That is, $\Pr(\rho \in Q(D[\mathcal{P}]))$ is the sum of the probabilities of all random documents d of $\mathcal{P}$, such that $\rho \in Q(d)$.

The result of applying the query Q to the p-document $\mathcal{P}$ is the set of all possible c-matches ρ of Q in random documents of $\mathcal{P}$. For each c-match ρ of the result, we compute the probability $\Pr(\rho \in Q(D[\mathcal{P}]))$. In [20], it is argued that matches with low probabilities are insignificant. We follow this principle and assume that the user specifies a *threshold* $p \in [0,1]$ and is interested only in results that have a probability of at least p. Next, we formally define the three semantics of queries in probabilistic XML.

3.2.1 The Complete Semantics

Consider a p-document $\mathcal{P}$ and a c-twig C. We define $C(\mathcal{P})$ as the set of all c-matches of C in the random documents of $\mathcal{P}$, that is,

$$C(\mathcal{P}) \stackrel{\text{def}}{=} \bigcup_{s \in \Omega(\mathcal{P})} C(Doc(s)) .$$

The output of C for the threshold p is denoted by $C^{\uparrow p}(\mathcal{P})$ and formally defined as follows.

$$C^{\uparrow p}(\mathcal{P}) \stackrel{\text{def}}{=} \{\phi \in C(\mathcal{P}) \mid \Pr(\phi \in C(D[\mathcal{P}])) \geq p\}$$

If the query is a projection of C onto a set of nodes X, then we denote it by $\pi_X C$. The set $\pi_X C(\mathcal{P})$ is obtained by applying the projection to each c-match of $C(\mathcal{P})$, that is,

$$\pi_X C(\mathcal{P}) = \left\{ \varphi^{[X]} \mid \varphi \in C(\mathcal{P}) \right\} .$$

The output of $\pi_X C$ for the threshold p is denoted by $\pi_X^{\uparrow p} C(\mathcal{P})$ and defined as follows.

$$\pi_X^{\uparrow p} C(\mathcal{P}) \stackrel{\text{def}}{=} \{\gamma \in \pi_X C(\mathcal{P}) \mid \Pr(\gamma \in \pi_X C(D[\mathcal{P}])) \geq p\}$$

Example 3.6. *Let $\mathcal{P}$ be the p-document of Figure 1. In this example, we interpret the twig T_1 of Figure 2 under the complete semantics and denote it by C. Note that C has only child edges and its node predicates are of the form $tag(\cdot) = \mathtt{Y}$. Let φ be the c-match $\{n_0 \mapsto v_0, n_1 \mapsto v_7, n_2 \mapsto v_9\}$. Clearly, $\Pr(\varphi \in C(D[\mathcal{P}]))$ is the same as the probability that v_9 exists in $D[\mathcal{P}]$. Therefore,*

$$\Pr(\varphi \in C(D[\mathcal{P}])) = 0.4 \times 0.5 = 0.2 .$$

Now, we consider the query $\pi_X C$, where $X = \{n_1\}$. The set $\pi_X C(\mathcal{P})$ consists of the mappings $\{n_1 \mapsto v_3\}$, $\{n_1 \mapsto v_5\}$ and $\{n_1 \mapsto v_7\}$. Let us compute the probability of the mapping $\gamma = \{n_1 \mapsto v_7\}$. Observe that $\gamma \in \pi_X C(D[\mathcal{P}])$ holds exactly when $D[\mathcal{P}]$ contains v_7 and at least one of v_8 and v_9. The probability that $D[\mathcal{P}]$ contains v_7 is 0.4. Since u_5 is an MXD node, the probability that either v_8 or v_9 exists, given that v_7 exists, is $0.4 + 0.5 = 0.9$. Therefore,

$$\Pr(\gamma \in \pi_X C(D[\mathcal{P}])) = 0.4 \times 0.9 = 0.36 .$$

3.2.2 The Incomplete Semantics

Consider a p-document $\mathcal{P}$ and an i-twig I. The set of all i-matches of I in the random documents of $\mathcal{P}$ is denoted by $I(\mathcal{P})$ and formally defined as follows.

$$I(\mathcal{P}) \stackrel{\text{def}}{=} \bigcup_{s \in \Omega(\mathcal{P})} I(Doc(s))$$

Given a threshold $p \in [0,1]$, the set of all i-matches with a probability of at least p is denoted by $I^{\uparrow p}(\mathcal{P})$ and formally defined as follows.

$$I^{\uparrow p}(\mathcal{P}) \stackrel{\text{def}}{=} \{\phi \in I(\mathcal{P}) \mid \Pr(\phi \in I(D[\mathcal{P}])) \geq p\}$$

However, $I^{\uparrow p}(\mathcal{P})$ is not the output of I, since it has redundant i-matches, due to the fact that if $\phi \in I^{\uparrow p}(\mathcal{P})$, then all the i-matches that are subsumed by ϕ are also in $I^{\uparrow p}(\mathcal{P})$. Thus, we define the output of I for the threshold p as the set $\max\left(I^{\uparrow p}(\mathcal{P})\right)$, which contains only i-matches of $I^{\uparrow p}(\mathcal{P})$ that are not subsumed by other i-matches of $I^{\uparrow p}(\mathcal{P})$.

Note that for a given i-match $\phi \in I(\mathcal{P})$, the probability $\Pr(\phi \in I(D[\mathcal{P}]))$ is the same as the probability that a random document contains the image of ϕ. Thus, this probability can be computed as follows. First, obtain the subtree $\mathcal{P}^{|\,Img(\phi)}$, i.e., the r-subtree of $\mathcal{P}$ that is reduced w.r.t. the image of ϕ (see the definition in Section 2.1). Then, compute the product of the probabilities $\mathcal{P}(u,v)$ for all edges (u,v) of $\mathcal{P}^{|\,Img(\phi)}$, such that u is a distributional node. Note that the same procedure computes the probability of a c-match ϕ of a c-twig C, i.e., $\Pr(\phi \in C(D[\mathcal{P}]))$. However, computing the probability of a projection of a c-match is much more subtle and is discussed in Section 4.

Example 3.7. *Consider again the p-document $\mathcal{P}$ of Figure 1. We now interpret T_1 (shown in Figure 2) as an i-twig and, hence, denote it by I. The following are some of the i-matches of $I(\mathcal{P})$.*

- $\gamma_1 = \{n_0 \mapsto v_0\}$,
- $\gamma_2 = \{n_0 \mapsto v_0, n_1 \mapsto v_7\}$,
- $\gamma_3 = \{n_0 \mapsto v_0, n_1 \mapsto, v_7, n_2 \mapsto v_8\}$, *and*
- $\gamma_4 = \{n_0 \mapsto v_0, n_1 \mapsto v_7, n_2 \mapsto v_9\}$.

Both γ_3 and γ_4 subsume γ_2 which, in turn, subsumes γ_1. Furthermore, no other match of $I(\mathcal{P})$ subsumes either γ_2, γ_3 or γ_4. Clearly, $\Pr(\gamma_1 \in I(D[\mathcal{P}])) = 1$, $\Pr(\gamma_2 \in I(D[\mathcal{P}])) = 0.4$, $\Pr(\gamma_3 \in I(D[\mathcal{P}])) = 0.16$ and $\Pr(\gamma_4 \in I(D[\mathcal{P}])) = 0.2$. Therefore, the i-matches among γ_1, γ_2, γ_3 and γ_4 that belong to $\max\left(I^{\uparrow p}(\mathcal{P})\right)$ are as follows. If $p = 0.3$, then only γ_2 is in $\max\left(I^{\uparrow p}(\mathcal{P})\right)$. If $p = 0.2$, then only γ_4 is in $\max\left(I^{\uparrow p}(\mathcal{P})\right)$. Finally, for $p = 0.1$, both γ_3 and γ_4 are in $\max\left(I^{\uparrow p}(\mathcal{P})\right)$ (whereas γ_1 and γ_2 are not).

Note that the i-match γ_4 is the same as the c-match φ of Example 3.6. Both γ_4 and φ have the same probability, which is not a coincidence, because the probability of a match (which is both an i-match and a c-match) depends only on the nodes in its image, but not on the semantics of the query.

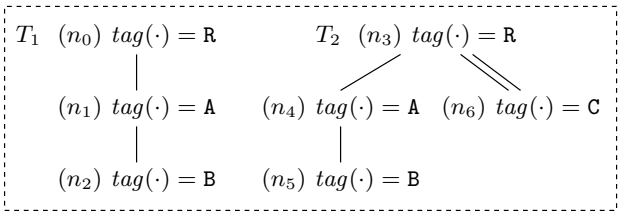

Figure 2: Twigs

Recall that the output of an i-twig I is defined as the set $\max\left(I^{\uparrow p}(\mathcal{P})\right)$, namely, we first choose all the i-matches that have a probability of at least p and then remove subsumed i-matches. This semantics is monotone in the sense that the output is increased if we either lower the threshold or add more data to the p-document. Note that when we say that the "output is increased," we mean that each i-match of the old output is subsumed by some i-match of the new output.

As an alternative, one could define the output by first removing subsumed i-matches and then selecting those that have a probability of at least p. But doing so would result in a semantics that is not monotone under addition of data to the p-document.

3.2.3 The Boolean Semantics

Given a Boolean query B and a p-document $\mathcal{P}$, we are only interested in the probability $\Pr\left(B(D[\mathcal{P}]) = \mathbf{true}\right)$. Clearly, $\Pr\left(B(D[\mathcal{P}]) = \mathbf{false}\right) = 1 - \Pr\left(B(D[\mathcal{P}]) = \mathbf{true}\right)$.

Instead of $\Pr\left(B(D[\mathcal{P}]) = \mathbf{true}\right)$ and $\Pr\left(B(D[\mathcal{P}]) = \mathbf{false}\right)$, we write $\Pr\left(B(D[\mathcal{P}])\right)$ and $\Pr\left(\neg B(D[\mathcal{P}])\right)$, respectively.

EXAMPLE 3.8. *Consider again the p-document $\mathcal{P}$ of Figure 1. We now interpret T_1 of Figure 2 as a b-twig and denote it by B. In this example, we compute* $\Pr\left(B(D[\mathcal{P}])\right)$. *There is a c-match of B in $D[\mathcal{P}]$ if and only if at least one of the following three independent events occur.* (1) *$D[\mathcal{P}]$ contains v_3,* (2) *$D[\mathcal{P}]$ contains v_6,* or (3) *$D[\mathcal{P}]$ contains v_7 and at least one of v_8 and v_9. Thus, the probability that none of these events occurs is equal to* $\Pr\left(\neg B(D[\mathcal{P}])\right)$.

Clearly, the probabilities that the first and second events do not occur are $1 - 0.3$ and $1 - 0.7$, respectively. The complement of the third event is the union of the following two disjoint events. The first one is that v_7 does not exist (and its probability is $1 - 0.4 = 0.6$). The second event is that v_7 exists, but neither v_8 nor v_9 exists (and its probability is $0.4 \times (1 - 0.4 - 0.5) = 0.04$, since u_5 is an MXD node). Thus, $\Pr\left(\neg B(D[\mathcal{P}])\right) = 0.7 \times 0.3 \times (0.6 + 0.04) = 0.1344$. *We conclude that* $\Pr\left(B(D[\mathcal{P}])\right) = 1 - 0.126 = 0.8656$.

Note that the above example is rather simple, because T_1 is a path. A similar approach would not work if the b-twig is defined by T_2 of Figure 2. In Section 4, we show how to compute the probability of an answer to a general b-twig.

3.3 Evaluation Problems

In the following sections, we consider the evaluation of queries over p-documents under the three semantics. In particular, we deal with three different computational problems:

- *EvalComplete*: Enumerate $\pi_X^{\uparrow p} C(\mathcal{P})$, given a c-twig C, a set $X \subseteq \mathcal{V}(C)$, a p-document $\mathcal{P}$ and a threshold p.
- *EvalMaximal*: Enumerate $\max(I^{\uparrow p}(\mathcal{P}))$, given an i-twig I, a p-document $\mathcal{P}$ and a threshold p.
- *EvalBoolean*: Determine $\Pr\left(B(D[\mathcal{P}])\right)$, given a b-twig B and a p-document $\mathcal{P}$.

4. EVALUATING C-TWIGS AND B-TWIGS

4.1 Reducing Projections to Boolean Queries

In this section, we study the complexity of the problems *EvalComplete* and *EvalBoolean*. We first describe an algorithm for solving both problems. This algorithm is efficient under data complexity, i.e., when the size of the query is bounded by a constant. Later in this section, we show that both problems are intractable under query-and-data complexity, even under significant simplifying assumptions.

Before presenting the algorithm for solving *EvalBoolean*, we show that *EvalComplete* can be reduced to *EvalBoolean*. Consider a p-document $\mathcal{P}$, a c-twig C, a subset X of the nodes of C and a threshold p. Our goal is to evaluate $\pi_X^{\uparrow p} C(\mathcal{P})$ by using an algorithm that computes $\Pr\left(B(D[\mathcal{P}])\right)$ for a b-twig B. Under data complexity, we can find all the mappings $\gamma : X \to \mathcal{V}^{ord}(\mathcal{P})$ in polynomial time. So, it is sufficient to show how to compute the probability of each mapping, since the output consists of the ones with a probability of at least p. (Clearly, optimizations can be applied.)

So, we consider a mapping $\gamma : X \to \mathcal{V}^{ord}(\mathcal{P})$ and show how to compute its probability, that is, $\Pr\left(\gamma \in \pi_X C(D[\mathcal{P}])\right)$. If there is an $n \in X$, such that $\gamma(n)$ does not satisfy the node predicate of n (i.e., $cond_n(\gamma(n)) = \mathbf{false}$), then the probability of γ is 0. So, suppose otherwise. We transform $\mathcal{P}$ into a new p-document $\mathcal{P}'$ as follows. For each $n \in X$, we replace the tag of $\gamma(n)$ with a new unique tag that does not appear anywhere else in $\mathcal{P}'$. We also construct a new b-twig B that is identical to C, except for the following. For each node $n \in X$, we replace $cond_n(\cdot)$ with $tag(\cdot) = \mathtt{Y}$, where $\mathtt{Y}$ is the new unique tag of $\gamma(n)$. It is easy to show that there is a one-to-one correspondence between the random documents of $\mathcal{P}$ and those of $\mathcal{P}'$, such that probabilities (of documents) are preserved. (The only difference between two corresponding documents is in the tags of the nodes that are in the image of γ.) Moreover, there is a one-to-one correspondence between the c-matches $\varphi \in C(\mathcal{P})$ that satisfy $\varphi^{[X]} = \gamma$ and the c-matches of B in random documents of $\mathcal{P}'$, such that the following holds. If φ and φ' is a pair of corresponding c-matches defined over C and B, respectively, and d and d' is a pair of corresponding random documents of $\mathcal{P}$ and $\mathcal{P}'$, respectively, then $\varphi \in C(d)$ if and only if φ' is a c-match of B in d'. Thus, the following holds.

$$\Pr\left(\gamma \in \pi_X C(D[\mathcal{P}])\right) = \Pr\left(B(D[\mathcal{P}'])\right)$$

To conclude, an efficient algorithm for solving *EvalBoolean* gives rise to an efficient solution of *EvalComplete*. So we now consider the problem *EvalBoolean*.

4.2 Notation

We start with some definitions that are necessary for describing the algorithm for evaluating b-twigs. Consider a tree T. Let r be the root of T and c be one of its children. The subtree of T that consists of the edge (r, c) and the subtree T_Δ^c is called a *branch* of T and is denoted by $T_\Delta^{\frac{r}{c}}$. If n is a node of T, then each branch of T_Δ^n is a *sub-branch* of T. Note that a branch is a special case of a sub-branch, and so is an edge (m, n) of T, such that n is a leaf.

Consider a b-twig B and a new node n that is not in B. The b-twig n/B is obtained by making n the new root and adding a child edge from n to the root of B. The b-twig $n//B$ is constructed similarly, except for using a descendant edge to connect n and the root of B. We use $*$ to denote a node that matches everything, i.e., $cond_*(\cdot) = \mathbf{true}$.

We generalize b-twigs by combining them to make *expressions,* using the logical operators $\vee$, $\wedge$ and $\neg$. For example, the expression $B_1 \vee \neg B_2$ is evaluated over a document d by first evaluating $B_1(d)$ and $B_2(d)$, and then applying the logical operators. The following important property holds for

conjunctions of b-twigs. Given the b-twigs $B_1, \ldots, B_h$, we can construct a new b-twig B' by merging the roots of the B_i. The predicate of the new root is the conjunction of the root predicates of $B_1, \ldots, B_h$. It is easy to verify that the conjunction $B_1 \wedge \cdots \wedge B_h$ is equivalent to B'. Consequently, if B has a height of at least 1 and $R_1, \ldots, R_q$ ($q \geq 1$) are all its branches, then we define the conjunction $R_1 \wedge \cdots \wedge R_q$ as the *branch representation* of B.

4.3 The Algorithm

The input to the algorithm is a p-document $\mathcal{P}$ and a b-twig $\bar{B}$. The output is $\Pr\left(\bar{B}(D[\mathcal{P}])\right)$. At first, we construct the set $\mathcal{B}$ from $\bar{B}$ as follows. Note that $\bar{B} \in \mathcal{B}$.

$\mathcal{B}$ consists of all the conjunctions $B_1 \wedge \cdots \wedge, B_h$ ($h \geq 1$), such that each b-twig B_i is one of the following.

- A sub-branch of $\bar{B}$.
- A leaf of $\bar{B}$.
- $*//\bar{B}^n_\Delta$, where n is a non-root node of $\bar{B}$.

The algorithm computes $\Pr\left(B(D[\mathcal{P}^v_\Delta])\right)$ for every node v of $\mathcal{P}$ and for all b-twigs B of $\mathcal{B}$. However, if w is a distributional node of $\mathcal{P}$, then $\mathcal{P}^w_\Delta$ is not a p-document, because the root of a p-document must be an ordinary node. Therefore, the following equation defines the meaning of $B(D[\mathcal{P}^w_\Delta])$ in the case that w is a distributional node.

$$B\left(D\left[\mathcal{P}^w_\Delta\right]\right) \stackrel{\text{def}}{=} B\left(D\left[\mathcal{P}^{\circ}_w\right]\right) \tag{1}$$

The p-document in the right side of the above equation is obtained from $\mathcal{P}^w_\Delta$ by adding a new root, denoted by $\circ$, and making w the only child of that root. Intuitively, the ordinary node $\circ$ satisfies the root predicate of every b-twig. Formally, it means that when applying a b-twig B to a random document that has $\circ$ as its root, then we treat B as if its root is $*$.

The computation of $\Pr\left(B(D[\mathcal{P}^v_\Delta])\right)$ is done bottom-up by applying a dynamic-programming algorithm. Thus, when node v is reached, then for every child u of v and for all $B \in \mathcal{B}$, the probability $\Pr\left(B(D[\mathcal{P}^u_\Delta])\right)$ has already been computed. In the following sections, we describe how to compute $\Pr\left(B(D[\mathcal{P}^v_\Delta])\right)$ for $B \in \mathcal{B}$, when visiting node v.

Note that dynamic programming by itself is not sufficient for getting a practical algorithm. Roughly speaking, the reason is that each branch of $\mathcal{P}^v_\Delta$ may satisfy an arbitrary set of branches of B. Basically, our algorithm efficiently converts $\Pr\left(B(D[\mathcal{P}^u_\Delta])\right)$ into either a sum of probabilities of disjoint events or a product of probabilities of independent events, where these probabilities are computed in prior steps.

4.3.1 Three Simple Cases

We first describe how to compute $\Pr\left(B(D[\mathcal{P}^v_\Delta])\right)$ when we need not know the probabilities for the children of v.

Case 1. *Node v is a leaf (and, hence, ordinary).* In this case, v itself is the only random document. Therefore, $\Pr\left(B(D[\mathcal{P}^v_\Delta])\right) = 1$ if B consists of a single node n and v satisfies the predicate of n; otherwise, $\Pr\left(B(D[\mathcal{P}^v_\Delta])\right) = 0$.

Case 2. *Node v is ordinary and it does not satisfy the root predicate of B.* In this case, $\Pr\left(B(D[\mathcal{P}^v_\Delta])\right) = 0$.

Case 3. *B consists of a single node n.* If v is either a distributional node or an ordinary node that satisfies the predicate of n, then $\Pr\left(B(D[\mathcal{P}^v_\Delta])\right) = 1$. Otherwise, this is the same as the previous case, i.e., $\Pr\left(B(D[\mathcal{P}^v_\Delta])\right) = 0$.

In the remainder of the algorithm, we assume that none of the above three cases holds. In other words, neither B nor $\mathcal{P}^v_\Delta$ is a leaf, and if v is ordinary, then it satisfies the root predicate of B. We first deal with three different cases where v has a single child, and then reduce the case of multiple children to that of a single child.

4.3.2 Ordinary Node and One Distributional Child

Suppose that v is an ordinary node with a single child u_1, which is distributional. Since v satisfies the root predicate of B (otherwise, Case 2 holds), Equation (1) implies the following (when substituting u_1 for w in that equation).

$$\Pr\left(B(D[\mathcal{P}^v_\Delta])\right) = \Pr\left(B(D[\mathcal{P}^{u_1}_\Delta])\right)$$

Since u_1 is a child of v, the right side of the above equation has already been computed in a previous iteration.

4.3.3 Ordinary Node and One Ordinary Child

Suppose that both v and its only child u_1 are ordinary nodes. Let $R_1 \wedge \cdots \wedge R_q$ be the branch representation of B and r be the root of B. Let $B_1, \ldots, B_q$ be obtained from $R_1, \ldots, R_q$, respectively, by deleting the root r. If only child edges emanate from r, then

$$\Pr\left(B(D[\mathcal{P}^v_\Delta])\right) = \Pr\left(\bigwedge_{j=1}^{q} B_j(D[\mathcal{P}^{u_1}_\Delta])\right).$$

However, this equation does not necessarily hold if r has one or more outgoing descendant edges. To handle descendant edges, we reduce the computation to that of a conjunction of negated branches as follows.

Since the negation of B is $\neg R_1 \vee \cdots \vee \neg R_q$,

$$\Pr\left(B(D[\mathcal{P}^v_\Delta])\right) = 1 - \Pr\left(\bigvee_{j=1}^{q} \neg R_j(D[\mathcal{P}^v_\Delta])\right). \tag{2}$$

Using the principle of inclusion and exclusion, we can formulate $\Pr\left(\bigvee_{j=1}^{q} \neg R_j(D[\mathcal{P}^v_\Delta])\right)$ as a sum of probabilities of the following form, where J is a nonempty subset of $\{1, \ldots, q\}$.

$$\Pr\left(\bigwedge_{j \in J} \neg R_j(D[\mathcal{P}^v_\Delta])\right) \tag{3}$$

Suppose that in some branch R_j, a descendant edge connects the root n to its only child, that is, R_j is of the form $n//B'$. We define $R^1_j = n/B'$ and $R^2_j = n/*//B'$. Clearly,

$$\neg R_j(D[\mathcal{P}^v_\Delta]) \equiv \neg R^1_j(D[\mathcal{P}^v_\Delta]) \wedge \neg R^2_j(D[\mathcal{P}^v_\Delta]).$$

So, without loss of generality, we assume that in each branch R_j of (3), a child edge emanates from the root (or else we replace $\neg R_j$ with $\neg R^1_j \wedge \neg R^2_j$). Since v satisfies the root predicate of every R_j (otherwise, Case 2 holds), we get the following.

$$\Pr\left(\bigwedge_{j \in J} \neg R_j(D[\mathcal{P}^v_\Delta])\right) = \Pr\left(\bigwedge_{j \in J} \neg B_j(D[\mathcal{P}^{u_1}_\Delta])\right)$$

To compute the right side, we first use the equation:

$$\Pr\left(\bigwedge_{j \in J} \neg B_j(D[\mathcal{P}^{u_1}_\Delta])\right) = 1 - \Pr\left(\bigvee_{j \in J} B_j(D[\mathcal{P}^{u_1}_\Delta])\right) \tag{4}$$

Then, we apply the principle of inclusion and exclusion in order to convert $\Pr\left(\bigvee_{j\in J} B_j(D[\mathcal{P}^{u_1}_\Delta])\right)$ to a sum of probabilities of the form $\Pr\left(\bigwedge_{j\in J'} B_j(D[\mathcal{P}^{u_1}_\Delta])\right)$, where J' is a nonempty subset of J. Each probability in the sum has already been computed in a previous iteration, because u_1 is a child of v and every conjunction $\bigwedge_{j\in J'} B_j$ is in $\mathcal{B}$.

4.3.4 Distributional Node and One Child

Suppose that v is a distributional node that has a single child u_1. B has at least one child (otherwise, Case 3 holds) and, hence, it could be satisfied only if v chooses u_1, i.e., $(v,u_1)\in S[\mathcal{P}^v_\Delta]$. Thus, $\Pr\left(B(D[\mathcal{P}^v_\Delta])\right)$ is equal to[2]

$$\Pr\left(B(D[\mathcal{P}^v_\Delta]) \mid (v,u_1)\in S[\mathcal{P}^v_\Delta]\right)\mathcal{P}(v,u_1)\,.$$

If u_1 is distributional, then by Equation (1), we get that

$$\Pr\left(B(D[\mathcal{P}^v_\Delta]) \mid (v,u_1)\in S[\mathcal{P}^v_\Delta]\right) = \Pr\left(B(D[\mathcal{P}^{u_1}_\Delta])\right),$$

where the right side was computed in a previous iteration.

If u_1 is ordinary, then by Equation (1), we get that

$$\Pr\left(B(D[\mathcal{P}^v_\Delta]) \mid (v,u_1)\in S[\mathcal{P}^v_\Delta]\right) = \Pr\left(B\left(D\left[\mathcal{P}^{\circ}_{u_1}\right]\right)\right)$$

and we can compute the right side as in Section 4.3.3.

4.3.5 Multiple Children

In this section, we assume that v has $k>1$ children $u_1,\ldots,u_k$. Suppose first that v is an MXD node. B has at least two nodes (otherwise, Case 3 holds) and hence, it cannot be satisfied if v chooses none of its children. Moreover, the events $(v,u_i)\in S[\mathcal{P}^v_\Delta]$ $(i=1,\ldots,k)$ are disjoint. Thus,

$$\Pr\left(B(D[\mathcal{P}^v_\Delta])\right) = \sum_{i=1}^{k}\Pr\left(B\left(D\left[\mathcal{P}^{v}_{u_i}\right]\right)\right).$$

The probabilities of the sum are computed as in Section 4.3.4.

Next, suppose that v is either an ordinary or IDD node. The difficulty of computing $\Pr\left(B(D[\mathcal{P}^v_\Delta])\right)$ in this case is that two branches of the random document can satisfy the same branch of B (or, more generally, intersecting sets of branches of B). Therefore, we cannot apply a naive reduction to the previous two cases. In the following we again use the principle of inclusion and exclusion and, as a result, get expressions that are evaluated by considering each branch of the document separately.

We start by rewriting $\Pr\left(B\left(D\left[\mathcal{P}^v_\Delta\right]\right)\right)$ as described at the beginning of Section 4.3.3. In other words, we use Equation 2 (where $R_1,\ldots,R_q$ is the branch representation of B), and then apply the principle of inclusion and exclusion to convert the probability on the right side of (2) into a sum of probabilities of the form shown in (3). The negated branch $\neg R_j$ $(j\in J)$ is satisfied in a document d if and only if it is satisfied in every branch of d. (In contrast to R_j that only has to be satisfied in one branch of d.) Since we assume that v is either an ordinary or IDD node, distinct branches of $D[\mathcal{P}^v_\Delta]$ are probabilistically independent. Therefore,

$$\Pr\left(\bigwedge_{j\in J}\neg R_j\left(D\left[\mathcal{P}^v_\Delta\right]\right)\right) = \prod_{i=1}^{k}\Pr\left(\bigwedge_{j\in J}\neg R_j\left(D\left[\mathcal{P}^{v}_{u_i}\right]\right)\right).$$

[2]Recall that $\mathcal{P}(v,u_1)$ is the probability that $(v,u_1)\in S[\mathcal{P}^v_\Delta]$.

Consequently, it suffices to compute the probability for each branch of $\mathcal{P}^v_\Delta$. So, consider the branch that includes u_i. If both v and u_i are ordinary nodes, then we continue exactly as described in Section 4.3.3 below (3). Otherwise, similarly to the last part of Section 4.3.3 (that starts at Equation (4)), we use the equation

$$\Pr\left(\bigwedge_{j\in J}\neg R_j\left(D\left[\mathcal{P}^{v}_{u_i}\right]\right)\right) = 1-\Pr\left(\bigvee_{j\in J} R_j\left(D\left[\mathcal{P}^{v}_{u_i}\right]\right)\right)$$

and the principle of inclusion and exclusion, so that the probabilities to be computed are of the following form (where J' is a nonempty subset of J).

$$\Pr\left(\bigwedge_{j\in J'} R_j\left(D\left[\mathcal{P}^{v}_{u_i}\right]\right)\right)$$

Since each conjunction $\bigwedge_{j\in J'} R_j$ is in $\mathcal{B}$, we can now proceed as in either Section 4.3.2 or Section 4.3.4, depending on whether v is ordinary or distributional.

4.3.6 Correctness

LEMMA 4.1. *The algorithm correctly computes the probability* $\Pr\left(B(D[\mathcal{P}^v_\Delta])\right)$ *for all b-twigs* $B\in\mathcal{B}$ *and nodes* $v\in\mathcal{V}(\mathcal{P})$*, in polynomial time under data complexity.*

Our algorithm shows that determining $\Pr\left(B(D[\mathcal{P}^v_\Delta])\right)$ is in fact *parametrically tractable* [22], that is, the size of the query does not affect the degree of the polynomial, only its coefficients. The above lemma and the reduction of Section 4.1 imply the following.

THEOREM 4.2. *Both EvalComplete and EvalBoolean can be solved in polynomial time, under data complexity.*

4.4 Query-and-Data Complexity

We now consider the query-and-data complexity of *EvalComplete* and *EvalBoolean*. Note that, under this measure, the output of *EvalComplete* may be exponentially larger than the input. In this case, an evaluation algorithm is efficient if it runs in *polynomial total time* [16], that is, polynomial time in the combined size of the input and the output. Under query-and-data complexity, both problems become intractable. Furthermore, these problems remain intractable even under strong simplifying assumptions.

THEOREM 4.3. *Under query-and-data complexity:*

- *Computing* $\Pr\left(B(D[\mathcal{P}])\right)$ *is #P-complete.*
- *Determining whether* $\pi^{\uparrow p}_X C(\mathcal{P})\neq\emptyset$ *is* $P^{\#P}$*-hard.*
- *Determining whether* $C^{\uparrow p}(\mathcal{P})\neq\emptyset$ *is NP-complete.*

Furthermore, these results hold even if (1) *there are no MXD nodes,* (2) *distributions are only over leaves,* (3) *all the probabilities are 0.5,* (4) *the twig has only child edges,* and (4) *the out degree of each node of the twig is at most 2.*

Recall that $\mathrm{P}^{\#\mathrm{P}}$ is the class of problems that can be solved by polynomial-time machines that have an oracle to some #P-complete problem (e.g., the number of satisfying assignments of a CNF formula). Note that this class contains the whole polynomial hierarchy [26]. Containment in #P is shown rather straightforwardly by an adaptation of the techniques of [13]. For showing #P-hardness, we use a reduction from #Set-Cover which is known to be #P-complete [23].

5. EVALUATING I-TWIGS

In the previous section, we showed that the evaluation of c-twigs is tractable only under data complexity. In this section, we consider i-twigs and prove that their evaluation is tractable even under query-and-data complexity.

We use two notions of efficiency of enumeration algorithms which are stronger than polynomial total time and relate the time that is needed for generating the ith result (which is an i-match in this case), after the first $i-1$ have already been created. *Incremental polynomial time* means that the ith i-match is generated in time that is polynomial in the combined size of the input and the first $i-1$ i-matches. The stronger notion of *polynomial delay* means that there is a polynomial delay between the output of every two consecutive i-matches. The algorithm that we present in this section enumerates the maximal i-matches in incremental polynomial time and is a reduction to a simplified problem which is discussed first.

5.1 Finding Maximal Subsumed Matches

We consider the problem of finding all the maximal i-matches among those that are subsumed by a given i-match φ and have a probability of at least p. Formally, suppose that $\mathcal{P}$ is a p-document, I is an i-twig and φ is an i-match of I in a random document of $\mathcal{P}$, i.e., $\varphi \in I(\mathcal{P})$. We denote by $I_{\sqsubseteq\varphi}(\mathcal{P})$ the set of all i-matches $\gamma \in I(\mathcal{P})$, such that γ is subsumed by φ. For a threshold $p \in [0,1]$, the set $I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P})$ consists of all the i-matches of $I_{\sqsubseteq\varphi}(\mathcal{P})$ with a probability of at least p, that is,

$$I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P}) \stackrel{\text{def}}{=} \{\gamma \in I(\mathcal{P}) \mid \gamma \sqsubseteq \varphi \wedge \Pr(\gamma \in I(D[\mathcal{P}])) \geq p\}.$$

The goal is to find $\max(I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P}))$, that is, to enumerate all i-matches $\gamma \in I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P})$, such that no i-match of $I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P})$ properly subsumes γ. The algorithm SubMatches of Figure 3 solves this problem. Next, we describe it.

Let $\varphi \in I(\mathcal{P})$. Recall that $\mathcal{P}^{|\,Img(\varphi)}$ denotes the r-subtree of $\mathcal{P}$ that is reduced w.r.t. the image of φ (see the definition in Section 2.1). In this section, we denote $\mathcal{P}^{|\,Img(\varphi)}$ by $\mathcal{P}_\varphi$.

The input of the algorithm SubMatches is a p-document $\mathcal{P}$, an i-twig I, an i-match $\varphi \in I(\mathcal{P})$, a threshold p and a set U of nodes, which is initially empty. The algorithm finds all i-matches $\varphi' \in \max(I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P}))$ that satisfy the following constraint. The reduced r-subtree $\mathcal{P}_{\varphi'}$ must include all the nodes of U.

We define the *accumulated probability* along a path in $\mathcal{P}$ as the product of all values $\mathcal{P}(v,u)$, where v is a distributional node and (v,u) is an edge on the path. (As a special case, note that an empty product is equal to 1.) Let v be a node of $\mathcal{P}_\varphi$. The *conditional connection cost* of v, denoted by $cCost(v)$, is the maximum of all accumulated probabilities along paths from v to a (not necessarily proper) descendant of v that is in $Img(\varphi)$. Note that $cCost(v) = 1$ if $v \in Img(\varphi)$.

Now, we describe the algorithm in detail. Lines 2–3 and Lines 4–6 deal with two special cases as follows. Line 2 tests whether $\mathcal{P}_\varphi$ contains U. If not (which may only occur for recursive calls but not for the initial one), then the output is empty and the algorithm terminates in Line 3. Line 4 tests whether the probability of φ is at least p. If so, then $\max(I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P})) = \{\varphi\}$ and, hence, the algorithm prints φ in Line 5 and terminates in Line 6. Note that $\Pr(\varphi \in I(D[\mathcal{P}]))$ is the same as the probability that $\mathcal{P}_\varphi$ is an r-subtree of $S[\mathcal{P}]$ (denoted by $\mathcal{P}_\varphi \preceq_r S[\mathcal{P}]$). Thus, $\Pr(\varphi \in I(D[\mathcal{P}]))$ is equal to the product of the probabilities $\mathcal{P}(v,u)$, where (v,u) is an edge of $\mathcal{P}_\varphi$ and v is distributional.

Algorithm SubMatches$(\mathcal{P}, I, \varphi, p, U)$

1: $\mathcal{P}_\varphi \leftarrow \mathcal{P}^{|\,Img(\varphi)}$
2: **if** $U \not\subseteq \mathcal{V}(\mathcal{P}_\varphi)$ **then**
3: **return**
4: **if** $\Pr(\varphi \in I(D[\mathcal{P}])) \geq p$ **then**
5: **output**(φ)
6: **return**
7: **for all** nodes $v \in \mathcal{V}(\mathcal{P}_\varphi) \setminus U$ **do**
8: $c[v] \leftarrow cCost(v)$
9: **if** v has a distributional parent u **then**
10: $c[v] \leftarrow c[v] \cdot \mathcal{P}(u,v)$
11: $w \leftarrow \text{argmin}_v \{c[v] \mid v \in \mathcal{V}(\mathcal{P}_\varphi) \setminus U\}$
12: $X \leftarrow \{n \in Dom(\varphi) \mid \varphi(n)$ is not a descendant of $w\}$
13: $\gamma \leftarrow \varphi^{[X]}$
14: **if** $\Pr(\gamma \in I(D[\mathcal{P}])) \cdot c[w] < p$ **then**
15: SubMatches$(\mathcal{P}, I, \gamma, p, U)$
16: $q \leftarrow \Pr\left(\mathcal{P}^{|U\cup\{w\}} \preceq S[\mathcal{P}]\right)$
17: **for all** leaves u of $\mathcal{P}^{|U\cup\{w\}}$ **do**
18: $q \leftarrow q \cdot cCost(u)$
19: **if** $q \geq p$ **then**
20: SubMatches$(\mathcal{P}, I, \varphi, p, U \cup \{w\})$

Figure 3: Computing $\max(I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P}))$

The main part of the algorithm (Lines 7–20) chooses a node $w \in \mathcal{V}(\mathcal{P}_\varphi)$ and divides the enumeration into two recursive calls. Line 15 and Line 20 enumerate all i-matches φ' of $\max(I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P}))$ such that $w \notin \mathcal{V}(\mathcal{P}_{\varphi'})$ and $w \in \mathcal{V}(\mathcal{P}_{\varphi'})$, respectively. The arguments of the recursive call in Line 20 are the original φ and a new set, namely, $U \cup \{w\}$. (Of course, the original $\mathcal{P}$, I and p are the remaining arguments.) In Line 15, the arguments are the original set U and a new i-match γ, where γ is the maximal i-match that satisfies $\gamma \sqsubseteq \varphi$ and $w \notin \mathcal{V}(\mathcal{P}_\gamma)$. (Clearly, γ is unique.) It is easy to show the following two facts. First, the same i-match cannot be enumerated by both recursive calls. Second, the two recursive calls together enumerate all the i-matches of $\max(I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P}))$. However, the recursive call in Line 15 may also enumerate some i-matches that do not belong to $\max(I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P}))$ if we choose node w arbitrarily. But this problem cannot occur because the particular choice of w and the test of Line 14 guarantee that $\max(I^{\uparrow p}_{\sqsubseteq\gamma}(\mathcal{P})) \subseteq \max(I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P}))$. Furthermore, if the test of Line 14 is **false**, then w appears in $\mathcal{P}_{\varphi'}$ for all i-matches φ' of $\max(I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P}))$, such that $\mathcal{V}(\mathcal{P}_{\varphi'})$ contains U. Therefore, none of these i-matches is in $\max(I^{\uparrow p}_{\sqsubseteq\gamma}(\mathcal{P}))$ and, hence, only the recursive call of Line 20 has to be executed. The proof is omitted due to lack of space.

Line 11 chooses w to be the node that has the minimum value in the array c, where c is computed in Lines 7–10 for all nodes of $\mathcal{V}(\mathcal{P}_\varphi) \setminus U$ as follows. If v has a distributional parent u, then $c[v]$ is the product of $cCost(v)$ and $\mathcal{P}(u,v)$. Otherwise, it is just $cCost(v)$. Line 12 finds the domain X of γ and Line 13 computes γ by applying projection onto X to φ.

To guarantee efficiency, it is crucial not to execute the recursive call of Line 20 if it returns an empty result. So,

Line 19 tests whether there exists an i-match $\psi \sqsubseteq \varphi$, with a probability of at least p, such that the r-subtree of $Img(\psi)$ contains $U \cup \{w\}$. This test is just $q \geq p$, where q is computed in Lines 16–18 as the product of the following numbers. (Recall that $\mathcal{P}^{|U\cup\{w\}}$ is the r-subtree of $\mathcal{P}$ that is reduced w.r.t. $U \cup \{w\}$).

- The probability that $U \cup \{w\}$ is contained in a random sample of $\mathcal{P}$, i.e., $\Pr\left(\mathcal{P}^{|U\cup\{w\}} \preceq S[\mathcal{P}]\right)$.
- $cCost(u)$ for all leaves u of $\mathcal{P}^{|U\cup\{w\}}$.

The following lemma shows the correctness and efficiency of SubMatches.

Lemma 5.1. SubMatches$(\mathcal{P}, I, \varphi, p, U)$ *enumerates all i-matches* $\gamma \in \max(I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P}))$*, such that* $Img(\gamma)$ *contains a descendant of* u *for each* $u \in U$*. It runs with polynomial delay.*

5.2 The Main Algorithm

We now describe the algorithm EvalITwig of Figure 4 for enumerating $\max(I^{\uparrow p}(\mathcal{P}))$. This algorithm is a reduction to the problem solved in the previous section, namely, enumerating $\max(I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P}))$; it is an adaptation of a general approach for enumerating maximal subgraphs that satisfy some connected hereditary property [5, 4, 20].

We start with some definitions. Consider a p-document $\mathcal{P}$, an i-twig I and a threshold $p \in [0, 1]$. Let γ be an i-match of $I(\mathcal{P})$ and let $n \mapsto v$ be a particular assignment that maps node n of I to node v of $\mathcal{P}$, where v is ordinary and neither n nor v is a root. We say that γ and $n \mapsto v$ are *partially consistent* if the following conditions hold.

- Node v satisfies the node predicate of n.
- The parent m of n is in $Dom(\gamma)$, i.e., $\gamma(m)$ is defined.
- If (m, n) is a child edge of I, then $\gamma(m)$ is the parent of v; otherwise, $\gamma(m)$ is a proper ancestor of v.

We say that γ and $n \mapsto v$ *violate mutual exclusion* if the following holds. There is a node $n' \in Dom(\gamma)$, such that $\gamma(n')$ and v belong to different branches of some MXD node of $\mathcal{P}$. Finally, we say that γ and $n \mapsto v$ are *fully consistent* if they are partially consistent and do not violate mutual exclusion.

The algorithm EvalITwig of Figure 4 accepts as input a p-document $\mathcal{P}$, an i-twig I and a threshold $p \in [0, 1]$. It uses two subroutines that we now describe. The first one, MaxExtend$(\mathcal{P}, I, \psi, p)$, accepts an i-match $\psi \in I^{\uparrow p}(\mathcal{P})$ and extends ψ to an i-match of $\max(I^{\uparrow p}(\mathcal{P}))$ by applying the following greedy process. We arbitrarily choose a node n of I, such that $n \notin Dom(\psi)$, and an ordinary node v of $\mathcal{P}$. Let $\psi^{n\mapsto v}$ denote the extension of ψ that maps the nodes of $Dom(\psi)$ as in ψ and maps n to v. If ψ and $n \mapsto v$ are fully consistent and $\Pr\left(\psi^{n\mapsto v} \in I(D[\mathcal{P}])\right) \geq p$, then we permanently replace ψ with $\psi^{n\mapsto v}$ and continue the greedy process; otherwise, we continue the greedy process with ψ by trying another assignment $n' \mapsto v'$. The algorithm terminates when ψ cannot be extended by any assignment.

The second subroutine, Merge$(\mathcal{P}, I, \gamma, n, v)$, *merges* an i-match $\gamma \in \max(I^{\uparrow p}(\mathcal{P}))$ with the assignment $n \mapsto v$, where $n \in \mathcal{V}(I)$, $v \in \mathcal{V}^{ord}(\mathcal{P})$ and neither n nor v is a root. The result of this merging is an i-match that maps n to v and maps the other nodes in its domain as γ. Note that the result could be $\perp$, that is, the i-match that is defined on the empty domain. Merge$(\mathcal{P}, I, \gamma, n, v)$ starts by testing whether γ and $n \mapsto v$ are partially consistent. If they are not, then the result is $\perp$; otherwise, the following is done. Let $X = Dom(\gamma)$. We remove from X every n', such that $\gamma(n')$ and v are in different branches of an MXD node of $\mathcal{P}$. We also remove all the descendants of n. The result of the merge operation is the i-match that maps every node of (the final value of) X as γ and maps n to v.

Algorithm EvalITwig$(\mathcal{P}, I, p)$

```
 1: Printed, ToExtend ← empty balanced search trees
 2: φ0 ← MaxExtend(P, I, {root(I) ↦ root(P)}, p)
 3: output(φ0)
 4: Printed.Insert(φ0)
 5: ToExtend.Insert(φ0)
 6: while ToExtend ≠ ∅ do
 7:   γ ← ToExtend.Remove()
 8:   for all nodes v ∈ V^ord(P) \ root(P) do
 9:     for all nodes n ∈ V(I) \ {root(I)} do
10:       φ ← Merge(P, I, γ, n, v)
11:       if φ ≠ ⊥ then
12:         for all ψ ∈ SubMatches(P, I, φ, p, ∅) do
13:           ψ ← MaxExtend(P, I, ψ, p)
14:           if ψ ∉ Printed then
15:             output(ψ)
16:             Printed.Insert(ψ)
17:             ToExtend.Insert(ψ)
```

Figure 4: Computing $\max(I^{\uparrow p}(\mathcal{P}))$

The algorithm EvalITwig of Figure 4 uses two balanced search trees for storing results. *Printed* stores the results that have been printed and *ToExtend* stores the results that need to be processed in Lines 6–17 (as describe below). Initially, the algorithm generates an arbitrary i-match $\varphi_0 \in \max(I^{\uparrow p}(\mathcal{P}))$ by maximizing the i-match that maps the root of I to that of $\mathcal{P}$. (We assume that $root(\mathcal{P})$ satisfies the predicate of $root(I)$, or else there are no i-matches.) In Lines 3–5, φ_0 is printed and stored in both *Printed* and *ToExtend*.

The loop of Line 6 is repeated as long as *ToExtend* is nonempty. In each iteration, Line 7 removes an i-match γ from *ToExtend*. In the two nested loops of Lines 8–9, the i-match γ is merged with every assignment $n \mapsto v$, such that $n \in \mathcal{V}(I)$, $v \in \mathcal{V}^{ord}(\mathcal{P})$ and neither n nor v is a root. If the merged i-matched φ is $\perp$, then the algorithm continues to the next iteration of the nested loops. Otherwise, Lines 13–17 are repeated for all i-matches $\psi \in \max(I^{\uparrow p}_{\sqsubseteq\varphi}(\mathcal{P}))$. The i-matches ψ are obtained by executing the algorithm SubMatches of Figure 3 (with the empty set as U). Each ψ is maximized in Line 13 and if it has just been generated for the first time (i.e., it is not in *Printed*), then it is sent to the output and inserted into both *Printed* and *ToExtend* (Lines 15–17). The following theorem shows the correctness of EvalITwig.

Theorem 5.2. *The algorithm* EvalITwig$(\mathcal{P}, I, p)$ *enumerates* $\max(I^{\uparrow p}(\mathcal{P}))$ *in incremental polynomial time, under query-and-data complexity.*

Corollary 5.3. *Under query-and-data complexity, Eval-Maximal can be solved in incremental polynomial time.*

5.3 Extensions and Intractability Results

5.3.1 Projection

In this section, we discuss i-twigs with projections. Consider an i-twig I and a set X of nodes of I. We use $\varphi^{[X]}$ to denote the projection of the i-match φ onto the nodes of $X \cap Dom(\varphi)$. The definitions of $\pi_X\, I(\mathcal{P})$ and $\pi_X^{\uparrow p}\, I(\mathcal{P})$ are analogous to the corresponding definitions in the case of the complete semantics (see Section 3.2.1). The goal is to find the maximal matches after projection, that is, the set $\max(\pi_X^{\uparrow p}\, I(\mathcal{P}))$ and the probabilities of its elements.

The probability of a mapping $\varphi^{[X]} \in \pi_X\, I(\mathcal{P})$ is the same as the probability that a random document contains the image of $\varphi^{[X]}$. So, it can be computed as described in Section 3.2.2. This is true because a leaf of I that is not in X has no effect on the query and, hence, can be removed. In fact, instead of using I itself, we can transform it into a smaller i-twig by removing every node that has no descendant (including itself) in X. Therefore, we get the following.

PROPOSITION 5.4. *The set* $\max(\pi_X^{\uparrow p}\, I(\mathcal{P}))$ *and the probabilities of its elements can be computed in polynomial time under data complexity.*

It is unknown whether $\max(\pi_X^{\uparrow p}\, I(\mathcal{P}))$ can be enumerated efficiently under query-and-data complexity. Nevertheless, the following theorem shows that we can compute $\max(\pi_X^{\uparrow p}\, I(\mathcal{P}))$ by first executing EVALITWIG$(\mathcal{P}, I, p)$ and then applying the projection followed by elimination of subsumed i-matches. (Note that $\varphi_1^{[X]}$ may be subsumed by $\varphi_2^{[X]}$ even if φ_1 is maximal.)

THEOREM 5.5. *Consider a p-document* $\mathcal{P}$*, an i-twig* I*, a set of nodes* $X \subseteq \mathcal{V}(I)$ *and a threshold* $p \in [0, 1]$*. Then*

$$\max\left(\pi_X\left(\max(I^{\uparrow p}(\mathcal{P}))\right)\right) = \max\left(\pi_X^{\uparrow p}\, I(\mathcal{P})\right).$$

Note that if I is replaced with a c-twig,[3] then the left-hand side is contained in the right-hand side but equality does not necessarily hold, because the probability of a projection of a c-match could be determined by multiple, dependent c-matches. However, in the case of i-twigs, if a projection is in $\max(\pi_X^{\uparrow p}\, I(\mathcal{P}))$, then there is at least one i-match in $\max(I^{\uparrow p}(\mathcal{P}))$ that subsumes that projection.

5.3.2 Completeness Constraints

We often want to restrict the degree of incompleteness by requiring that some specific nodes of the query are always matched. This requirement is extended in [19] to *completeness constraints*, that is, constraints of the following form. For an edge (n, m) of the i-twig I, if n is in the domain of the result, then m must also be in that domain. In [19], it is shown that such constraints can be efficiently supported in ordinary data. Generally, in the case of p-documents, i-twigs with these types of constraints cannot be evaluated efficiently under query-and-data complexity, since c-twigs are a special case of i-twigs with these constraints.

An important observation is the following one. While it is impossible to support completeness constraints as defined in [19], supporting the following type of constraints can be achieved by extending the algorithm EVALITWIG (without changing its complexity). For an edge (n, m) of I, if n is in the domain of an i-match φ, then m must also be in the domain of φ *and* the path in $\mathcal{P}$ from $\varphi(n)$ to $\varphi(m)$ must consist of only ordinary nodes (that is, should not include distributional nodes).

[3] In this case, max is the identity operator.

5.3.3 Ranked Enumeration

Maximal results of queries over probabilistic data may largely differ in their relevancy to the user. Therefore, it is desirable to assess a *score* for each result and output the results in *ranked order*, i.e., in descending order of score. Two measures are likely to determine the semantic strength of a result: *uncertainty* and *incompleteness*. In the first case, the score of a result is proportional to its probability, whereas in the second case, the score is proportional to the size of its domain.

The algorithm EVALITWIG of Figure 4 enumerates the results in an arbitrary order. Naturally, enumeration in ranked order is more desirable. Note that under data complexity, ranking does not increase the complexity of evaluation, since one can sort the results prior to presenting them the user. However, this approach cannot yield enumeration in incremental polynomial time (or with polynomial delay) under query-and-data complexity. In [19], it is shown that in the case of ordinary documents, one can enumerate the maximal matches by increasing amount of incompleteness, with polynomial delay under query-and-data complexity (even if additional ranking functions are involved). A similar result, however, cannot be realized in the case of probabilistic documents, as shown by the following theorem.

THEOREM 5.6. *For the two scoring measures described above, an algorithm that runs in incremental polynomial time, under query-and-data complexity, cannot solve Eval-Maximal by enumerating* $\max(I^{\uparrow p}(\mathcal{P}))$ *in ranked order, unless P=NP.*

6. RELATED WORK

Probabilistic data models for XML are described in [1, 25, 14, 15, 21]. In [14, 15], semistructured data are viewed as acyclic directed graphs where probabilities (or intervals of probabilities [14]) are defined over sets of children. Our p-documents are similar to those of [21]. In [1], two specific models of probabilistic XML are presented. The "simple probabilistic trees" (SP trees) are a restriction of our p-documents to independent distributions, whereas the "fuzzy trees" generalize our data model by associating with each node a conjunction of atomic events (which may include negation). Theoretic aspects of the fuzzy-tree model are studied in [25]. Interestingly, there is an efficient translation of an SP tree into a p-document and of a p-document into a fuzzy tree.

Neither projection (which includes Boolean queries as a special case) nor incompleteness has been studied in any probabilistic XML model. Thus, in the above models, query evaluation reduces to *(1)* evaluating the query as if the document is ordinary (i.e., without probabilities), *and (2)* evaluating for each match, the probability that its image occurs in a random document (as discussed in Section 3.2.2 just before Example 3.7).[4] Note that this straightforward approach cannot handle projection. It can be used for finding

[4] In [21], there is an additional operation that groups multiple results that differ only in the values of the leaves.

the probability of a partial answer, but it would entail finding all partial answers prior to selecting the maximal ones.

The fuzzy trees are more general than our model, but the tractability results of [25] are only for queries without projections. We can show that it is #P-hard, under data complexity, to evaluate twig queries with projections over fuzzy trees (even under strong restrictions on the queries and data[5]). Furthermore, it is NP-hard, under query-and-data complexity, to find the maximal answers to twig queries (without projections) over fuzzy trees.[6] We can show similar intractability results for the DAG models of [14, 15].

Probabilistic models for relational databases are investigated in [2, 3, 7, 8, 9, 11, 20]. These models substantially differ from our model, due to the conditional nature of probabilistic XML (where the random process of generating data is hierarchal). The work of [9, 10] shows that projection significantly increases the data complexity—even projections of very simple (tree) queries become intractable. As a simple example, [9] shows that any b-twig that consists of a single path with at least four nodes is #P-hard when the data form a graph (rather than a tree). In contrast, we have shown that all twigs with projections have a tractable data complexity in our model. Hence, our results do not follow from those of [9, 10].[7]

Maximal answers to queries over ordinary data have been investigated in [4, 5, 12, 17, 18, 24, 20]. Similarly to [5, 4, 20], we adapt a general approach that can be abstractly described as enumerating maximal subgraphs that satisfy some connected hereditary property. This adaptation enables us to reduce the problem to the one that is solved in Section 5.1. The only other work that deals with maximal answers in the context of probabilistic data is [20]. However, [20] is couched in the relational model which is not hierarchical, there is no root and a maximal answer has at most one tuple from each relation. Therefore, the results of this paper do not follow from those of [20] and, more importantly, even the problem itself cannot be phrased in the framework of [20].

7. CONCLUSION

The large variety of roles that XML has in practice necessitates several semantics for query evaluation. We have studied the evaluation of twig queries over probabilistic XML under three semantics, namely, complete, incomplete and Boolean. In comparison to previous work on probabilistic XML [1, 14, 15, 21], we have considered c-twigs with projections. Supporting projection substantially complicates query evaluation, because the probability of a result is derived from multiple, highly dependent events, namely, the results of the sub-twigs that are projected out. We have presented an algorithm for evaluating queries under the complete and Boolean semantics. Our algorithm is efficient under data complexity. In comparison, very simple queries are highly intractable (under data complexity) in the probabilistic relational models of [9, 10], when projection is allowed. We have shown that, under query-and-data complexity, query evaluation is intractable, even under strong simplifying assumptions (in particular, even if there is no projection). For the incomplete semantics, we have given an algorithm that is efficient under query-and-data complexity; for i-twigs with projections, the evaluation problem is tractable under data complexity.

[5]Namely, each node of the document consists of one positive (not necessarily distinct) event and the query consists of only two child edges.

[6]It is in polynomial time under data complexity.

[7]Moreover, [9, 10] do not consider mutually exclusive distributions and it is not clear how to express descendant edges in their framework.

Acknowledgments

The authors thank Yuri Kosharovsky and the anonymous referees for helpful comments and suggestions.

8. REFERENCES

[1] S. Abiteboul and P. Senellart. Querying and updating probabilistic information in XML. In *EDBT*, 2006.
[2] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 4(5), 1992.
[3] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In P. M. Stocker, W. Kent, and P. Hammersley, editors, *VLDB*, 1987.
[4] S. Cohen, I. Fadida, Y. Kanza, B. Kimelfeld, and Y. Sagiv. Full disjunctions: Polynomial-delay iterators in action. In *VLDB*, 2006.
[5] S. Cohen and Y. Sagiv. An incremental algorithm for computing ranked full disjunctions. In *PODS*, 2005.
[6] S. Cohen and Y. Sagiv. An incremental algorithm for computing ranked full disjunctions. To appear in *J. Comput. Syst. Sci.* (an extended version of [5]), 2006.
[7] N. N. Dalvi. Query evaluation on a database given by a random graph. In *ICDT*, 2007.
[8] N. N. Dalvi, G. Miklau, and D. Suciu. Asymptotic conditional probabilities for conjunctive queries. In *ICDT*, 2005.
[9] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.
[10] N. N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, 2007.
[11] D. Dey and S. Sarkar. A probabilistic relational model and algebra. *ACM Trans. Database Syst.*, 21(3), 1996.
[12] C. A. Galindo-Legaria. Outerjoins as disjunctions. In *SIGMOD*, 1994.
[13] E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *PODS*. ACM Press, 1998.
[14] E. Hung, L. Getoor, and V. S. Subrahmanian. Probabilistic interval XML. In *ICDT*, 2003.
[15] E. Hung, L. Getoor, and V. S. Subrahmanian. PXML: A probabilistic semistructured data model and algebra. In *ICDE*, 2003.
[16] D. Johnson, M. Yannakakis, and C. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27, 1988.
[17] Y. Kanza, W. Nutt, and Y. Sagiv. Querying incomplete information in semistructured data. *J. Comput. Syst. Sci.*, 64(3), 2002.
[18] Y. Kanza and Y. Sagiv. Computing full disjunctions. In *PODS*. ACM, 2003.
[19] B. Kimelfeld and Y. Sagiv. Combining incompleteness and ranking in tree queries. In *ICDT*, 2007.
[20] B. Kimelfeld and Y. Sagiv. Maximally joining probabilistic data. In *PODS*, 2007.
[21] A. Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *VLDB*, 2002.
[22] C. H. Papadimitriou and M. Yannakakis. On the complexity of database queries. In *PODS*. ACM, 1997.
[23] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4), 1983.
[24] A. Rajaraman and J. D. Ullman. Integrating information by outerjoins and full disjunctions. In *PODS*, 1996.
[25] P. Senellart and S. Abiteboul. On the complexity of managing probabilistic XML data. In *PODS*, 2007.
[26] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 21(2), 1992.