

Query Ranking in Probabilistic XML Data

Lijun Chang, Jeffrey Xu Yu, Lu Qin

The Chinese University of Hong Kong, Hong Kong, China
{ljchang, yu, lqin}@se.cuhk.edu.hk

ABSTRACT

Twig queries have been extensively studied as a major fragment of XPATH queries to query XML data. In this paper, we study PXML-RANK query, (Q, k) , which is to rank top- k probabilities of the answers of a twig query Q in probabilistic XML (PXML) data. A new research issue is how to compute top- k probabilities of answers of a twig query Q in PXML in the presence of containment (ancestor/descendant) relationships. In the presence of the ancestor/descendant relationships, the existing dynamic programming approaches to rank top- k probabilities over a set of tuples cannot be directly applied, because any node/edge in PXML may have impacts on the top- k probabilities of answers. We propose new algorithms to compute PXML-RANK queries efficiently and give conditions under which a PXML-RANK query can be processed efficiently without enumeration of all the possible worlds. We conduct extensive performance studies using both real and large benchmark datasets, and confirm the efficiency of our algorithms.

1. INTRODUCTION

Probabilistic XML (PXML) have been extensively studied recently [16, 11, 12, 22, 1, 20, 14, 5]. The issues studied widely cover the PXML models, semantics, data integration, constraints, expressiveness, query evaluation, query tractability, and complexity analysis. In this paper, we study a new research issue, and we study PXML rank query, (Q, k) , which is to rank top- k probabilities of the answers of a twig query Q in PXML data. The new challenging is how to compute top- k probabilities of answers of a twig query Q in PXML in the presence of containment (ancestor/descendant) relationships, where an answer of a twig query can be judged using any score function as studied in [7, 19, 2]. In the presence of the ancestor/descendant relationships, the existing dynamic programming approaches [23, 24, 9, 10] to compute top- k probabilities over a set of tuples cannot be directly applied, because in the context of PXML any node/edge may possibly have impacts on the top- k probabilities of answers. To the best of our knowledge, it is the first work which studies ranking of twig query results in the context of PXML.

We study three types of PXML-RANK queries, (Q, k) , where Q

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the ACM. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permissions from the publisher, ACM. *EDBT 2009*, March 24–26, 2009, Saint Petersburg, Russia. Copyright 2009 ACM 978-1-60558-422-5/09/0003 ...\$5.00

can be a node query ($//A$), a path query ($//A//B$), or a tree query ($//A[.//C]//B$). The main contributions of this work are summarized below. First, we focus on node queries, and propose a new dynamic programming algorithm which can compute top- k probabilities for the answers of node queries based on the previously computed results in PXML data. Our algorithm considers the containment issue (ancestor/descendant) as well as the top- k probability and the score ranking (score functions) issues. We further propose optimization techniques to share the computational cost. Second, we show that our techniques can be used to support any path queries, and certain tree queries efficiently without enumerating all the possible worlds. We give conditions on the tree queries, and discuss our approaches. Third, we conduct extensive performance studies using both real and large benchmark datasets, and confirm the efficiency of our algorithms.

The remainder of the paper is organized as follows. Section 2 reviews the definition of probabilistic XML, and gives our problem statement. In Section 3, we discuss the technique details of answering a PXML-RANK node query, and in Section 4, we discuss how to extend the algorithms of node query to process all path queries and certain tree queries. Experimental studies are given in Section 5, followed by discussions on related work in Section 6. Finally, we conclude the paper in Section 7.

2. PXML AND PXML-RANK

An XML document can be modeled as a rooted, unordered, and node-labeled tree, $T_X(V_X, E_X)$, where V_X represents a set of XML elements (nodes), and E_X represents a set of parent/child relationships (edges) between elements in XML. In an XML tree, a node is associated with a value x_i which belongs to a type (tag-name) X , denoted as $x_i \in X$. An XML tree is weighted if nodes and edges in the XML tree, $T_X(V_X, E_X)$, are associated with non-negative weights, denoted as $w_v(v)$ for $v \in V_X$ and $w_e(e)$ for $e \in E_X$, respectively. In the following, an XML tree is a weighted XML tree unless otherwise specified.

A probabilistic XML (or PXML for short) defines a probability distribution over XML trees. Following the model given in [16], which is the $\text{PrXML}^{\{\text{ind}, \text{mux}\}}$ model in [14], in this paper, we define a PXML, $T_P(V_P, E_P)$, over a weighted XML tree $T_X(V_X, E_X)$. Here, V_P is a set of nodes $V_P = V_X \cup V_D$, where V_X is a set of ordinary nodes that appear in an XML tree, and V_D is a set of distribution nodes (e.g. independent, mutually exclusive). Consider a node u , which has a set of child nodes, V_u , in an XML tree T_X . In PXML, T_P , the ordinary node, u , may have several distribution nodes, as its child nodes, which specify the probability distributions over the disjoint subsets of the children of u , V_u . And E_P is a set of edges

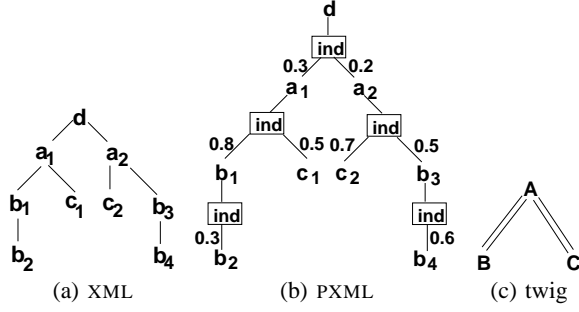


Figure 1: XML, PXML, and twig query

$E_P = E_{XX} \cup E_{XD} \cup E_{DD} \cup E_{DX}$ where E_{XX} is a set of edges that appear in E_X , E_{XD} is a set of edges from V_X nodes to V_D nodes, E_{DD} is a set of edges from V_D nodes to V_D nodes, and E_{DX} is a set of edges from V_D nodes to V_X nodes. Below, we call an E_{XX} edge an ordinary edge, and an edge in $E_{XD} \cup E_{DD} \cup E_{DX}$ a distribution edge. A positive probability is only associated with an edge, $e \in E_{DD} \cup E_{DX}$, denoted as $\rho_e(e)$. Note that a node in V_X has a node-weight, and an edge in $E_{XX} \cup E_{DX}$ is associated with an edge weight.¹

Example 1: An XML tree, T'_X , is shown in Fig. 1(a). There is a D -typed node d , two A -typed nodes (a_1 and a_2), four B -typed nodes (b_i for $1 \leq i \leq 4$), and two C -typed nodes (c_1 and c_2). A PXML tree, T'_P , based on the XML tree T'_X , is shown in Fig. 1(b). In T'_P , d has an independent distribution node as its child, which specifies that its two child nodes, a_1 and a_2 are independent. The probabilities of having a_1 and a_2 are 0.3 and 0.2, as indicated in the incoming edges to a_1 and a_2 , respectively. In a similar fashion, there are other four independent distribution nodes. A node-weight, say $w_v(d)$, in T'_X can be specified as the node-weight associated with $w_v(d)$ in T'_P , and an edge-weight, say $w_e(d, a_1)$ can be specified in the incoming edge to a_1 in T'_P . \square

A PXML tree, T_P , is a compact representation of probability distribution over a collection of XML trees, T_{X_1}, T_{X_2}, \dots , which is generated in two steps.

First, we traverse the PXML tree, T_P , in a top-down fashion. When we visit an independent distribution node, ind_i , which has l children, we divide T_P into 2^l subtrees where each of them has a subset of the l children. When we visit a mutually exclusive distribution node, mux_i , which has l children, we divide T_P into l subtrees where each of them has one child. We repeat the same procedure for each of the divided subtrees recursively, and obtain the set of PXML subtrees, where every connected PXML subtree shares the same root node of the PXML tree. Let T'_P be one PXML subtree. The probability of T'_P , denoted as $\Pr(T'_P)$, is computed in Eq. (1).

$$\Pr(T'_P) = \prod_{u \in V'_P} \Pr(u) \quad (1)$$

Here, if u is an ordinary node, $\Pr(u) = 1$. If u is a distribution node, $\Pr(u)$ is computed as follows. Let u be a mutually exclusive node, and suppose u has l children. There are only two cases, selecting one of l children or none because it is mutually exclusive. For the former, $\Pr(u)$ is the probability associated with its outgoing edge to the selected child node. For the latter, it is one minus the summation of all the l existence probabilities. Let u be an in-

¹For simplicity, we assumed that default weights are zero.

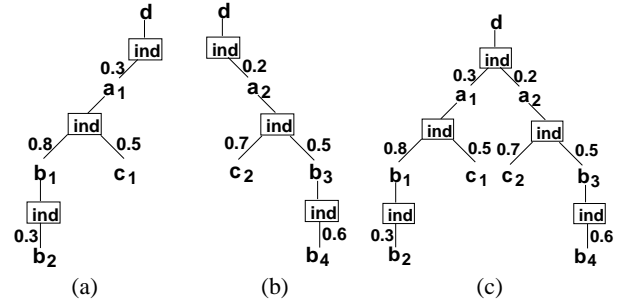


Figure 2: PXML subtrees

dependent node. Suppose u has n children, u_1, u_2, \dots, u_n , out of l children in total in the PXML tree T_P , $u_1, u_2, \dots, u_n, u_{n+1}, \dots, u_l$. $\Pr(u)$ is computed by multiplying the existence probability associated with u_i , for $1 \leq u_i \leq u_n$, and the absence probability (one minus the existence probability) for u_i for $u_n < u_i \leq u_l$.

Following the first step, for the PXML tree (Fig. 1(b)) 4 intermediate PXML subtrees will be generated when visiting the first independent distribution node under the root node. Three are shown in Fig. 2 in addition to the PXML subtree which contains the root node d only. Then, 6 PXML subtrees will be generated from the PXML subtrees Fig. 2(a)(b), respectively, and 36 PXML subtrees will be generated from the PXML subtree Fig. 2(c). In total, 49 PXML subtrees will be generated.

Second, for each of the PXML subtrees, T'_P , where $\Pr(T'_P) > 0$, we construct an XML tree, denoted as $\text{tree}(T'_P)$, by removing all distribution nodes/edges and connecting two ordinary nodes if there are distribution nodes/edges in between. The entire set of such XML trees for a PXML tree is then uniquely identified. We denote it as $\text{pwd}(T_P) = \{T_{X_1}, T_{X_2}, \dots\}$. The probability of T_{X_i} is given by

$$\Pr(T_{X_i}) = \sum_{\text{tree}(T_{P_j})=T_{X_i}} \Pr(T_{P_j}) \quad (2)$$

because the same XML tree T_{X_i} can be constructed from several PXML subtrees, T_{P_j} . The set $\text{pwd}(T_P)$ forms the possible worlds of the probabilistic XML (PXML), T_P , and it satisfies the condition that $\sum_{T_{X_i} \in \text{pwd}(T_P)} \Pr(T_{X_i}) = 1$.

A twig query is a fragment of XPATH queries that can be represented as a query tree, $Q(V, E)$. Here, $V = (V_1, V_2, \dots, V_n)$ is a set of nodes representing types (tag-names), and E is a set of edges. An edge between two typed nodes, for example, A and D , is either associated with an XPATH axis operator $//$ or $/$ to represent $A//D$ or A/D . Given an XML tree T_X , the former is to retrieve all A and D typed nodes that satisfy the ancestor/descendant relationships, and the latter is to retrieve all A and D typed nodes that satisfy parent/child relationships. We call the former $//$ -edge and the latter $/$ -edge in short. As a special case, the root node in the query tree has an incoming $//$ - or $/$ -edge to represent an XPATH query, $//A$ or $/A$, suppose the root node is A -typed. The answer of a n -node twig query, $Q(V, E)$, against an XML tree T_X , is a set of connected subtrees, where a connected subtree consists of n nodes (v_1, v_2, \dots, v_n) in T_X , for $v_i \in V_i$ ($1 \leq i \leq n$), that satisfy all the structural relationships imposed by Q , and the minimal additional nodes/edges connecting the n nodes as a connected subtree. An example of an XPATH query is $Q = //A[./C]//B$ (Fig. 1(c)). In this paper, we consider three classes of twig queries: (1) node query, (2) path query, and (3) tree query. For example, $//A$, $//A//B$,

Algorithm 1 PXML-RANK (T_P, Q, k)

Input: a PXML tree T_P , a twig query Q , and an integer k .
Output: XML trees, $\varphi_1, \dots, \varphi_M$, with top- k probabilities s.t.
 $\rho(\varphi_1) \geq \dots \geq \rho(\varphi_M)$.

- 1: $M \leftarrow \text{twigQuery}(Q, T_P)$;
 - 2: sort $M = \{\varphi_1, \dots, \varphi_N\}$ in the non-increasing order of their scores;
 - 3: $\mathcal{M} \leftarrow \text{P-RANK}(T_P, k, M)$;
 - 4: remove all φ_i from \mathcal{M} if $\rho(\varphi_i) = 0$;
 - 5: sort \mathcal{M} in the non-increasing order of their top- k probabilities ($\rho(\varphi_i)$);
 - 6: **return** \mathcal{M} ;
-

and $\|A\| \|C\| \|B$ are examples of node query, path query, and tree query, respectively.

A twig query, Q , against a PXML tree, T_P , can be processed by ignoring the existence of the distribution nodes/edges in T_P . The result is a set of XML trees, $M(Q, T_P) = \{\varphi_1, \varphi_2, \dots, \varphi_N\}$. Let φ_i be an XML tree in the result for a twig query. The score of φ_i , denoted as $\omega(\varphi_i)$, can be computed using any score function as studied in [7, 19, 2]. For simplicity we define it as the total sum of its node/edge weights in this paper, i.e. $\omega(\varphi_i) = \sum_{u \in \varphi_i} w_v(u) + \sum_{e \in \varphi_i} w_e(e)$. The top- k probability of φ_i , $\rho(\varphi_i)$, is given below.

$$\rho(\varphi_i) = \sum_{\substack{T_{X_j} \in \text{pwd}(T_P) \\ \varphi_i \in \text{topk}(T_{X_j})}} \Pr(T_{X_j}) \quad (3)$$

Here, T_{X_j} is one XML tree in the possible worlds of the PXML tree T_P ($\text{pwd}(T_P)$), and the probability of T_{X_j} , $\Pr(T_{X_j})$, is computed using Eq. (2). The probability of φ_i in the possible world, T_{X_j} , is $\Pr(T_{X_j})$ if φ_i is contained in T_{X_j} and the score of φ_i , $\omega(\varphi_i)$, is at least the k -th largest value in T_{X_j} ($\varphi_i \in \text{topk}(T_{X_j})$). It is important to note that several answers may appear in one possible world simultaneously. The $\rho(\varphi_i)$ is defined as the sum of such probability for every possible world where φ_i is contained.

Problem Statement [Top- k PXML Ranking (PXML-RANK)]: Let T_P be a PXML tree with possible worlds $\text{pwd}(T_P)$. A PXML-RANK query, (Q, k) , is specified by a twig query, Q , and a positive number k , against T_P . It ranks the top- k probabilities for the answers, φ_i , that satisfy the twig query Q .

The algorithm for processing a PXML-RANK query, (Q, k) , is outlined in Algorithm 1. First, it obtains a set of XML trees, $M = \{\varphi_1, \varphi_2, \dots, \varphi_N\}$, that satisfy Q , against T_P (line 1). It can be done over an XML tree which virtually treats every distribution path between two ordinary nodes in T_P as an edge between the two ordinary nodes. Any efficient existing algorithms that process twig query can be adapted [17]. Second, it sorts M in the non-increasing order using the scores, such as φ_i appears before φ_j on the sorted M if $\omega(\varphi_i) \geq \omega(\varphi_j)$ (line 2). Third, it calls P-RANK to compute the top- k probabilities for all answers in M (line 3). P-RANK returns \mathcal{M} , which is a set of pairs $(\varphi_i, \rho(\varphi_i))$ for every answer φ_i in M . Finally, it removes all answers φ_i from \mathcal{M} if their top- k probabilities are zero ($\rho(\varphi_i) = 0$) (line 4), and sorts \mathcal{M} in the non-increasing order of their top- k probabilities ($\rho(\varphi_i)$) (line 5). Such \mathcal{M} is returned in line 6. It is worth noting that P-RANK is a time-consuming task in computing PXML-RANK queries. Given a set of answers, $\{\varphi_1, \varphi_2, \dots, \varphi_N\}$, a naive approach needs to compute $\rho(\varphi_i)$ by enumerating all the possible worlds, $\text{pwd}(T_P)$, using Eq. (3).

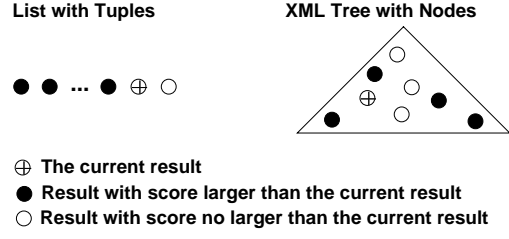


Figure 3: List v.s. XML Tree

Below, we will first discuss how to process node queries (e.g. $\|A$), and then based on our techniques to process node queries we will discuss how to process any path queries (e.g. $\|A\| \|B$), and certain tree queries (e.g. $\|A\| \|C\| \|B$).

3. NODE QUERY

In this section, we discuss processing PXML-RANK queries, (Q, k) , where Q is a node query in the form of $\|A$. A node query is to find all A -typed nodes in PXML T_P to be ranked. Let the answer set M be $M = \{\varphi_1, \varphi_2, \dots, \varphi_N\}$ which is processed by $\text{twigQuery}(Q, T_P)$ in line 1 of Algorithm 1. Note that, here, an answer φ_i is an ordinary node in PXML tree T_P .

In the following, we first introduce some existing algorithms for processing ranking queries in a similar but different setting and discuss their deficiencies for processing PXML-RANK queries, followed by discussions of our new approaches.

3.1 New Containment Issues

In [10], Hua et al. discussed how to answer ranking queries in x-Relation uncertain data. In the x-Relation model, there is a set of independent x-tuples where an x-tuple consists of a set of mutually exclusive tuples (called alternatives). Each tuple in an x-tuple is associated with a score and a probability. A possible world is generated by choosing at most one tuple from each x-tuple. Under the x-Relation model, to process a ranking query, algorithms based on dynamic programming are proposed. The main issue is how to compute the probability that a tuple, t_i , to be the j -th largest ranked tuple in possible worlds, denoted as $p_{i,j}$. Assume all tuples are sorted in the decreasing order based on their scores, $\{t_1, t_2, \dots, t_N\}$. The existing algorithms compute $p_{i,j}$, for $1 \leq i \leq N$ and $1 \leq j \leq k$, where k is the top- k value. First, consider every x-tuple has exactly one alternative, or equivalently, all the tuples are independent. The probability that t_i ranks j -th in a randomly generated possible world from the sorted tuple set $\{t_1, \dots, t_i\}$ is $p_{i,j} = \Pr(t_i) \cdot r_{i-1,j-1}$. Here, $\Pr(t_i)$ is the existence probability of tuple t_i . $r_{i,j}$ is the probability that a randomly generated possible world from the tuple set $\{t_1, \dots, t_i\}$ has exactly j tuples, and can be computed by the following dynamic programming equations.

$$r_{i,j} = \begin{cases} \Pr(t_i) \cdot r_{i-1,j-1} + (1 - \Pr(t_i)) \cdot r_{i-1,j} & \text{if } i \geq j \geq 0; \\ 1 & \text{if } i = j = 0; \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

With the above equations, all $r_{i,j}$ can be computed for $1 \leq i \leq N$ and $j = 0, 1, \dots, k-1$, based on the previous values, namely, $r_{i-1,j-1}$ and $r_{i-1,j}$. When x-tuples represent multiple alternatives, the same dynamic programming equations can be applied with additional tuple transformations [10].

Like the x-Relation model, in our PXML model, we consider independent and/or mutually exclusive nodes, as well as the scores

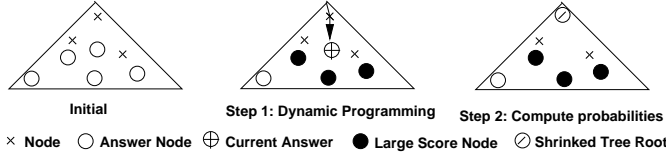


Figure 4: An Overview of Our Approach

and top- k probabilities. Unlike the x-Relation model, we consider one additional criterion, containment. In other words, for a node query, an answer φ_i can be an ancestor/descendant of another answer φ_j in the PXML tree T_P . The additional criterion makes it difficult to apply the existing techniques [24, 10, 25] to solve the problem in our setting, even for node queries. Fig. 3 illustrates the main differences. First, in an x-Relation model, the tuples, $\{t_1, t_2, \dots, t_N\}$, themselves are the context in which the top- k probabilities are computed. The current $r_{i,j}$ for the sorted set of tuples $\{t_1, t_2, \dots, t_{i-1}, t_i\}$ can be computed by the previously computed $r_{i-1,j-1}$ and $r_{i-1,j}$ for the sorted set of tuples $\{t_1, t_2, \dots, t_{i-1}\}$. Note that all tuples are sorted based on their scores in a decreasing order. Every time for computing $r_{i,j}$ the algorithm only needs to consider an additional tuple t_i . The tuples $\{t_{i+1}, \dots, t_N\}$, which have smaller scores than the current tuple t_i , are not needed in the x-Relation model, because they do not affect $r_{i,j}$ computing. However, in our problem setting, it becomes invalid that the nodes which have smaller scores than the current node are not relevant. As shown on the right side of Fig. 3, a node (“o”) with a smaller score than the current node (“⊕”) under consideration can be an ancestor or descendant of the current node. The existence/absence of every node may have impacts on the current node.

Remark 1: The top- k probabilities for answers, $\{\varphi_1, \varphi_2, \dots, \varphi_N\}$, need to be determined in the context of the entire PXML tree. \square

3.2 An Overview of Our Approach

We outline our basic ideas for processing node queries in Fig. 4. We will discuss how to extend the basic ideas to process path queries and certain tree queries, and our optimization techniques later. Let the set of answers, M , be $M = \{\varphi_1, \varphi_2, \dots, \varphi_N\}$ which is processed by $\text{twigQuery}(Q, T_P)$ in Algorithm 1. All such answers in M are identified in the PXML tree T_P . It is shown in Fig. 4, in the initial stage, where tree nodes (not answers) and answers in T_P are indicated as “x” and “o”, respectively. Then, we compute $p_{i,j}$ for every answer $\varphi_i \in M$, for $1 \leq j \leq k$. The answer φ_i to be computed at an iteration is called the current. For the current φ_i , we compute $p_{i,j}$ in two steps, computing a $r_{i,j}$ -like variable (step 1) and computing $p_{i,j}$ (step 2).

In step 1, given the current answer φ_i (indicated as “⊕” in Fig. 4), the $r_{i,j}$ -like variable we compute is $r_{\varphi_i,j}^{\omega(\varphi_i)}$. There exist main differences between $r_{i,j}$ and $r_{\varphi_i,j}^{\omega(\varphi_i)}$. Recall that $r_{i,j}$ is the probability that a randomly generated possible world from the sorted tuple set $\{t_1, \dots, t_i\}$ has exactly j tuples. In Eq. (4), $r_{i,j}$ is computed for the current tuple t_i using the answers that have a larger score than t_i ’s by utilizing the sorted tuple set, $\{t_1, \dots, t_{i-1}\}$, in the decreasing order of the scores. The value of i in $r_{i,j}$ means the position of the i -th tuple itself on the sorted tuple set. In our problem setting, there does not exist such a sorted set. In order to simulate the sorted set, in other words, the set of answers that have a larger score than the score of the current φ_i ($\omega(\varphi_i)$), the superscript of $r_{\varphi_i,j}^{\omega(\varphi_i)}$

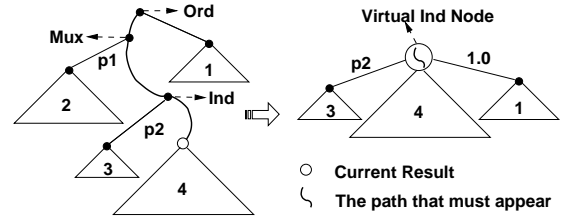


Figure 5: Computing $p_{i,j}$

is introduced. Also, because the sorted tuple set does not exist, the indicator of i used in $r_{i,j}$ for x-Relation model needs to be reconsidered. In our model, instead of i , we use a subscript φ_i to indicate a subtree in PXML tree T_P rooted at node φ_i . The meaning of j in $r_{\varphi_i,j}^{\omega(\varphi_i)}$ remains unchanged. In summary, $r_{\varphi_i,j}^{\omega(\varphi_i)}$ is the probability that a randomly generated possible world from the subtree of the PXML T_P rooted at φ_i has exactly j answers whose score is larger than the score of φ_i , $\omega(\varphi_i)$.

In addition, there is a new issue on containment. For computing $r_{\varphi_i,j}^{\omega(\varphi_i)}$, we need compute all nodes $v \in T_P$ including the answers in M as well as the current node φ_i itself. For this purpose, we introduce a general variable $r_{v,j}^h$ where v is a node in the PXML tree T_P and h is a score. We compute $r_{v,j}^{\omega(\varphi_i)}$ for every node $v \in T_P$, based on the score $\omega(\varphi_i)$, using dynamic programming. It is important to note that $r_{v,j}^{\omega(\varphi_i)}$ can be computed based on the subtrees of the subtree rooted at v in T_P . Upon completion of the computation, $r_{v,j}^{\omega(\varphi_i)}$ are known for every node (including the current) and for $0 \leq j \leq k-1$. An answer $\varphi_l \in M$ is marked as “●” in Fig. 4, if it has a larger score than the current’s ($\omega(\varphi_l) > \omega(\varphi_i)$).

In fact, up to this stage, $r_{\varphi_i,j}^{\omega(\varphi_i)}$ computed is local, since it is computed based on the subtree rooted at φ_i and is not computed in the entire PXML tree T_P globally. Note that there is a path from the root of PXML tree T_P to the current φ_i as indicated by “ $\rightarrow \oplus$ ” in Fig. 4. The $r_{\varphi_i,j}^{\omega(\varphi_i)}$ needs to be computed globally under the condition that the path “ $\rightarrow \oplus$ ” must exist. The condition of the existence of such a path “ $\rightarrow \oplus$ ” may affect some other $r_{v,j}^h$ which in turn affect $r_{\varphi_i,j}^{\omega(\varphi_i)}$ for the current φ_i .

In step 2, based on the condition that the path “ $\rightarrow \oplus$ ” must exist, we compute global $r_{\varphi_i,j}^{\omega(\varphi_i)}$ and $p_{i,j}$ for the current φ_i for $1 \leq j \leq k$. This is done by condensing the path “ $\rightarrow \oplus$ ” into a node indicated as “○” in Fig. 4. In other words, the PXML tree T_P is virtually transformed into another PXML tree T_P' where the path “ $\rightarrow \oplus$ ” in T_P becomes a node “○” in T_P' and all nodes that are connected to the nodes along the path “ $\rightarrow \oplus$ ” in T_P are connected to the node “○” in T_P' . It is worth noting that the global $r_{\varphi_i,j}^{\omega(\varphi_i)}$ and therefore $p_{i,j}$ can be computed using the same dynamic programming because the subtree rooted at φ_i is the entire PXML tree. Fig. 5 illustrates the main idea. The left tree is T_P where the path “ $\rightarrow \oplus$ ” consists of an ordinary node (*ord*), a mutually exclusive node (*mux*), an independent node (*ind*), and the current node φ_i (the root of the subtree (marked 4)). The right tree is T_P' . The subtree (marked 2) and its incoming edge are removed, because the *mux* node implies that the subtree (marked 2) cannot exist. The subtree (marked 3) in T_P is directly linked to the root node in T_P' with the same probability. The *ord* node is treated as an independent node with probability one to the subtree (marked 1), which is connected to the root node in T_P' .

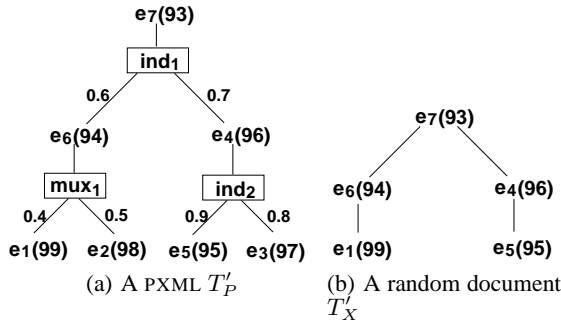


Figure 6: An Example

Finally, given $p_{i,j}$, $\rho(\varphi_i)$ can be computed using the following Eq. (5) instead of Eq. (3).

$$\rho(\varphi_i) = \sum_{j=1}^k p_{i,j} \quad (5)$$

Below, we discuss how to compute $\rho(\varphi_i)$ in a conditional probability viewpoint. Suppose M' is the set of answers where every $\varphi_l \in M'$ has a larger score than φ_i such as $(\omega(\varphi_l) > \omega(\varphi_i))$. The probability of φ_i to appear in the top- k answers, $\rho(\varphi_i)$, can be formulated as follows.

$$\begin{aligned} & \Pr(\varphi_i \text{ appears in the top-}k \text{ answers}) \\ = & \Pr(\varphi_i \text{ appears and at most } k-1 \text{ answers in } M') \\ = & \Pr(\varphi_i \text{ appears}) \times \Pr(\text{at most } k-1 \text{ answers in } M' \mid \varphi_i \text{ appears}) \end{aligned}$$

In [5] Cohen et al studied probabilistic XML with constraints (constraint satisfaction, query evaluation, and sampling), and the computation of $\Pr(\text{at most } k-1 \text{ answers in } M' \mid \varphi_i \text{ appears})$ can be transformed to a constraint satisfaction problem. The constraint satisfaction problem can be specified by modifying the PXML T_P as follows: along the path from the root to φ_i , for each edge (u, v) , (i) if u is a distribution node, then change the probability $\rho_e(u, v)$ to one, (ii) if u is a mutually exclusive node, then remove other children and the corresponding subtrees. Let the modified PXML be T'_P . Then, $\Pr(\text{at most } k-1 \text{ answers in } M' \mid \varphi_i \text{ appears})$ is equal to the probability that a random generated XML tree from T'_P satisfies the constraints that it contains at most $k-1$ answers in M' . Cohen et al. show that the constraint satisfaction problem is polynomial time solvable, and propose an algorithm to solve it. In this work, we compute $\rho(\varphi_i)$, for $1 \leq i \leq N$ for the following main reasons. Although the constraint satisfaction problem is polynomial time solvable, it is proposed for general constraints, and is still time-consuming. For a different φ_i , there is a different T'_P , and the algorithm [5] needs to compute $\rho(\varphi_i)$ individually. Instead we mainly consider how to share the costs of computing different $\rho(\varphi_i)$'s using specific constraints as discussed above.

3.3 An Example

In this section, from a different viewpoint (conditional probability viewpoint), we explain how to compute $\rho(\varphi_i)$ using an example. Suppose φ_i is the current answer and M' is the set of answers where every $\varphi_l \in M'$ has a larger score than φ_i such as $(\omega(\varphi_l) > \omega(\varphi_i))$. The probability of φ_i to appear in the top- k answers can be formulated as follows.

$$\begin{aligned} & \Pr(\varphi_i \text{ appears in the top-}k \text{ answers}) \\ = & \Pr(\varphi_i \text{ appears and at most } k-1 \text{ answers in } M') \\ = & \sum_{j=0}^{k-1} \Pr(\varphi_i \text{ appears and exact } j \text{ answers in } M') \\ = & \sum_{j=0}^{k-1} \Pr(\varphi_i \text{ appears}) \times \Pr(\text{exact } j \text{ answers in } M' \mid \varphi_i \text{ appears}). \end{aligned}$$

Here, $\Pr(\varphi_i \text{ appears})$ can be easily computed by multiplying all probabilities, $\rho_e(\cdot)$, along the path from root node of the PXML tree T_P to φ_i (“ $\rightarrow \oplus$ ”). The conditional probability of $\Pr(\text{exact } j \text{ answers in } M' \mid \varphi_i \text{ appears})$ is computed upon the condensed new PXML tree T'_P .

Fig. 6(a) shows a PXML T'_P that specifies the relationships among E-products (E -typed). There are 7 E-products e_i for $1 \leq i \leq 7$. An E-product has a score (indicated in the brackets) as its performance. There are some uncertainties. The distribution node ind_1 implies that e_7 is a part of e_6 with probability 0.6 and is a part of e_4 with probability 0.7. The two are independent. The distribution node mux_1 implies that either e_6 is used in e_1 with probability 0.4 or is used in e_2 with probability 0.5, but cannot be used in both. The two are mutually exclusive to each other. Suppose a PXML-RANK query (Q, k) is issued against T'_P (Fig. 6(a)), where $Q = \parallel E$ and $k = 1$. The set of E-products to be ranked is $M = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ which is computed using $\text{twigQuery}(Q, T'_P)$ in Algorithm 1. Note that M is sorted in the decreasing order of the scores (performance indicators). Next, all E-products in M will be ranked based on top- k probabilities, $\rho(e_i)$, for $1 \leq i \leq 7$, against the possible words $\text{pwd}(T'_P)$.

One of the possible worlds (e.g. XML tree T'_X) is shown in Fig. 6(b). T'_X is with the conditions that e_6 and e_4 coexist under the independent node ind_1 , e_5 is present alone under the independent node ind_2 , and e_1 is present under the mutually exclusive node mux_1 . The probability of T'_X is $\Pr(T'_X) = (0.4 \times 0.6) \times ((0.9 \times (1 - 0.8)) \times 0.7) = 0.03024$ where 0.4 is the probability of the subtree rooted at e_6 , and $(0.9 \times (1 - 0.8))$ is the probability of the subtree rooted at e_4 in T'_X , respectively. It is infeasible to compute $\rho(e_i)$ using Eq. (3) because it needs to enumerate all possible worlds $\text{pwd}(T'_P)$ and summarize the top- k probabilities for e_i to be ranked top. Note that the size of possible worlds can be exponential. Instead we efficiently compute $\rho(e_i)$ using Eq. (5).

There are several obvious cases. (1) e_7 has the smallest score (93) in M . The only possible world for e_7 to be ranked top is the XML tree with e_7 only. $\rho(e_7) = (1 - 0.6) \times (1 - 0.7) = 0.12$. (2) e_1 has the largest score in M . If it appears in a possible world, it will be ranked top $\rho(e_1) = 0.6 \times 0.4 = 0.24$. (3) e_2 is ranked top if and only if e_1 does not appear in the possible worlds where e_2 appears. Note that e_1 and e_2 are mutually exclusive. In other words, if e_2 appears, then e_1 will not appear. $\rho(e_2) = 0.6 \times 0.5 = 0.3$. (4) e_5 can not be ranked top, because its ancestor e_4 has a higher score than e_5 , and whenever e_5 appears e_4 will always appear. $\rho(e_5) = 0$.

Next consider a PXML-RANK query (Q, k) against T'_P (Fig. 6(a)) where Q is the same $\parallel E$ but $k = 3$. The set of answers to be ranked is the same $M = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$. We discuss computation of top- k probability, $\rho(e_6)$, for e_6 to be ranked top- k . Let e_6 to be the current with $\omega(e_6) = 94$. The E-products that have larger scores than $\omega(e_6) = 94$ are $M_h = \{e_1, e_2, e_3, e_4, e_5\}$ which we call h-answers.

Consider e_6 . The PXML tree T'_P can be divided into several parts, P , T_1 , and T_2 , as shown in Fig. 7. Here, P represents “ $\rightarrow \oplus$ ” which must appear because e_6 must appear. Obviously, $\Pr(e_6 \text{ appears}) = \Pr(P \text{ appears}) = 0.6$. We have $\rho(e_6) = \Pr(P \text{ appears and at most } 2 \text{ h-answers appear}) = \sum_{j=0}^2 \Pr(P \text{ appears and exact } j \text{ h-answers appear}) = \sum_{j=0}^2 \Pr(P \text{ appears}) \times \Pr(\text{exact } j \text{ h-answers appear} \mid P \text{ appears})$. Note that $\Pr(P \text{ appears}) = 0.6$. We explain how to

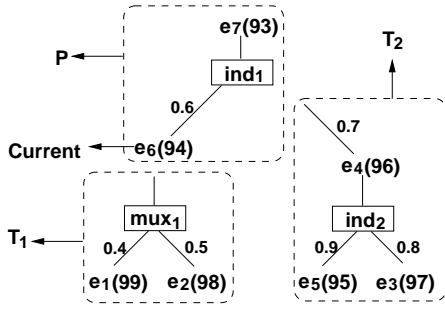


Figure 7: Cut the tree into several parts

compute

$$\Pr(\text{exact } j \text{ h-answers appear} \mid P \text{ appears}) \quad (6)$$

for $0 \leq j \leq 2$ below. Recall that, given the current e_6 , in our notation, the probability that exact j answers from M_h appear in the subtree rooted at v is denoted as $r_{v,j}^{\omega(e_6)} = r_{v,j}^{94}$. For simplicity, we use $r_{v,j}$ below.

Case-1 ($j = 0$): Eq. (6) equals to $\Pr(0 \text{ h-answer in } T_1 \text{ appears}) \times \Pr(0 \text{ h-answer in } T_2 \text{ appears})$. For the first part, it is $r_{mux_1,0}$. The only situation that 0 h-answer appears in the subtree rooted at mux_1 (T_1) is that none of e_1 and e_2 appears. Since e_1 and e_2 are mutually exclusive, we have $r_{mux_1,0} = 1.0 - \rho_e(mux_1, e_1) - \rho_e(mux_1, e_2) = 1.0 - 0.4 - 0.5 = 0.1$. The second part is equal to $\Pr(\text{the absence of edge } (ind_1, e_4)) + \Pr(\text{the existence of edge } (ind_1, e_4)) \times r_{e_4,0} = (1.0 - 0.7) + 0.7 \times r_{e_4,0}$. Here, $r_{e_4,0} = 0$ because e_4 must appear when considering the subtree rooted at e_4 , so the second part is $0.3 + 0.7 \times 0 = 0.3$. Combining the two parts, Eq. (6), for $j = 0$, equals to $0.1 \times 0.3 = 0.03$.

Case-2 ($j = 1$): Eq. (6) equals to $\Pr(0 \text{ h-answer in } T_1 \text{ appears}) \times \Pr(1 \text{ h-answers in } T_2 \text{ appear}) + \Pr(1 \text{ h-answers in } T_1 \text{ appear}) \times \Pr(0 \text{ h-answer in } T_2 \text{ appears})$. Note that $\Pr(0 \text{ h-answer in } T_1 \text{ appears}) = 0.1$ and $\Pr(0 \text{ h-answer in } T_2 \text{ appears}) = 0.3$ are computed in Case-1.

Here, $\Pr(1 \text{ h-answer in } T_1 \text{ appears}) = r_{mux_1,1}$. The only situation that 1 h-answer in the subtree rooted at mux_1 (T_1) appears is that either e_1 appears or e_2 appears. Since e_1 and e_2 are mutually exclusive, we have $\Pr(1 \text{ h-answer in } T_2 \text{ appears}) = r_{mux_1,1} = \rho_e(mux_1, e_1) + \rho_e(mux_1, e_2) = 0.4 + 0.5 = 0.9$.

Also $\Pr(1 \text{ h-answer in } T_2 \text{ appears}) = \Pr(\text{the existence of edge } (ind_1, e_4)) \times r_{e_4,1} = 0.7 \times r_{e_4,1}$. The only situation that 1 h-answer appears in the subtree rooted at e_4 (T_2) is that 0 h-answer appears in the subtree rooted at ind_2 , i.e., $r_{e_4,1} = r_{ind_2,0}$. It means that neither e_5 nor e_3 appears, i.e. $r_{ind_2,0} = (1 - \rho_e(ind_2, e_5)) \times (1 - \rho_e(ind_2, e_3)) = (1 - 0.9) \times (1 - 0.8) = 0.02$. Then, we have $\Pr(1 \text{ h-answer in } T_2 \text{ appears}) = 0.7 \times 0.02 = 0.014$.

Therefore, Eq. (6), for $j = 1$, equals to $0.1 \times 0.014 + 0.9 \times 0.3 = 0.2714$.

Case-3 ($j = 2$): Eq. (6) equals to $\Pr(0 \text{ h-answer in } T_1 \text{ appears}) \times \Pr(2 \text{ h-answers in } T_2 \text{ appear}) + \Pr(1 \text{ h-answer in } T_1 \text{ appears}) \times \Pr(1 \text{ h-answer in } T_2 \text{ appears}) + \Pr(2 \text{ h-answers in } T_1 \text{ appear}) \times \Pr(0 \text{ h-answer in } T_2 \text{ appears}) = 0.1 \times \Pr(2 \text{ h-answers in } T_2 \text{ appear}) + 0.9 \times 0.014 + \Pr(2 \text{ h-answers in } T_1 \text{ appear}) \times 0.3$.

Algorithm 2 P-RANK (T_P, k, M)

Input: a PXML T_P , an integer k , and a sorted set of twig query answers $M = \{\varphi_1, \dots, \varphi_N\}$, s.t. $\omega(\varphi_1) \geq \dots \geq \omega(\varphi_N)$.
Output: $(\varphi_i, \rho(\varphi_i))$, for $1 \leq i \leq N$.

```

1: for  $i \leftarrow 1$  to  $N$  do
2:   H-PROB ( $T_P, \varphi_i, k$ );
3:    $P \leftarrow \text{path}(T_P, \varphi_i)$ ;
4:    $\mathcal{T}_P \leftarrow \text{PATH-CONDENSE}(T_P, \varphi_i)$ ;
5:    $v' \leftarrow \text{root}(\mathcal{T}_P)$ ;
6:   H-TOPK ( $v', k, \omega(\varphi_i), \text{children}(v')$ );
7:    $s \leftarrow \text{count}(P)$ ;
8:    $p \leftarrow \prod_{e \in P} \rho_e(e)$ ;
9:    $p_{i,j} \leftarrow 0$ , for  $1 \leq j \leq s$ ;
10:   $p_{i,j} \leftarrow p \cdot r_{v',j-s-1}^{\omega(\varphi_i)}$ , for  $s+1 \leq j \leq k$ ;
11:   $\rho(\varphi_i) \leftarrow \sum_{j=1}^k p_{i,j}$ ;
12: return  $\{(\varphi_1, \rho(\varphi_1)), \dots, (\varphi_N, \rho(\varphi_N))\}$ ;

```

The probabilities for $j < 2$ are computed already in Case-1 and Case-2.

Here, $\Pr(2 \text{ h-answers in } T_1 \text{ appear}) = r_{mux_1,2}$. To have 2 h-answers appear in the subtree rooted at mux_1 , both e_1 and e_2 must appear. This is impossible because e_1 and e_2 are mutually exclusive. We have $\Pr(2 \text{ h-answers in } T_1 \text{ appear}) = r_{mux_1,2} = 0$.

On the other hand, $\Pr(2 \text{ h-answers in } T_2 \text{ appear}) = \Pr(\text{existence of edge } (ind_1, e_4)) \times r_{e_4,2} = 0.7 \times r_{e_4,2}$. The only situation that 2 h-answers appear in the subtree rooted at e_4 is that 1 h-answer appears in the subtree rooted at ind_2 , i.e., $r_{e_4,2} = r_{ind_2,1}$. It means that either (a) e_5 appears but e_3 does not appear or (b) e_5 does not appear but e_3 appears. We have $r_{ind_2,1} = \rho_e(ind_2, e_3) \times (1 - \rho_e(ind_2, e_5)) + (1 - \rho_e(ind_2, e_3)) \times \rho_e(ind_2, e_5) = 0.9 \times (1 - 0.8) + (1 - 0.9) \times 0.8 = 0.26$. We have $\Pr(2 \text{ h-answers in } T_2 \text{ appear}) = 0.7 \times 0.26 = 0.182$. Therefore, Eq. (6), for $j = 2$, equals to $0.1 \times 0.182 + 0.9 \times 0.014 + 0 \times 0.3 = 0.0308$.

With all the three cases, $\rho(e_6) = \Pr(P \text{ appears}) \times \sum_{j=0}^2 \Pr(\text{exact } j \text{ h-answers appear} \mid P \text{ appears}) = 0.6 \times (0.03 + 0.2714 + 0.0308) = 0.19932$. For the PXML-RANK query ($\parallel E, 3$) against T'_P (Fig. 6(a)), the ranking is shown below.

$\rho(e_4)$	$\rho(e_3)$	$\rho(e_7)$	$\rho(e_5)$	$\rho(e_2)$	$\rho(e_1)$	$\rho(e_6)$
0.7	0.56	0.37924	0.35784	0.3	0.24	0.19932

3.4 Algorithms

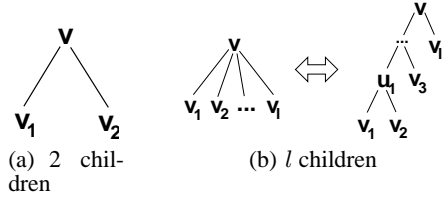
The algorithm to compute P-RANK for node queries is given in Algorithm 2. It takes three inputs. The PXML tree T_P , the top- k value k , and a set of answers $M = \{\varphi_1, \varphi_2, \dots, \varphi_N\}$ which is sorted in the decreasing order of their scores $\omega(\cdot)$. For each φ_i (the current), in a for-loop, it processes the following tasks. It computes its local $r_{\varphi_i,j}^{\omega(\varphi_i)}$ using dynamic programming (line 2). It identifies the path from the root of T_P to the current φ_i (" $\rightarrow \oplus$ "), and assigns it to P (line 3). Then, it virtually reconstructs T_P to \mathcal{T}_P by condensing the path P into a node which is the root of \mathcal{T}_P , v' (line 4-5). It computes the global $r_{\varphi_i,j}^{\omega(\varphi_i)}$ in line 6 using dynamic programming where $\text{children}(v')$ indicates the children of the root node v' . In order to compute $p_{i,j}$ where i implies φ_i , it counts how many nodes on the path P (" $\rightarrow \oplus$ ") that are with a score greater than $\omega(\varphi_i)$ (line 7). The algorithm computes $p_{i,j}$ (line 8-10), and then computes $\rho(\varphi_i)$ in line 11.

Algorithm 3 H-PROB (T_P, φ, k)**Input:** a PXML $T_P(V_P, E_P)$, an answer φ , and an integer k .**Output:** $r_{v,j}^{\omega(\varphi)}$, for $v \in V_P$ and $0 \leq j \leq k-1$.

```

1:  $h \leftarrow \omega(\varphi)$ ;
2: for every  $v \in V_P$  in the post-order traversing order do
3:   if  $v$  is a leaf node then
4:      $r_{v,j}^h \leftarrow 0$ , for  $0 \leq j \leq k-1$ ;
5:   if  $v$  is an answer with  $\omega(v) > h$  then
6:      $r_{v,1}^h \leftarrow 1$ ;
7:   else
8:      $r_{v,0}^h \leftarrow 1$ ;
9:   else
10:    let  $\{v_1, \dots, v_l\}$  be the set of children of  $v$  in  $T_P$ ;
11:    H-TOPK ( $v, k, h, \{v_1, \dots, v_l\}$ );
12:  return  $r_{v,j}^{\omega(\varphi)}$ ;

```

**Figure 8:** Computing $r_{v,j}^h$

We explain H-PROB (used in line 2, Algorithm 2). The H-PROB algorithm is given in Algorithm 3. It takes three inputs, the PXML tree T_P , the current answer φ , and the value of k . The main task of H-PROB is to compute local $r_{v,j}^h$ where $h = \omega(\varphi)$ in a bottom-up fashion. For non-leaf nodes, it further calls H-TOPK (Algorithm 4) to compute using dynamic programming. The H-TOPK algorithm takes four inputs to compute $r_{v,j}^h$, for $0 \leq j < k$. Here, h is the score of the current φ , v is the non-leaf node in question, $\{v_1, \dots, v_l\}$ are the children of v . It assumes that $r_{v_i,j}^h$, for $1 \leq i \leq l$, have already been computed. (Note that the H-PROB algorithm uses a bottom-up traversal to compute.) There are several cases handled in the H-TOPK algorithm: (1) v is a mutually exclusive distribution node (line 1-3), (2) v is an independent distribution node (line 4-11), and (3) v is an ordinary node (line 12-20).

Below, we explain the case when v has 2 children ($l = 2$): $\{v_1, v_2\}$ (Fig. 8(a)). Note that both $r_{v_1,j}^h$ and $r_{v_2,j}^h$ have been computed.

If v is a mutually exclusive node, for $j > 0$, v_1 and v_2 can not appear simultaneously. $r_{v,j}^h$ can be computed in two cases, either with the subtree rooted at v_1 or with the subtree rooted at v_2 . That is, $r_{v,j}^h = \rho_e(v, v_1) \cdot r_{v_1,j}^h + \rho_e(v, v_2) \cdot r_{v_2,j}^h$. For $j = 0$, it needs to consider an additional case that none of the two subtrees are selected. $r_{v,0}^h$ is computed as follows. $r_{v,0}^h = \rho_e(v, v_1) \cdot r_{v_1,0}^h + \rho_e(v, v_2) \cdot r_{v_2,0}^h + (1 - \rho_e(v, v_1) - \rho_e(v, v_2))$.

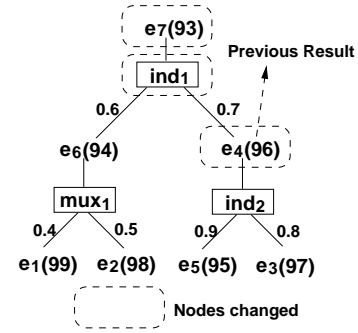
If v is an independent node, the existence of v_i is independent from each other where $i = 1, 2$. v chooses either v_1 , or v_2 , or both, or none. Let $B_{v_i,j}^h$ be the probability that a randomly generated XML tree, from the branch (v, v_i) which consists of the edge (v, v_i) and the subtree rooted at v_i , contains exactly j answers with score greater than h . When $j > 0$, $B_{v_i,j}^h = \rho_e(v, v_i) \cdot r_{v_i,j}^h$, and, when $j = 0$, $B_{v_i,0}^h = \rho_e(v, v_i) \cdot r_{v_i,0}^h + (1 - \rho_e(v, v_i))$. Then, $r_{v,j}^h$ includes the cases that, in a randomly generated XML tree, one branch, say (v, v_1) , contains exactly s ($\leq j$) answers with a score greater than h , and the other branch, say (v, v_2) , contains

Algorithm 4 H-TOPK ($v, k, h, \{v_1, \dots, v_l\}$)**Input:** a node $v \in T_P$, an integer k , a weight h , and the children set of v , $\{v_1, \dots, v_l\}$, with $r_{v_i,j}^h$ values computed.**Output:** $r_{v,0}^h, \dots, r_{v,k-1}^h$.

```

1: if  $v$  is mutually exclusive then
2:    $r_{v,j}^h \leftarrow \sum_{i=1}^l \rho_e(v, v_i) \cdot r_{v_i,j}^h$ , for  $0 \leq j \leq k-1$ ;
3:    $r_{v,0}^h \leftarrow r_{v,0}^h + (1 - \sum_{i=1}^l \rho_e(v, v_i))$ ;
4: if  $v$  is independent then
5:   for  $i \leftarrow 1$  to  $l$  do
6:      $B_{v_i,j}^h \leftarrow \rho_e(v, v_i) \cdot r_{v_i,j}^h$ , for  $0 \leq j \leq k-1$ ;
7:      $B_{v_i,0}^h \leftarrow B_{v_i,0}^h + (1 - \rho_e(v, v_i))$ ;
8:    $r_{u_0,j}^h \leftarrow B_{v_1,j}^h$ , for  $0 \leq j \leq k-1$ ;
9:   for  $i \leftarrow 1$  to  $l-1$  do
10:     $r_{u_i,j}^h \leftarrow \sum_{s=0}^j r_{u_{i-1},s}^h \cdot B_{v_{i+1},j-s}^h$ , for  $0 \leq j \leq k-1$ ;
11:    $r_{v,j}^h \leftarrow r_{u_{l-1},j}^h$ , for  $0 \leq j \leq k-1$ ;
12: if  $v$  is ordinary then
13:    $r_{u_0,j}^h \leftarrow r_{v_1,j}^h$ , for  $0 \leq j \leq k-1$ ;
14:   for  $i \leftarrow 1$  to  $l-1$  do
15:     $r_{u_i,j}^h \leftarrow \sum_{s=0}^j r_{u_{i-1},s}^h \cdot r_{v_{i+1},j-s}^h$ , for  $0 \leq j \leq k-1$ ;
16:   if  $v$  is an answer with  $\omega(v) > h$  then
17:      $r_{v,j}^h \leftarrow r_{u_{l-1},j-1}^h$ , for  $1 \leq j \leq k-1$ ;
18:      $r_{v,0}^h \leftarrow 0$ ;
19:   else
20:      $r_{v,j}^h \leftarrow r_{u_{l-1},j}^h$ , for  $0 \leq j \leq k-1$ ;
21:  return  $r_{v,0}^h, \dots, r_{v,k-1}^h$ ;

```

**Figure 9:** Computing H-PROB (T_P, φ_{i+1}, k)

exactly $j - s$ answers with a score greater than h , and is computed as $r_{v,j}^h = \sum_{s=0}^j B_{v_1,s}^h \cdot B_{v_2,j-s}^h$, for $0 \leq j \leq k-1$.

If v is an ordinary node, it can be computed by treating v as an independent node, i.e. v has two independent children v_1 and v_2 , with probability $\rho_e(v, v_1) = 1$ and $\rho_e(v, v_2) = 1$, which means that both edges must exist with probability one. First compute $\tilde{r}_{v,j}^h$ by treating v as an independent node, i.e. $\tilde{r}_{v,j}^h = \sum_{i=0}^j r_{v_1,i}^h \cdot r_{v_2,j-i}^h$. Note that in this case $B_{v_i,s}^h = r_{v_i,s}^h$. If v itself is an answer with score greater than h , then $r_{v,j}^h = \tilde{r}_{v,j-1}^h$ for $0 < j \leq k-1$ and $r_{v,0}^h = 0$, otherwise $r_{v,j}^h = \tilde{r}_{v,j}^h$ for $0 \leq j \leq k-1$.

We design our algorithm for handling general l children of a given node v . If v is an independent node or an ordinary node, in order to compute $r_{v,j}^h$, we need to consider how many exact answers out of j answers are from which subtrees by enumerating all the sequences, i_1, \dots, i_l , such that $\sum_{s=1}^l i_s = j$ for any fixed $j \in \{0, \dots, k-1\}$. We handle l children of a given node v , if it is an independent/ordinary node, by transforming the node v with l children into a left-deep binary subtree, as shown in Fig. 8(b).

node (v)	$r_{v,j}^h$ for $h = \omega(e_4)$			$r_{v,j}^h$ for $h = \omega(e_5)$		
	$j = 0$	$j = 1$	$j = 2$	$j = 0$	$j = 1$	$j = 2$
e_1	0	1.0	0	0	1.0	0
e_2	0	1.0	0	0	1.0	0
e_3	0	1.0	0	0	1.0	0
e_4	0.2	0.8	0	0	0.2	0.8
e_5	1.0	0	0	1.0	0	0
e_6	0.1	0.9	0	0.1	0.9	0
e_7	0.2024	0.4952	0.3024	0.138	0.2264	0.3332
mu_{x_1}	0.1	0.9	0	0.1	0.9	0
ind_1	0.2024	0.4952	0.3024	0.138	0.2264	0.3332
ind_2	0.2	0.8	0	0.2	0.8	0

Table 1: Consecutive computing $r_{v,j}^h$ for $h = \omega(e_4), \omega(e_5)$

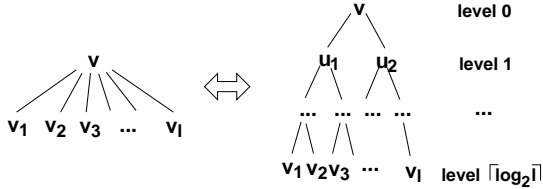


Figure 10: Convert a node with l children to a complete binary tree

There are $2l - 1$ nodes in the transformed binary tree in total, $\{v_1, \dots, v_l, u_1, \dots, u_{l-2}, v\}$. Here, v_1, \dots, v_l are leaf nodes, u_1 has two children (v_1 and v_2), u_i , for $i > 1$, has two children (u_{i-1} and v_{i+1}), for $2 \leq i \leq l-2$. The node v has two children u_{l-2} and v_l . If v is an ordinary node, then u_1, \dots, u_{l-2} are ordinary nodes with weight 0. If v is an independent node, then u_1, \dots, u_{l-2} also are independent nodes, the probability $\rho_e(v, v_i)$ is specified on the incoming edge to v_i . All other edges have probability one. It can be verified that the transformed left-deep tree will give the same result. The H-TOPK algorithm (Algorithm 4) is designed using the left-deep binary tree, where v_1 is treated as u_0 and v is treated as u_{l-1} .

Optimization-I: As indicated in the P-RANK algorithm, it needs to call the H-PROB algorithm for every answer φ_i in $M = \{\varphi_1, \varphi_2, \dots, \varphi_N\}$, which is sorted in the decreasing order of the scores. The cost of computing using dynamic programming is costly. In fact, the cost can be shared between successive calls of H-PROB, e.g., H-PROB (T_P, φ_i, k) and H-PROB (T_P, φ_{i+1}, k). Consider the same PXML-RANK query ($\|E, 3$) against the PXML tree T'_P (Fig. 6(a)). Table 1 shows the results of $r_{v,j}^h$ when computing the two consecutive answers, e_4 and e_5 . It shows that most values when computing $r_{v,j}^h$ for e_5 remain unchanged, given $r_{v,j}^h$ computed for e_4 . The possible change part is highlighted in the dot rectangles in Fig. 9, which is along the path from the root to the previous answer. A lemma is given below.

Lemma 1: Let H-PROB (T_P, φ_i, k) and H-PROB (T_P, φ_{i+1}, k) be two consecutive executions, for two answers, φ_i and φ_{i+1} in the sorted answer set M . When $\omega(\varphi_i) \neq \omega(\varphi_{i+1})$, the values $r_{v,j}^{\omega(\varphi_i)}$ and $r_{v,j}^{\omega(\varphi_{i+1})}$ are identical for the nodes that are not on the path from the root of T_P to the node φ_i . When $\omega(\varphi_i) = \omega(\varphi_{i+1})$, for all nodes, $r_{v,j}^{\omega(\varphi_i)} = r_{v,j}^{\omega(\varphi_{i+1})}$. \square

Due to space limit, we omit the proof. The P-RANK and H-PROB algorithms only need to be slightly changed to adapt the optimization-I.

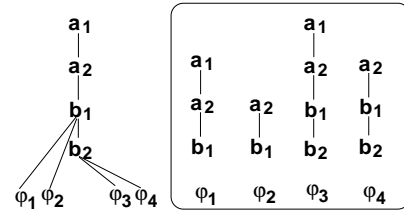


Figure 11: Handling Multiple Path Results in a Single Node

Optimization-II: As stated in Optimization-I, for two successive H-PROB (T_P, φ_i, k) and H-PROB (T_P, φ_{i+1}, k), only the $r_{v,j}^{\omega(\varphi_i)}$ values for those nodes v on a certain path change. For each such a node v , suppose it has l children v_1, v_2, \dots, v_l , if v is an independent node or an ordinary node, we have to spend $O(k^2 \cdot l)$ time to compute $r_{v,j}^h$ using H-TOPK. When l is large, the cost can be large. In the following, we show that we can reduce it to $O(k^2 \cdot \log(l))$. We construct a complete binary tree \mathcal{B} with $d = \lceil \log_2 l \rceil + 1$ levels, where the root is at level 0 and the p -th level has 2^p nodes for $0 \leq p < d$. The root of \mathcal{B} is v , the non-leaf nodes are independent nodes marked u_i , and the leaf nodes are v_1, v_2, \dots, v_l under nodes in level $d - 1$ as shown in Fig. 10. The probability associated with the edge between v and v_i is reserved for the edges incoming v_i in \mathcal{B} . Other newly added edges have probability 1. It can be proved that $r_{v,j}^{\omega(\varphi_i)}$ on the new tree is equal to those on the original tree. Utilizing the complete binary tree, we only need $O(k^2 \cdot \log(l))$ to compute v without affecting other nodes because the depth of the new tree is $O(\log(l))$ and in the path from v to v_i , the degree of each node is at most 2.

4. PATH QUERY AND TREE QUERY

In this section, we discuss other twig queries, namely, path queries, $\|A\|B$, and tree queries $\|A\|C\|B$. Our techniques can efficiently process any path queries. Consider a PXML-RANK query ($\|A\|B, k$), against the PXML tree T_P (Fig. 1(b)). The answers are a set of paths, $M = \{a_1 \rightarrow b_1, a_1 \rightarrow b_1 \rightarrow b_2, a_2 \rightarrow b_3, a_2 \rightarrow b_3 \rightarrow b_4\}$. In our problem setting, the existence probability of a node is equal to the probability of the path from the root to the node, because the existence of a node depends on the existence of its ancestors. Therefore, we can compute the top- k probability for an answer of a path query, as to compute the last node of the answer. Taken $a_1 \rightarrow b_1 \rightarrow b_2$ as an example, we can compute its top- k probability as to compute the top- k probability for b_2 . Our techniques can be used for processing any PXML-RANK queries (Q, k), where Q is a path query.

We further explain how to process when several answers that have the same lowest node, using an example. Suppose that there are four answers to be ranked, $M = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$, where an answer φ_i is shown in the rectangle in Fig. 11 over the data path in PXML tree on the left side in Fig. 11. Here, the two answers, φ_1 and φ_2 , share the same lowest node b_1 , and the other two answers, φ_3 and φ_4 , share the same lowest node b_2 . When processing top- k probabilities for the four (path) answers, we virtually add four additional nodes to represent the four (path) answers as indicated along the path on the left side in Fig. 11. Here, b_1 has two additional virtual children indicated φ_1 and φ_2 , and b_2 has two additional virtual children φ_3 and φ_4 . With the additional virtual nodes, we can process top- k probabilities using the same techniques we discussed for processing top- k probabilities for node queries.

4.1 Discussions on tree query

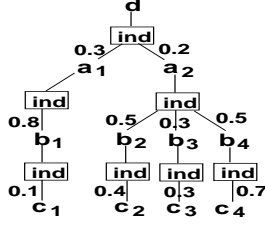


Figure 12: A PXML tree

However, it is difficult to efficiently compute any PXML-RANK queries (Q, k) , where Q is an arbitrary tree query, even for $k = 1$. We explain it using an example. Consider the PXML tree in Fig. 12, and a PXML-RANK query, (Q, k) where $Q = //A[//B]//C$, and $k = 1$. There are 10 answers. For simplicity, we use a 3-tuple to indicate a resulting subtree for the PXML-RANK query. One resulting subtree is $r_1 = (a_1, b_1, c_1)$. There are other 9 resulting subtrees rooted at a_2 with any one of the three b_i , for $2 \leq i \leq 4$, and any one of the three c_j , for $2 \leq j \leq 4$. As one example, consider $r_2 = (a_2, b_2, c_3)$ where a_2 has three children $\{b_2, b_3, b_4\}$, b_2 is in the subtree rooted at b_2 and c_3 is in the subtree rooted at b_3 . The fact states that our dynamic programming techniques cannot be used to efficiently compute top- k probabilities even for r_1 over the 10 subtrees. It is because that we cannot compute $r_{a_2, j}^h$ based on the values of $r_{b_i, j}^h$, for $2 \leq i \leq 4$, that are obtained for the children. Any combinations are possible, and we need to enumerate all possible worlds.

We give conditions. A general PXML-RANK tree query, (Q, k) , can be computed in polynomial time if one of the conditions are satisfied. The conditions are imposed on the set of answers, $M = \{\varphi_1, \varphi_2, \dots, \varphi_N\}$, to be ranked, which is generated by `twigQuery(Q, TP)` in Algorithm 1. The conditions can be checked when processing twig queries without high overhead. Below, for a given tree query Q , we call a path in Q a primary path and denote it as P_Q . For example, $//A//B$ is a primary path of $//A[//C]//B$. Note that an answer φ_i can be a subtree.

1. The edges of results do not overlap with each other. $\varphi_i \cap \varphi_j = \emptyset$ for $i \leq j$.
2. Every edge, e' , of every answer $\varphi_i \in M$, which is not on the primary path P_Q , must be associated with $\rho_e(e') = 1$. (An ordinary edge, e' , is considered as an edge with $\rho_e(e') = 1$).
3. Let φ_i and φ_j be two different answers. Suppose β_i and β_j are the paths in φ_i and φ_j that match P_Q , respectively. There exist two subtrees $\gamma_i = \varphi_i - \beta_i$ and $\gamma_j = \varphi_j - \beta_j$ where $\varphi_i = \beta_i \cup \gamma_i$ and $\varphi_j = \beta_j \cup \gamma_j$. If there exist a node v_i on the path β_i that is a descendant of a node v_j on the path β_j , then γ_i and γ_j must be identical.

As an example, consider the PXML tree, T'_P in Fig. 1(b), and the twig query $Q = //A[//C]//B$ (Fig. 1(c)). The answer set $M = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$ where $\varphi_1 = \{(a_1, b_1), (a_1, c_1)\}$, $\varphi_2 = \{(a_1, b_1), (b_1, b_2), (a_1, c_1)\}$, $\varphi_3 = \{(a_2, b_3), (a_2, c_2)\}$, and $\varphi_4 = \{(a_2, b_3), (b_3, b_4), (a_2, c_2)\}$. Let $P_Q = //A//B$. The four answers, φ_i , for $1 \leq i \leq 4$, do not satisfy the first and the second condition. But they satisfy the third condition. For example, consider φ_1 and φ_2 , $\beta_1 = (a_1, b_1)$ and $\beta_2 = (a_1, b_1)(b_1, b_2)$; and $\gamma_1 = (a_1, c_1)$ and $\gamma_2 = (a_1, c_1)$. There exist b_1 on β_1 that is a descendant of a node a_1 on β_2 . $\gamma_1 = \gamma_2$.

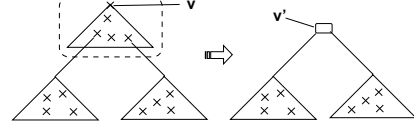


Figure 13: Compute top- k probabilities for a subtree answer

The conditions allow us to compute a PXML-RANK tree query using our dynamic programming techniques. In brief, when either the second or the third condition is satisfied, we can compute top- k probabilities for a subtree φ_i as to compute top- k probabilities for the lowest node of φ_i on the primary path P_Q in the same way of processing path queries. Processing a tree query, when the first condition is satisfied, is complicated, because some nodes of a subtree answer φ_i can be ancestors of another subtree answer φ_j . We discuss our main ideas below, due to the space limit.

Let $M = \{\varphi_1, \varphi_2, \dots\}$ be a set of the subtree answers to be ranked, and let φ be the current subtree. We call a subtree answer φ_i a h-answer if it has a larger score than the current's. For each φ_i , we use $root(\varphi_i)$ to denote the root of φ_i , P_{φ_i} to denote the path from the root node in the PXML tree to $root(\varphi_i)$, and $Pr_{\varphi_i} = \prod_{e \in \varphi_i} \rho_e(e)$ if there are no mutually exclusive nodes in φ_i , otherwise $Pr_{\varphi_i} = 0$. We use $r_{v, j}$ instead of $r_{v, j}^{\omega(\varphi)}$ for short below. First, for computing the probability $Pr(j$ h-answers appear $| \varphi$ appears), we set all $\rho_e() = 1$ for the edges in $P_{\varphi} \cup \varphi$, and remove the branches which are mutually exclusive with any nodes in $P_{\varphi} \cup \varphi$. We cannot simply condense P_{φ} to a node, because P_{φ} may contain some part of other h-answers. The process of setting $\rho_e() = 1$ serves the same purpose of condensing a path into a node. Second, we compute $r_{v, j}$ where a subtree answer φ_i is rooted on v . Note that v is an ordinary node. We condense φ_i to a virtual node v' and remove the branches that are mutually exclusive with any nodes in φ_i (see Fig. 13). The true $r_{v, j}$ is computed as follows: $r_{v, j} = r_{v, j} - Pr_{\varphi_i} \times r_{v', j} + Pr_{\varphi_i} \times r_{v', j-1}$, where the $r_{v, j}$ appears on the right side is computed on the left tree in Fig. 13 where every node/edge in φ_i is considered separately. The existence of the entire φ_i is ensured on the right tree in Fig. 13. It can be easily handled when multiple subtree answers are rooted at the same node v .

5. EXPERIMENTAL STUDIES

We conduct extensive experiments to test the performance of our algorithms. We have implemented our PXML-RANK algorithm. The main algorithm to be tested is P-RANK. We have implemented P-RANK without Optimization-I and Optimization-II, the P-RANK algorithm using Optimization-I, and the P-RANK algorithm using both Optimization-I and Optimization-II. We denote them as pRank, pRank-I and pRank-II, respectively. All algorithms were implemented in C++. We conducted all the experiments on a 2.8GHz CUP and 2GB memory PC running XP.

We use two real datasets, DBLP (<http://dblp.uni-trier.de/xml/>) and Mondial (<http://www.informatik.uni-freiburg.de/~may/lopix/lopix-mondial.html>), and the synthetic XML benchmark dataset XMark (<http://monetdb.cwi.nl/xml/>) for testing. For XMark, we also generate many datasets with different sizes. For each XML dataset used, we generate the corresponding PXML tree, using the same method as used in [14]. We visit the nodes in the original XML tree from top to bottom. For each node v visited, we randomly choose some distribution nodes with

ID	Query	Result
D1	dblp//book[./author]/key	1, 684
D2	dblp//article[./title/sub]/key	2, 928
D3	dblp//proceedings[key]/series[href]	5, 909
D4	dblp//incollection[key]/author	8, 842
D5	dblp//article[./cite[label]]/key	13, 785
M1	mondial//river[./located[country]/province][id]/name	237
M2	mondial//city[country][./population/year][latitude]/province	705
M3	//country[./province[name][population/city[id]][capital]/total_area	2, 595
M4	mondial//organization[established][headq]/members[type]	5, 226
M5	mondial//organization[./members[type]/country][name]/abbrev	7, 505
X1	site//category[./text/bold]/id	712
X2	site//description/keyword/emph	1824
X3	//namerica/item[./parlist/listitem/listitem]/id	3, 043
X4	//closed_auctions/closed_auction/itemref[item]	5, 850
X5	//open_auctions/open_auction[id]/author[person]	7, 200

Table 2: Queries used for all datasets

random types and probability distributions to be the children of v , then for the original children of v , we choose some of them to be the children of the new generated distribution nodes. We control the percentage of the distribution nodes to generate different PXML trees for each dataset.

In answering a PXML-RANK query, we first compute all answers using a modified twig pattern matching algorithm based on [17]. The algorithm we use can process the entire PXML tree in a streaming manner, and therefore does not need to keep the entire PXML tree in memory. Then we prune nodes, v , on the PXML tree if the subtree rooted at v does not effect the ranking, and get another projected PXML tree. We run our three algorithms pRank, pRank-I and pRank-II, to compute top- k probabilities for all results over the projected PXML tree if they satisfy one of our conditions. For each test, we record the time and space consumption of all algorithms. The time consumption consists the query processing time to generate all the results, the projection time and the time for computing top- k probabilities for all results. The main space consumption is caused by maintaining $r_{v,j}^h$ values. For pRank, $r_{v,j}^{\omega(\varphi_i)}$ values for a node v_i can be released when $r_{v,j}^{\omega(\varphi_i)}$ has been computed where v is the parent of v_i . For pRank-I and pRank-II, in order to share the computational cost, $r_{v,j}^{\omega(\varphi_i)}$ values need to be kept for computing $r_{v,j}^{\omega(\varphi_{i+1})}$. pRank-II consumes more memory because Optimization-II needs to maintain complete binary trees.

For the DBLP dataset, the original XML tree has 13, 318, 516 nodes with 41 different tags and the maximum depth of 6. We range the percentage of distribution node from 10% to 50% and generate 5 different PXML trees. The queries used for the DBLP dataset are listed in Table 2 from D1 to D5, with combination of both // and / operators. We list them in increasing order of result size. The parameters used for testing DBLP dataset are listed in Table 3, where DistNode means the percentage of distribution node. For the Mondial dataset, the original XML tree has 70, 459 nodes with 51 different tags and the maximum depth of 5. The queries used and parameters with default values are listed in Table 2 from M1 to M5 and Table 3 respectively. For the XMark datasets, we generate 5 different datasets with different sizes for testing. There are 77 different tags with the maximum depth of 12 for each of the generated XML trees. We set the percentage of distribution node to be 30% and convert them to the corresponding PXML tree. The number of nodes for each PXML tree is shown in the last row of Table 3. The queries used and parameters with default values are listed in Table 2 from X1 to X5 and Table 3 respectively.

Parameter	Range	Default
DistNode(All)	10%, 20%, 30%, 40%, 50%	30%
Top- k (All)	10, 20, 30, 40, 50	30
Query(DBLP)	D1, D2, D3, D4, D5	D3
Query(Mondial)	M1, M2, M3, M4, M5	M3
Query(XMark)	X1, X2, X3, X4, X5	X3
Node Number(XMark)	0.5, 1, 1.5, 2, 2.5 ($\times 10^6$)	1.5

Table 3: Parameters used for testing

5.1 Test-DBLP

Fig. 14 shows the testing results over DBLP datasets. From Fig. 14(a) and 14(b), we know that when the percentage of distribution nodes increases, both the time and memory consumption for the three algorithms marginally increase. pRank-II is more than 300 times faster than pRank although cost about three times more memory than pRank. pRank-I is more than 10 times faster than pRank although cost about 2 times more memory than pRank. Fig. 14(c) and Fig. 14(d) show that when the number of results increases, the time and memory used for the three algorithms do not necessarily increase. It is because the increasing of the number of results does not mean that the size of the projected PXML tree to be tested also increases. The time for all the three algorithm is influenced by both the number of results and the size of the projected PXML tree. The memory consumption for all the three algorithms reflects the size of the projected tree. In Fig. 14(e) and Fig. 14(f), we can see that, when k increases, the time for all the three algorithms will increase. The memory consumption for pRank-I and pRank-II will increase linearly with k , while the memory consumption for pRank is not influenced by k . pRank-II is also much more faster (about 100 times faster) than pRank although cost some more memory (not larger than 8 times more). The performance of pRank-I is between the other two algorithms.

5.2 Test-Mondial

Fig. 15 shows the performance of the three algorithms over the Mondial dataset. Fig. 15(a) and Fig. 15(b) show that when the percentage of distribution nodes increases, the time and memory consumption for all the three algorithms will increase. pRank-II is more than 100 times faster than pRank, and is 2 times faster than pRank-I. The memory consumption of pRank-II and pRank-I are almost the same and are 3 times more than pRank. In Fig. 15(c) and Fig. 15(d), when the number of results increases in the Mondial dataset, the size of the projected tree will also increase (which is reflected by the increasing of memory consumption), so the time for all the three algorithms will increase. Fig. 15(e) and Fig. 15(f) show that when k increases, the time for all the three algorithms will increase. The memory consumption for pRank-I and pRank-II will increase linearly while the the memory consumption for pRank remains the same. Comparing to the DBLP dataset, in the Mondial dataset, the time for pRank-I is more similar to pRank-II, because in the DBLP dataset, we can always find nodes with very large number of children in the projected PXML tree, for example the root node tagged "DBLP", which will decrease the efficiency of pRank-I, whereas, in the Mondial dataset, the degree of each node is not large which makes the advantages of optimization-II less obvious.

5.3 Test-XMark

Fig. 16 shows the performance of the algorithms on the XMark datasets. Fig. 16(a) and Fig. 16(b) show that when the number of distribution nodes increases, the time and memory consumption for all the three algorithms will also increase. Fig. 16(c) and Fig. 16(d) show that when the number of result increases, the time for all the three algorithms will increase if the size of the projected

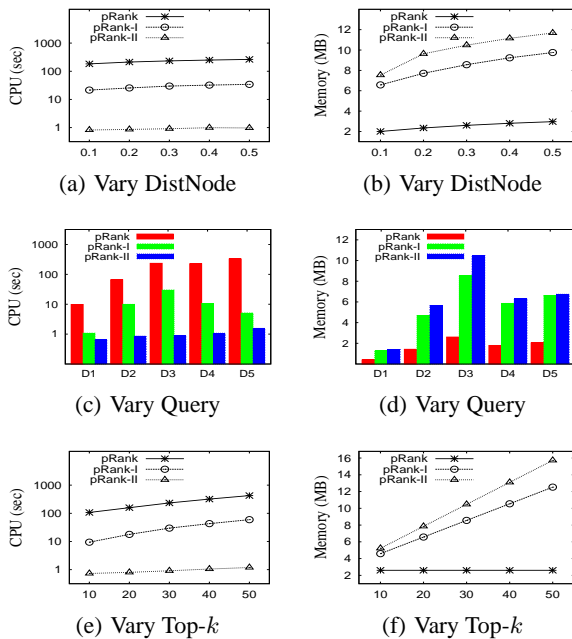


Figure 14: Testing DBLP dataset

PXML tree (memory consumption) increases. Otherwise, the time is influenced by the size of the projected PXML tree. Fig. 16(e) and Fig. 16(f) show the performance of the three algorithms when increasing k , which are similar to that in the DBLP dataset. In Fig. 16(g) and Fig. 16(h), when the size of the data (Node Number) increases, the time and memory consumption for all the three algorithms will increase, because when the size of the data increases, both the number of result and the size of the projected PXML tree will increase. The time used for pRank-II is about 100 times faster than pRank, and the time used for pRank-I is about 30 times faster than pRank for all experiments on the XMark datasets. The memory consumption for pRank-I is similar to the memory cost for pRank-II, and is at most 5 times the memory consumption for pRank for all experiments on the XMark datasets.

6. RELATED WORK

Ranking Results in XML: XML twig queries have been extensively studied [4]. The result of a twig query over an XML tree is a set of answers. In [7, 19, 2], the authors integrate IR strategies to rank the results of twig queries. The scores of answers are computed using IR models, incorporating with other factors. In [6, 8], the authors retrieve the top- k results of a keyword search over XML tree. In [3, 15], the authors treat XML trees as XML graphs, and assign weights to the nodes and edges of XML graphs, where the weight of a node indicates its importance and the weight of an edge represents the strength of its semantic connection in the XML tree. The uncertainty and probability are not addressed.

Probabilistic XML: The topic of probabilistic XML (PXML) has been studied recently. Many models have been proposed, together with the complexity analysis of query evaluations. Nierman et al. [16] first introduce a simple probabilistic XML model, ProTDB, which is a probabilistic tree database. Hung et al. [11, 12] model the probabilistic XML as directed acyclic graphs, with probabilities defined on sets of children. Keulen et al. [22] use a probabilistic tree approach for data integration. Abiteboul et al. [1] propose a

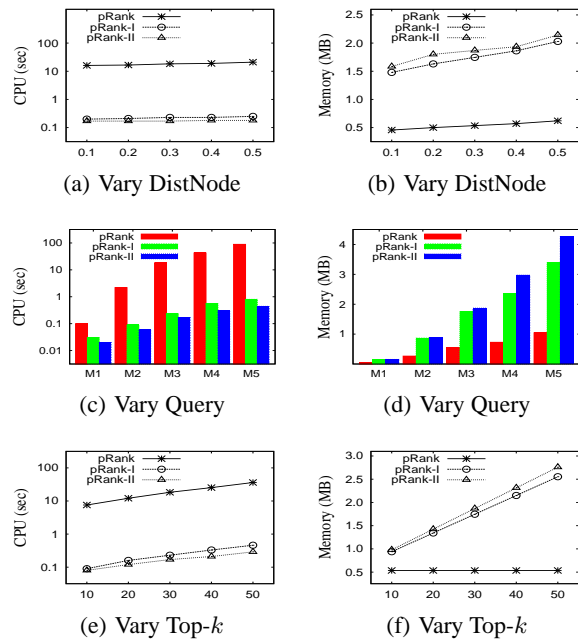


Figure 15: Testing Mondial dataset

“fuzzy trees” model, where nodes are associated with conjunctions of probabilistic event variables, they also give a full complexity analysis of query and update on the “fuzzy tree” in [20]. Cohen et al. [5] incorporate a set of constraints to express more complex dependencies among the probabilistic data. They also propose efficient algorithms to solve the constraint-satisfaction, query evaluation, and sampling problem under a set of constraints. In [14], Kimelfeld et al. summarize and extend the probabilistic XML models previously proposed, the expressiveness and tractability of queries on different models are discussed. The ranking issues are not addressed in their work.

Top-k Queries in Probabilistic Data: Uncertain databases have received increasing attention recently. Apart from the works on different models of uncertain relational database, some recent works concern on answering top- k queries in uncertain database. There are two scenarios in ranking the query results, [18] or [21]. In [18], Re et al. find the k most probable answers for a given general SQL query. In this scenario, each answer has a probability instead of a score, which intuitively represents the confidence of its existence, ranking is only based on probabilities. They use Monte Carlo simulations to get the top- k results efficiently. Another definition is ranking the results by the interplay between score and uncertainty. In the setting of [21, 23, 9, 25, 24, 10], each result is a tuple, associated with both a score and a probability. U-TopK and U-kRanks queries are first proposed in [21], Yi et al. [23, 24] improve the performance of the two queries using a dynamic programming approach. Hua et al. [9, 10] define the PT- k query, and proposed three approaches to answer the PT- k query, which are, dynamic programming method, sampling method, and Poisson approximation based method. Zhang et al. propose a Global-Topk definition in [25]. Jin et al. [13] adapt the U-TopK/U-kRanks/PT-k/Global-Topk (Where Global-Topk is the same as Pk-topk in [13]) queries in a uncertain stream environment with sliding-window, and design both space- and time-efficient synopses to continuously monitor the top- k results. The works do not consider the containment issues and rank in XML trees.

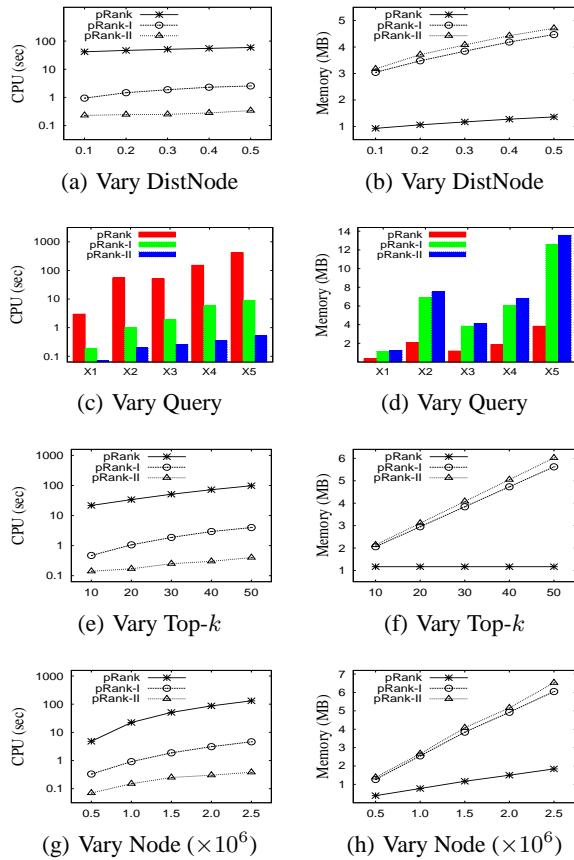


Figure 16: Testing XMark datasets

In our study, we consider all the three issues, namely, ranking, probability, and structures.

7. CONCLUSION

In this paper, we studied ranking twig queries in probabilistic XML (PXML) data, called PXML-RANK query, (Q, k) , where Q is a twig query, which can be node queries, path queries, and tree queries. We proposed dynamic programming algorithms with optimization techniques to efficiently rank answers of node queries, and showed that our techniques can be used to efficiently rank answers of path queries. For tree queries, we gave conditions under which we gave our solutions to rank answers of such tree queries without enumerating all the possible worlds. Our extensive experimental studies confirmed the efficiency of our approaches.

Acknowledgment

This work was supported by a grant of RGC (No. 418206), Hong Kong SAR, China.

8. REFERENCES

- [1] S. Abiteboul and P. Senellart. Querying and updating probabilistic information in XML. In *Proc. of EDBT'06*, pages 1059–1068, 2006.
- [2] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman. Structure and content scoring for XML. In *Proc. of VLDB'05*, pages 361–372, 2005.
- [3] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *Proc. of ICDE'02*, pages 431–440, 2002.

- [4] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal XML pattern matching. In *Proc. of SIGMOD'02*, pages 310–321, 2002.
- [5] S. Cohen, B. Kimelfeld, and Y. Sagiv. Incorporating constraints in probabilistic XML. In *Proc. of PODS'08*, pages 109–118, 2008.
- [6] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSearch: A semantic search engine for XML. In *Proc. of VLDB'03*, pages 45–56, 2003.
- [7] N. Fuhr and K. Großjohann. XIRQL: A query language for information retrieval in XML documents. In *Proc. of SIGIR'01*, pages 172–180, 2001.
- [8] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *Proc. of SIGMOD'03*, pages 16–27, 2003.
- [9] M. Hua, J. Pei, W. Zhang, and X. Lin. Efficiently answering probabilistic threshold top-k queries on uncertain data. In *Proc. of ICDE'08*, pages 1403–1405, 2008.
- [10] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: A probabilistic threshold approach. In *Proc. of SIGMOD'08*, 2008.
- [11] E. Hung, L. Getoor, and V. S. Subrahmanian. Probabilistic interval XML. In *Proc. of ICDT'03*, pages 358–374, 2003.
- [12] E. Hung, L. Getoor, and V. S. Subrahmanian. PXML: A probabilistic semistructured data model and algebra. In *Proc. of ICDE'03*, 2003.
- [13] C. Jin, K. Yi, L. Chen, J. X. Yu, and X. Lin. Sliding-window top-k queries on uncertain streams. In *Proc. of VLDB'08*, 2008.
- [14] B. Kimelfeld, Y. Kosharovskiy, and Y. Sagiv. Query efficiency in probabilistic XML models. In *Proc. of SIGMOD'08*, 2008.
- [15] B. Kimelfeld and Y. Sagiv. Twig patterns: From XML trees to graphs. In *Proc. of WebDB'06*, 2006.
- [16] A. Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *Proc. of VLDB'02*, pages 646–657, 2002.
- [17] L. Qin, J. X. Yu, and B. Ding. *TwigList*: Make twig pattern matching fast. In *Proc. of DASFAA'07*, pages 850–862, 2007.
- [18] C. Re, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. of ICDE'07*, pages 886–895, 2007.
- [19] T. Schlieder and H. Meuss. Querying and ranking XML documents. *Proc. of JASIST'02*, 53(6):489–503, 2002.
- [20] P. Senellart and S. Abiteboul. On the complexity of managing probabilistic XML data. In *PODS*, pages 283–292, 2007.
- [21] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Top-k query processing in uncertain databases. In *Proc. of ICDE'07*, pages 896–905, 2007.
- [22] M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic XML approach to data integration. In *Proc. of ICDE'05*, pages 459–470, 2005.
- [23] K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases. In *Proc. of ICDE'08*, 2008.
- [24] K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases with x-relations. In *Proc. of TKDE'08*, 2008.
- [25] X. Zhang and J. Chomicki. On the semantics and evaluation of top-k queries in probabilistic databases. In *Proc. of DBRank'08*, pages 556–563, 2008.