Information Processing and Management 48 (2012) 1181–1202

Contents lists available at SciVerse ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/infoproman

Efficient probabilistic XML query processing using an extended labeling scheme and a lightweight index

Jung-Hee Yun [a], Chin-Wan Chung [b,*]

[a] Information Resource Division, eGovframework Center, National Information-Society Agency, Seoul, South Korea
[b] Division of Web Science and Technology & Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, South Korea

ARTICLE INFO

Article history:
Received 28 April 2010
Received in revised form 26 December 2011
Accepted 14 January 2012
Available online 8 February 2012

Keywords:
XML
Probabilistic XML
Labeling scheme
Probabilistic XML query

ABSTRACT

Recently there is a growing interest in the data model and query processing for probabilistic XML data. There are many potential applications of probabilistic data, and the XML data model is suitable to represent hierarchical information and data uncertainty of different levels naturally. However, the previously proposed probabilistic XML data models and query processing techniques separate finding data matches with evaluating the probabilities of results. Therefore, they should repeatedly access the data and need to get full data of paths given in queries to calculate the probabilities of results.

In this paper, we propose an extended interval-based labeling scheme for the probabilistic XML data tree and an efficient query processing procedure using the labeling scheme. Against previous researches, our method accesses only the labels of data specified in queries and finds data matches simultaneously with evaluating the probability of each data match. Also, we present an extended probabilistic XML query model with the predicates for the values of probabilities and a lightweight index for those probabilities in order to eliminate unnecessary access to data that will not be included in results.

Experimental results show that our approach is efficient in probabilistic XML query processing and our index scheme significantly improves the performance of query processing when the predicates for the values of probabilities are given.

1. Introduction

Probabilistic data has been widely investigated due to many potential applications such as information extraction, natural language processing and scientific experimental data management. These applications can generate and treat uncertain data, and require the management and query processing of these uncertain data efficiently. Therefore, it is necessary to insert data into a database, along with a proper indication of the level of certainty associated with it. For the query processing, these certainty levels should be returned with the query results. In most cases, the indication of the level of certainty can be a probability, and uncertain data can be represented by probabilistic data.

For the information extraction application, traditional information retrieval can retrieve documents containing specified keywords. Afterwards, it is extended to automatically extract information from retrieved text documents. This extracted information may have some form of certainty or probability associated with it. This information is probabilistic data and can be represented by probabilistic XML data. Especially, if the extracted information is unstructured or hierarchical, then the probabilistic XML data is more suitable for that. For example, when extracting some information about John from a text document which describes the information about John, the result of the retrieval for the age of John can be 35 with 0.3, 37

* Corresponding author. Tel.: +82 42 350 3537; fax: +82 42 350 3510.
E-mail addresses: yunjh@nia.or.kr (J.-H. Yun), chungcw@cs.kaist.ac.kr (C.-W. Chung).

0306-4573/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ipm.2012.01.005

with 0.3 and 39 with 0.4 in probabilities due to the characteristics of the processing method of the engine or the uncertainty of the document itself. Similarly, the name of John's mother can be Julia with 0.5 and Jane with 0.5 in probabilities. This could be represented in probabilistic relational data as in existing researches. However, it is more natural to represent this information using the probabilistic XML data in Fig. 1. The representation of the data can be different according to the probabilistic XML data model. Probabilistic queries about John can be processed on the given probabilistic XML data. The efficiency of query processing will depend on the probabilistic XML data model and probabilistic query processing method. This example data look simple, but the extracted data could be more structured and complex. Additionally, for the scientific data, the uncertainty of data is inevitable because of the reliability of experiments.

To manage probabilistic data, there has been a lot of work on supporting probabilistic data in relational databases (Deay and Sarkar, 1996; Eiter et al., 2000; Fuhr and Rölleke, 1997; Lakshmanan et al., 1997). They considered the modeling of uncertain data in the relational database, and there have been some issues such as the unit with which probabilities will be associated and whether the resulting database is to remain in the first normal form. Several methods proposed to solve these issues generated some drawbacks such as a large amount of redundant storage and the loss of information. One of the reasons for these drawbacks is the flat structure of relational databases, and it seems that XML is more flexible and allows greater heterogeneity of structure and incompleteness of information compared with relational data. These considerations make researches about probabilistic XML data.

The probabilistic XML data is XML data with uncertainty that may be represented by probabilities on elements, attributes or text data in the XML document. Recent studies (Abiteboul and Senellart, 2006; Cohen et al., 2009; Hung et al., 2003a, 2003b; Kimelfeld et al., 2008; Li et al., 2006; Nierman and Jagadish, 2002) reveal that XML data model is proper not only to represent hierarchical information but also to represent data uncertainty of different levels naturally because of the tree-like structure and flexibility. Therefore, there is a growing interest in the data model and query processing for probabilistic XML data.

The researches about probabilistic XML data include data modeling and query processing of probabilistic XML data. The issues about probabilistic XML data are the expressiveness of the model, and the efficiency for the query processing which should generate the query results with the probability.

Several researches (Abiteboul and Senellart, 2006; Cohen et al., 2009; Hung et al., 2003a, 2003b; Kimelfeld et al., 2008; Li et al., 2006; Nierman and Jagadish, 2002) studied the data modeling of probabilistic XML data. These studies proposed several data models for the representation of probabilistic XML data, such as the DTD adaptation method (Nierman and Jagadish, 2002), possible worlds model (Abiteboul and Senellart, 2006), a simple probabilistic tree model (Abiteboul and Senellart, 2006), fuzzy tree model (Abiteboul and Senellart, 2006), and p-documents model (Abiteboul et al., 2009; Kimelfeld et al., 2008, 2009). Each model has a different expressiveness, but most of them assume that a node is given the conditional probability on the existence of its parent node. Most of them proposed the query processing methods, which found data matches and calculated the possibility of the matches.

Query evaluation over probabilistic XML data needs not only to find data matches but also to compute the probability of each match. To find data matches, the previously proposed various methods such as indexing or labeling scheme (Bruno et al., 2002; Chen et al., 2004; Chien et al., 2002; Li and Moon, 2001; Srivastava et al., 2002; Tatarinov et al., 2002; Wu et al., 2004; Zhang et al., 2001) can be applied, but to evaluate the probability of each answer, an additional process is needed. Some existing techniques for finding data matches lose their efficiencies and advantages by the additional processing for computing the probability of each result when they are applied to the query processing for probabilistic XML data.

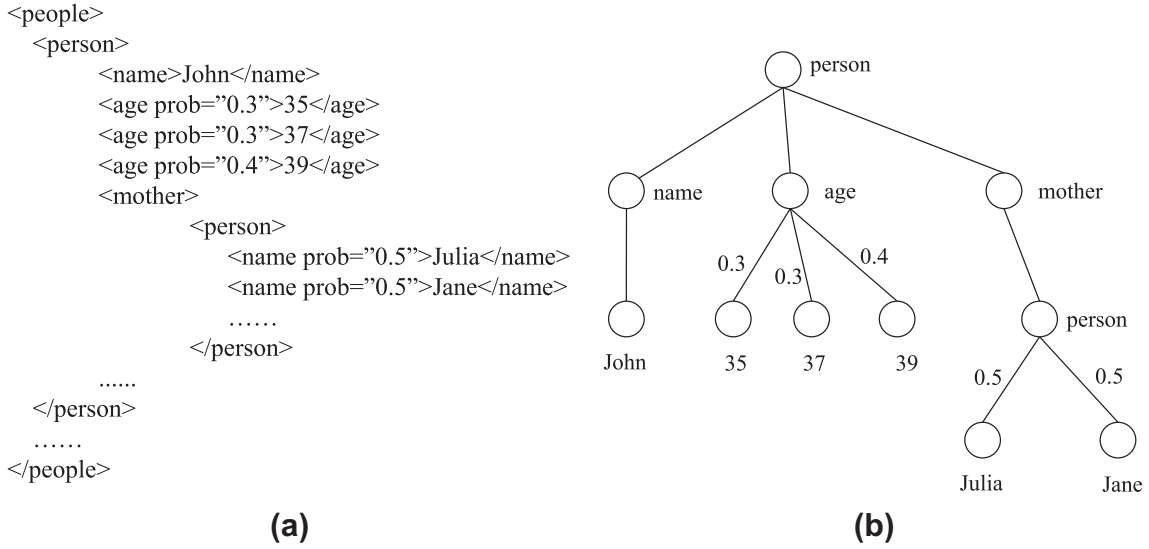

Fig. 1. Examples of probabilistic XML data.

The interval-based labeling scheme has advantages in the size of labels. The sizes of labels in the interval-based labeling scheme are the same for all nodes because the label is composed of two integers for the start position and the end position. However, the sizes of labels in the path-based labeling scheme could be large if nodes are in the deeper level, and the size of delimiters cannot be ignored. In other words, the interval-based labeling schemes have good efficiency in the perspective of the size of labels, whereas in the path-based labeling schemes, the size of labels is large and incurs high storage overhead. In order to keep the probability information, we should extend the labels from previous labeling schemes. Therefore, in the terms of the size of labels, we choose the interval-based labeling scheme. In addition, the interval-based labeling scheme has advantages in processing queries including ancestor–descendant relationships.

In addition to the storage overhead, path-based method has another disadvantage in efficiency in the comparison between labels. For path-based method, the comparison between labels is not simple such as the substring comparison, whereas for the interval-based method, the comparison is the integer comparison. This is another advantage of query processing in the interval-based method.

Previous XML query processing techniques based on interval-based labeling schemes (Chien et al., 2002; Li and Moon, 2001; Srivastava et al., 2002; Zhang et al., 2001) have strong advantages in processing queries including ancestor–descendant relationships. They need not to know the full path information of the ancestor–descendant relationships for query processing. However, when they are adapted to the probabilistic XML query processing, it is necessary to access all the nodes in the path from ancestor to descendant in order to compute the probability of the result. For example, when considering a query `/country//name`, to calculate the probability of a query result using existing techniques, we need to traverse the data tree from the node matching `country`, retrieve the probabilities all the nodes in the path to the node matching `name`. Therefore, the techniques in Srivastava et al. (2002), Li and Moon (2001), Chien et al. (2002), and Zhang et al. (2001), which efficiently handles descendant axis traversal via a labeling scheme without accessing intermediate data nodes, cannot be fully utilized to achieve the efficiency.

Our data model for probabilistic XML data is a probabilistic XML tree which has two types of nodes, ordinary and distributional. The ordinary node indicates elements, attributes or texts and the distributional node presents the distribution over the subsets of its children.

In this paper, we present an extended interval-based labeling scheme for probabilistic XML tree. The proposed labeling scheme extends the previous interval-based labeling scheme in order to manage not only ordinary nodes but also distributional nodes, and it includes two values of conditional and existence probabilities. The conditional probability represents the probability value of a node to be present assuming its parent exists and the existence probability is the probability value of a node to exist in possible XML trees. This labeling scheme makes it possible to combine the evaluation of query result probability with finding data matches of the query. In addition, the evaluation of query result's probability is possible without accessing all ancestors of nodes in the query result. Accordingly we can accomplish the improvement in the performance of probabilistic XML query processing.

Moreover, we propose an extended probabilistic XML query model with the predicates for the values of conditional or existence probabilities and a lightweight index for those probabilities. The index uses the characteristics of the probability value, which is at least zero and at most one. Using the proposed index for probability, we improve the performance of query processing which has the conditions on probabilities.

The contributions of this paper are the following:

- We propose an extended interval-based labeling scheme, which extends the previous interval-based labeling scheme to manage not only ordinary nodes but also distributional nodes, and includes the values of conditional and existence probabilities.
- We present procedures to process probabilistic XML queries efficiently using the extended interval-based labeling scheme. Our method accesses only the labels of data specified in queries and finds data matches simultaneously with evaluating the probability of results.
- We present an extended probabilistic XML query model and a lightweight index for probabilities. This index makes it possible to eliminate unnecessary accesses to data that will not be included in results.

The rest of the paper is organized as follows. Section 2 reviews related works. Section 3 gives basic preliminaries including probabilistic XML tree and probabilistic XML query tree, while Section 4 discusses the extended interval-based labeling scheme. We describe the method of query processing in Section 5. Section 6 reports experimental results and then Section 7 concludes the paper.

2. Related works

2.1. Probabilistic XML data processing

Several probabilistic XML data models have been proposed in Nierman and Jagadish (2002), Hung et al. (2003a, 2003b), Abiteboul and Senellart (2006), Li et al. (2006), Kimelfeld et al. (2008), and Cohen et al. (2009). The DTD adaptation method was presented in Nierman and Jagadish (2002) as the probabilistic XML data representation model. Each element could have the `Prob` attribute to represent the value of probability, and a new element type `Dist` is added in order to express the

distribution among sibling elements which is either independent or mutual-exclusive. This model is similar to that of Kimelfeld and Sagiv (2007). Moreover, Nierman and Jagadish (2002) describes a method to compute the result probabilities for queries in which child axes and branches are supported.

The probabilistic representation for semi-structured data expressed by acyclic directed graphs was proposed in Hung et al. (2003a, 2003b). The probabilities or intervals of probabilities are defined over sets of children to support arbitrary distributions over the relationships between an object and its children. In Hung et al. (2003b), an algebra is proposed to express queries with child axes.

A possible worlds model, a simple probabilistic tree model and a fuzzy tree model were considered in Abiteboul and Senellart (2006). The possible worlds model represents the probabilistic information as a possible worlds set which is a finite set of a data tree and a probability value pairs. In the simple probabilistic tree model, each node in the tree can be attached the probability value of that node to be present assuming its parent is. In the fuzzy tree model, a probabilistic condition which is a conjunction of atomic events is assigned to each node, and each atomic event has probability. However, query processing method and query performance are not mentioned in Abiteboul and Senellart (2006).

A query-friendly probabilistic XML database, presented as PEPX (Probability Encoded Probabilistic XML), was described in Li et al. (2006). It represents a model for probabilistic XML data, which is similar to the model in Nierman and Jagadish (2002) except that the stored probability of each node is the chain probability. The chain probability is the existence probability of a node that is computed by the product of all conditional probabilities from root to the node. In PEPX, query trees are evaluated by two phases, query decomposition and P-joins. In the first phase, a query tree is decomposed to a set of subqueries, such that each subquery does not have branch nodes. Then, the intermediate results of subqueries are stitched using P-joins to compute the final query result.

Recently, Kimelfeld et al. (2008, 2009) and Abiteboul et al. (2009) provide an abstract model for representing probabilistic XML data as p-documents that are trees with two kinds of nodes, ordinary and distributional. They consider several types of distributional nodes: deterministic (det), independent (ind), mutually exclusive (mux), explicit (exp) and conjunction of independent events (cie). The distributional node specifies the probability distribution of choosing a subset of the children of the distributional node. The det means that each child is chosen with probability 1, the ind means that the choices of distinct children are independent, the mux means that at most one child can be chosen, the exp means that the probability of choosing each subset of children is explicitly given unless it is zero, and the cie means that each child is chosen according to a conjunction of probabilistically independent events, which can be used globally throughout the document. This model can represent different families of p-documents in terms of the types of distributional nodes that are allowed which are denoted in the form of $\mathrm{PrXML}^{\{\mathrm{type}_1,\mathrm{type}_2,\ldots\}}$. For example, the model in Kimelfeld et al. (2008) is the same as that using the ind and mux types, and the model in Hung et al. (2003a, 2003b) is using the exp type if the data is tree-structured. Moreover, Kimelfeld et al. (2008) and Abiteboul et al. (2009) consider the expressive power of the families of p-documents. They study the possibility of translating p-documents between families. In Cohen et al. (2009), this model is extended to represent a probabilistic XML data as a p-document and a set of constraints.

2.2. Interval-based labeling scheme

In the interval-based labeling scheme, such as Li and Moon (2001) and Zhang et al. (2001), the label of any node in an XML data tree is represented as the tuple (DocID, StartPos, EndPos, LevelNum), where (i) DocID is the identifier of the document; (ii) StartPos and EndPos can be generated by counting the number of words from the beginning of the document with identifier DocID to the start of the element and end of the element, respectively, or by the sequential assignment of positive integers during the depth first traversal of the XML data tree; and (iii) LevelNum is the depth of the element in the document.

The ancestor–descendant or parent–child relationship between two tree nodes N_1 and N_2 whose positions are recorded in this fashion, such as (D_1, S_1, E_1, L_1) and (D_2, S_2, E_2, L_2), can be determined easily: (i) *ancestor–descendant*: N_2 is a descendant of N_1 iff $D_1 = D_2$, $S_1 < S_2$ and $E_2 < E_1$; (ii) *parent–child*: N_2 is a child of N_1 iff $D_1 = D_2$, $S_1 < S_2$, $E_2 < E_1$ and $L_1 + 1 = L_2$.

There are major indications about efficiency of labeling schemes: the efficiency of the query processing using labels, the size of labels and the easiness of update processing. Firstly, if a labeling scheme can be used to identify relationships among nodes, then it increases the efficiency of the query processing using labels. For example, the interval-based labeling schemes have strong advantages in processing queries including ancestor–descendant relationships. Secondly, if the size of labels is smaller than others, then the labeling scheme has the efficiency for the size of labels. For example, the interval-based labeling schemes have good efficiency in the perspective of the size of labels, whereas in the prefix labeling schemes, the size of labels is large and incurs high storage overhead. Lastly, when there are insertions and deletions of nodes, if they do not change the labels of many existing nodes, then it is efficient in the perspective of the easiness of update processing.

3. Preliminaries

3.1. Probabilistic XML tree

Generally, the data model for probabilistic XML should present a probabilistic distribution over a set of ordinary XML documents. A specific model provides a mechanism for defining that distribution in terms of probabilistic process that generates possible XML trees with probabilities.

In this paper, we present the *probabilistic XML tree* as the data model to represent the probabilistic XML data. The probabilistic XML tree is defined as a node-labeled tree with two types of nodes, ordinary and distributional. The ordinary node indicates elements, attributes or texts in probabilistic XML data, and the labels of ordinary nodes are element names, attribute names or text values. For the distributional node, the label of a node is the type of the distributional node such as ind, mux or exp. A distributional node specifies a distribution over the subsets of its children. In the case of ind or mux, each child of the distributional node has one probability. Specially for the mux, the sum of all children's probabilities cannot be over one and at most one child can be chosen. In the case of exp, a probability is given to each subset in the finite subsets of the exp distributional node's children and the sum of all subset's probabilities equals one. Moreover, a node of type exp can choose exactly one subset. Surely a distributional node is neither the root nor a leaf.

The probabilistic XML tree in this paper is the family $\text{PrXML}^{\{\text{ind,mux,exp}\}}$ in Kimelfeld et al. (2008) which can be efficiently translated into the family $\text{PrXML}^{\{\text{exp}\}}$ that is the most expressive family with efficient evaluation algorithms for query trees. Therefore we consider the probabilistic XML data model which is the most expressive among the previously proposed ones which could have efficient query evaluation algorithms. Even though the family $\text{PrXML}^{\{\text{ind,mux,exp}\}}$ can be translated into the family $\text{PrXML}^{\{\text{exp}\}}$, we treat the family $\text{PrXML}^{\{\text{ind,mux,exp}\}}$ because the data represented by the family $\text{PrXML}^{\{\text{exp}\}}$ has more distributional nodes and is more hierarchical than the data represented by the family $\text{PrXML}^{\{\text{ind,mux,exp}\}}$.

Definition 1. The probabilistic XML tree is the node labeled tree $T = (V, E)$, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges. $V = V_{ord} \cup V_{dist}$, where V_{ord} and V_{dist} are the disjoint sets of ordinary and distributional nodes, respectively. Each node v has the label $l(v)$. Especially the label $l(v)$ of the distributional node v is ind, mux or exp, and a distributional node v has the probabilistic distribution `dist` over its children $v_1, v_2, \ldots, v_n$ as follows:

1. ind type: $\texttt{dist} = \{(v_k, p_k) | 1 \leqslant k \leqslant n\}$, v chooses v_k with probability p_k.
2. mux type: $\texttt{dist} = \{(v_k, p_k) | 1 \leqslant k \leqslant n \text{ and } \sum p_k \leqslant 1\}$, v chooses at most one child v_i with the probability p_i.
3. exp type: $\texttt{dist} = \{(s_k, p_k) | 1 \leqslant k \leqslant m \leqslant 2^n \text{ and } \sum p_k = 1\}$, v chooses exactly one subset s_i with the probability p_i among the m subsets which are given probabilities.

The probabilistic XML tree is similar to the model used in previous works (Kimelfeld et al., 2008), and we used the notions from Kimelfeld et al. (2008) such as PrXML, ind, mux and exp. However, the representation of the probabilistic distribution is different from that in Kimelfeld et al. (2008). While only the requirements of a distributional node for the distribution over the subsets of its children are given in Kimelfeld et al. (2008), in this paper, we give specific definition for each distributional node as a set of pairs each of which is composed of (1) a children node and the probability of the node for ind and mux and (2) a subset of children and the probability of the subset for exp.

As the previous researches (Abiteboul and Senellart, 2006; Cohen et al., 2009; Hung et al., 2003a, 2003b; Kimelfeld et al., 2008; Li et al., 2006; Nierman and Jagadish, 2002), each probability in the source XML data is assigned conditioned on the fact that the parent element exists and we call it *conditional probability*. For the probabilistic distribution $\texttt{dist} = \{(v_k, p_k) | 1 \leqslant k \leqslant n\}$ of each distributional node v with the type ind or mux, p_k is the probability $\texttt{Prob}(v_k | p(v))$, where $p(v)$ is the parent node of v. When the type of distributional node is exp, the p_k is the probability that all nodes in a subset s_k simultaneously exist given that the parent exists.

Actually, the probability that each node exists in possible XML trees could be computed by multiplying the conditional probabilities along the path from the node to the root by the Bayes' formula as shown in Nierman and Jagadish (2002) and Li et al. (2006), and we call it *existence probability*. Therefore, to compute the probability of a query result, it is inevitable to access all the ancestors of the nodes in a query result to get the conditional probabilities of ancestors.

An example for the probabilistic XML tree is shown in Fig. 2. Each node has an identifier that is marked inside the node. The ordinary node is represented by a circle and the label is marked at the side of that circle. And the distributional node is represented by a rectangle inside which the type is marked. Each distributional node has the probabilistic distribution `dist` over its children which is represented under the tree.

In order to get possible XML trees from a given probabilistic XML tree, firstly one subset of each distributional node is chosen and the remaining children and their descendants are deleted. And then all the distributional nodes are removed. At this time, if an ordinary node loses the parent, then the proper ancestor becomes the new parent of that.

3.2. Probabilistic XML query tree

Generally, an XML query tree is the node-labeled tree with parent–child edges depicted using a single line and ancestor–descendant edges depicted using a double line. The label of node could be an element name, an attribute name, a text value or $*$, where $*$ is satisfied by every label. Fig. 3 shows examples of the XML query tree.

A match of an XML query tree to an XML document is a mapping from the nodes of the query tree to those of the document, such that query node value predicates are satisfied by the corresponding data node, and the axes (/ or //) between query nodes are satisfied by the structural relationships between the corresponding data nodes. The XML query result is a set or a list of matches.

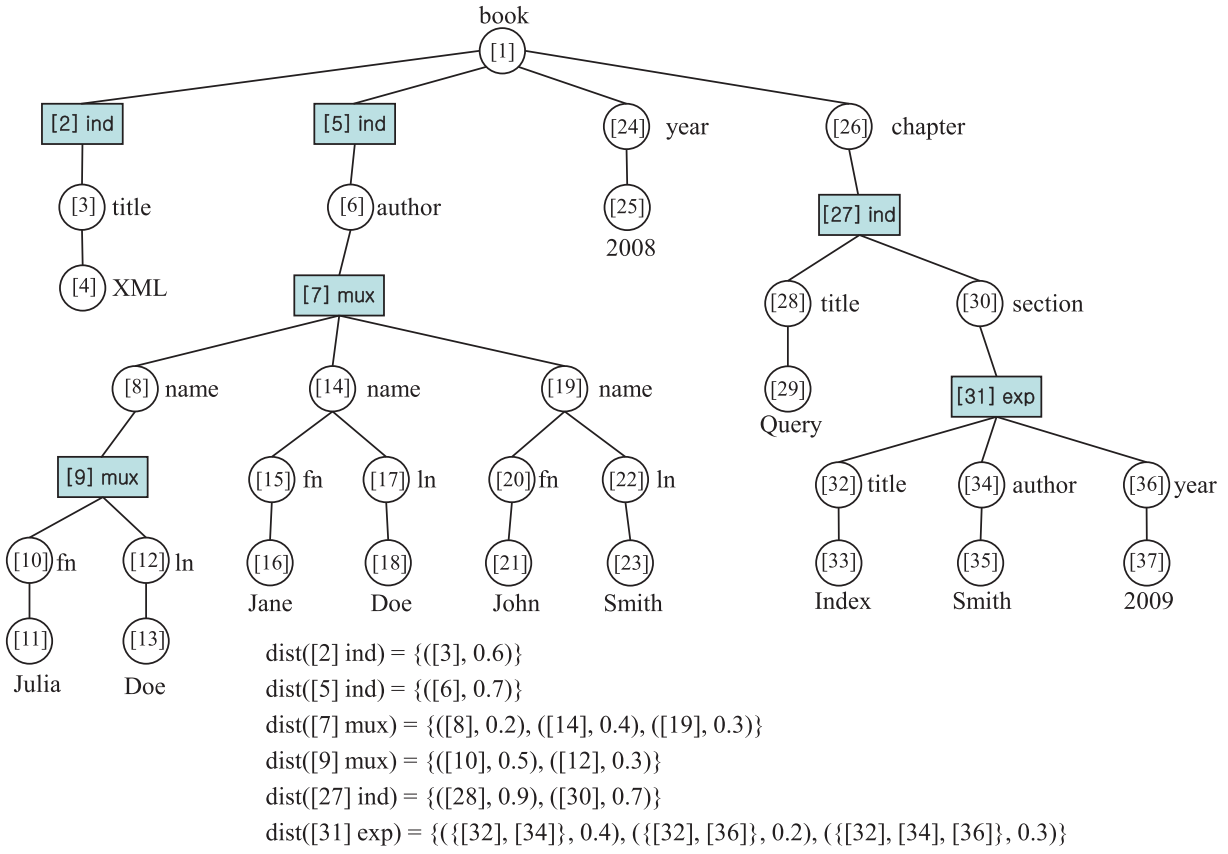

Fig. 2. Probabilistic XML tree.

For example, the answer of the query tree in Fig. 3b can be a list of matches. In this case, there is a match (1, 3, 6, 4, 15, 17, 16, 18) identified by node ID's for (book, title, author, XML, fn, ln, Jane, Doe). Therefore, the answer for this example is a list which is composed with a match (1, 3, 6, 4, 15, 17, 16, 18). The hierarchical structure of the answer can be rebuilt by referencing the query tree. The query tree has the hierarchical structure. One element of the list is matched to the query tree and we know that XML is the title of the book and that Jane Doe is the name of the author according to the query tree.

The XML query tree can be used in order to issue a query over probabilistic XML data without recognizing the representation format or the storage method for probability information. In this case, the XML query tree can be adopted without modifications. However, the user can require the restrictions on the values of probabilities as followings: (i) restrictions on the probabilities of matches in the query result such as ranking or a threshold and (ii) restrictions on probabilities of elements, attributes or values given in the query.

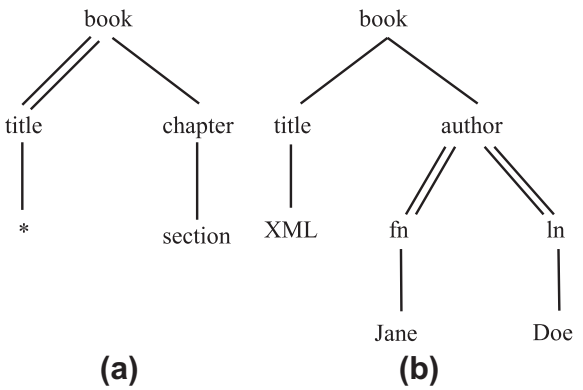

Fig. 3. XML query trees.

To support the case (ii), the syntax of XML query tree should be modified to represent the conditions for probabilities of elements, attributes or values.

In this paper, we define *probabilistic XML query tree* that attaches the conditions for probabilities.

Definition 2. The probabilistic XML query tree is the node-labeled tree with parent–child edges depicted using a single line and ancestor–descendant edges depicted using a double line. Each node v has the label $l(v)$ which could be an element name, an attribute name, a text value or $*$. Also each node can have the probability conditions, which use an equality or an inequality operator. The probability can be either conditional probability p_c or existence probability p_e.

Fig. 4 shows an example of the probabilistic XML query tree which limits the title element with existence probability more than 0.2 and the author element with conditional probability greater than 0.3 and less than 0.7.

4. Extended interval-based labeling scheme

Given a probabilistic XML tree t and a probabilistic XML query tree q, the evaluation of q over t is finding all matches of q over all possible XML trees with probabilities that represent the certainty of obtaining a match when querying a possible XML tree. If some nodes in q have probability conditions, all matches should satisfy them. However, it is not efficient to evaluate a query over the whole possible XML trees because the number of possible XML trees is huge. Therefore, it is necessary to evaluate the query over a probabilistic XML tree without enumerating all possible XML trees.

The previous techniques for probabilistic XML query processing mostly take an approach that firstly matches data over a given probabilistic XML tree without considering the distributional nodes and probabilities, and then evaluates the probability of each match. At the first step, the data matches can be a superset of the result. The distributional nodes and probabilities are considered when evaluating the probability of each match.

Several methods have been proposed for efficient XML query processing. Especially the interval-based labeling scheme can decide the ancestor–descendant relationships easily only with integer comparisons and it can be used to find matches in probabilistic XML query processing. However, the set of resulting matches is a superset of the exact result, and the calculation of the probability of a match in the result requires the navigation from the root to the nodes in the match. Therefore, additional data access is needed to treat the distributional nodes and probabilities.

In this paper, we propose an extended interval-based labeling scheme which makes it possible to avoid the additional data access and to process probabilistic XML queries only by accessing the labels of nodes in the query. By the extended interval-based labeling scheme, the label of each node in a probabilistic XML tree can be represented as (`DocID`, `StartPos`, `EndPos`, `Level`, `Cprob`, `Eprob`) for an ordinary node and (`DocID`, `StartPos`, `EndPos`, `Level`, `ChildList`) for a distributional node, where (i) `DocID` is the identifier of the document; (ii) `StartPos` and `EndPos` can be generated by the sequential assignment of positive integers during the depth first traversal of the probabilistic XML tree; (iii) `Level` is the depth of the node; (iv) `Cprob` and `Eprob` are the values of conditional probability and existence probability of the node, respectively; and (v) `ChildList` is the list of `StartPos`'s of the children. In the case of `exp`, the conditional probability `Cprob` of each node is the sum of all probabilities each of which is the probability of the subset that includes the node.

Fig. 5 shows the probabilistic XML tree in Fig. 2 with node labels which are generated by the extended interval-based labeling scheme. According to presentation convenience, `Cprob`'s of a distributional node's children are marked at the edges between the distributional node and children, and `DocID`, `Level` and `Cprob` are omitted in the labels. Each distributional node's probabilistic distribution `dist` over its children is substituted to the one in which each node's identifier is converted to the `StartPos` of that, and is represented under the tree. The `exp` node with the identifier 31 appears to have the probabilistic distribution that is composed of three subsets with probabilities 0.4, 0.2 and 0.3. However, the probabilistic distribution implies that an empty set with probability 0.1 is included.

When using the extended interval-based labeling scheme, the parent–child or ancestor–descendant relationship between two tree nodes can be determined easily the same as the original interval-based labeling scheme. Moreover, the label of

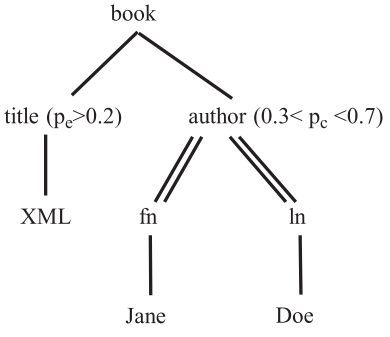

Fig. 4. Probabilistic XML query tree.

distributional node can remove needless matches, and the value of `Eprob` in the label of each ordinary node can prevent the additional data access for evaluating the probability of the result.

When data updates occur, the extended interval-based labeling scheme has to re-compute the existence probabilities and change the values in the labels which are included in the subtree rooted by the node updated. We consider it as future works to efficiently handle this issue.

5. Probabilistic XML query processing

The probabilistic XML query tree can be considered in two cases, without or with probability conditions in nodes. The first case is similar to the previous XML query processing, but we should treat the distributional nodes and evaluate the probabilities of results because the data is given by a probabilistic XML tree. The second case requires not only the same processes in the first case, but also checking the probability conditions of nodes. In the next two subsections, we describe the probabilistic XML query processing in the two cases.

5.1. Without probability conditions in nodes

In this subsection, we describe a method to process a probabilistic XML query tree without probability conditions in nodes. The data is a probabilistic XML tree labeled by the extended interval-based labeling scheme. Each node in the probabilistic XML tree has the label by the extended interval-based labeling scheme and these labels are managed by lists. A distinct node name is associated with a list of labels each of which is given to a node with the name.

The parent–child or ancestor–descendant relationship between two nodes in an XML query tree such as `book/author` or `book//name` can be processed by using two lists of nodes' labels in the given probabilistic XML tree. For example, if considering the relationship `book/author`, we must process the structural join between the list of labels for `book` and that for `author` sorted in `StartPos`.

However, to consider an XML query tree with more than two nodes, we should split the XML query tree into several binary structural relationships which are XML queries with two nodes such as `book/author`. And then we process all binary structural relationships and join the results of them in order to get the result of the XML query tree.

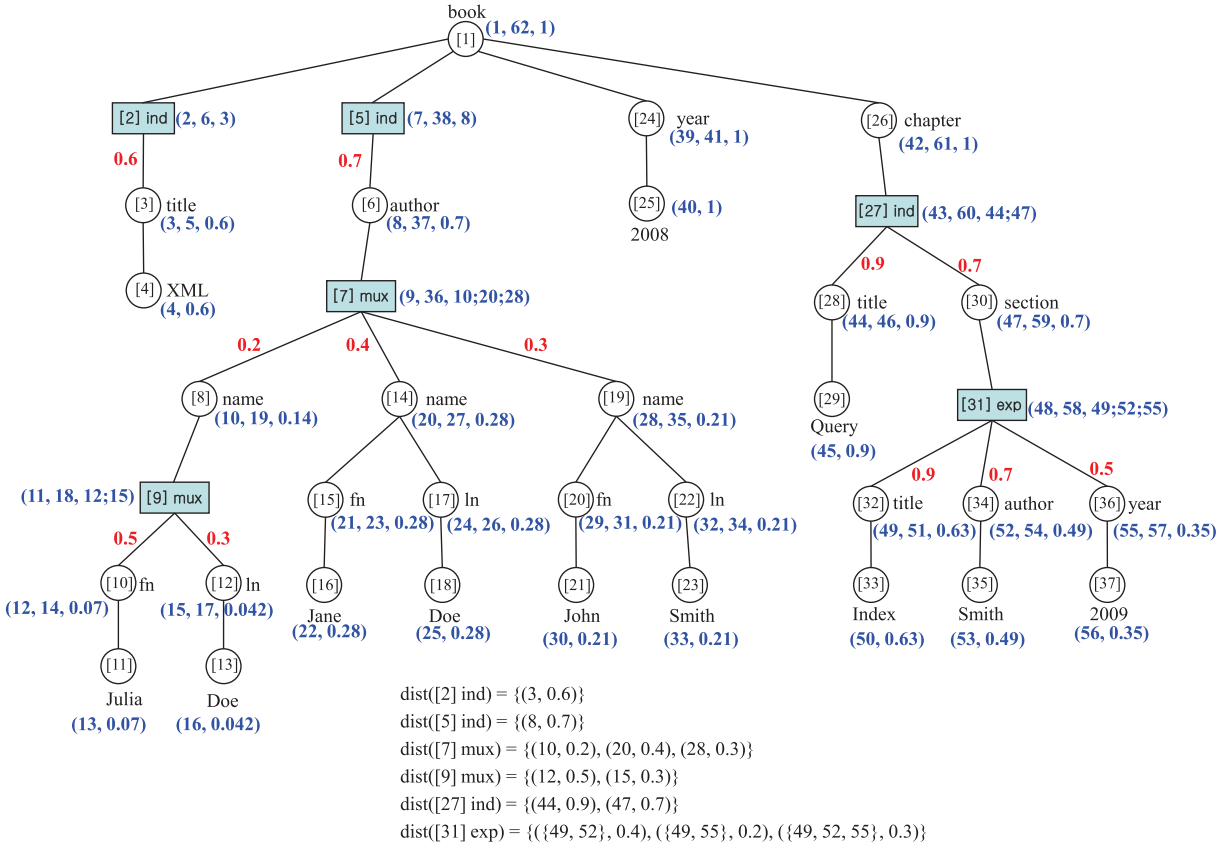

Fig. 5. Extended interval-based labeling.

When joining the results of binary structural relationships, we should consider mutually exclusive or explicit relationships which can be considered by mux or exp distributional nodes in the probabilistic XML tree.

For example, when we process the XML query tree in Fig. 6c over the probabilistic XML tree in Fig. 5, if we do not consider the distributional nodes, the result includes (12, 15), (12, 24), (12, 32), (21, 15), (21, 24), ..., (29, 32) as the pairs of (fn, ln). However, if we consider the mux distributional node, the result should include only (21, 24), (29, 32) as the pairs of (fn, ln). At this time, each node is identified by the StartPos value.

Accordingly, when we do not consider distributional nodes, the result of query processing can be a superset of the result when considering distributional nodes. In other words, it means that unnecessary processing is done for joining the results of binary structural relationships. This cannot occur in simple path queries, but can occur in query trees with branch nodes.

A distributional node such as mux or exp can restrict the existence of descendants of the distributional node. In the case of mux, at most one child can exist. Therefore, the children of a node, in a probabilistic XML tree which has the same label as a branch node of a query tree, cannot exist at the same time if a mux node exists under the node. In Fig. 6a, fn and ln are children of the branch node name. If this query is processed over the data in Fig. 5, the name with startPos 10 has mux under it and there are fn and ln under the mux. But these fn and ln cannot exist at the same time because of the mux and cannot be included in the result. Therefore, we can improve the cost of joining process by removing these not-possible answers. This feature can be considered for exp node similarly.

As the above, when we consider the distributional node in query processing, the branch node in the query tree should be treated properly and it is better to consider all the children of the branch node together than to consider each root-to-leaf path separately.

Definition 3. For any two nodes n_1 and n_2 in a probabilistic XML tree, $n_1 \nsim n_2$ denotes that n_1 *cannot exist simultaneously* with n_2. $n_1 \nsim n_2$ means that the probability that n_1 and n_2 exist simultaneously is zero. Otherwise, n_1 and n_2 can exist simultaneously, denoted by $n_1 \sim n_2$.

Lemma 1. *Let node a be the ancestor of node d. If d exists, then a exists.*

Proof. This lemma holds by the property of the tree structure. □

Theorem 1. *For any two nodes n_1 and n_2 in a probabilistic XML tree, if $n_1 \nsim n_2$, then n_1 or any descendant of n_1 cannot exist simultaneously with n_2 or any descendant of n_2.*

Proof. Suppose that n_1 or a descendant of n_1, denoted by d_1, can exist simultaneously with n_2 or a descendant of n_2, denoted by d_2. If d_1 and d_2 exist simultaneously, then, n_1 exists and n_2 exists by Lemma 1. Therefore n_1 and n_2 can exist simultaneously. □

In a probabilistic XML tree, two nodes n_1 and n_2 such that $n_1 \nsim n_2$ can occur by mux or exp distributional nodes. The following shows how $n_1 \nsim n_2$ occurs by mux or exp, and Fig. 7 shows the case of mux.

- In the case of mux
  - By Definition 1 of mux, for the children $c_1, c_2, \ldots, c_n$ of mux node, $c_i \nsim c_j$ holds if c_i and c_j are two different children of the mux.
  - By Theorem 1, for any two different children c_i and c_j of mux node, $n_1 \nsim n_2$ holds if n_1 is included in the c_i-rooted subtree and n_2 is included in the c_j-rooted subtree.
- In the case of exp
  - By Definition 1 of exp, for the children $c_1, c_2, \ldots, c_n$ of exp node, $c_i \nsim c_j$ hold if c_i and c_j are not included in the same probability subset of the exp.

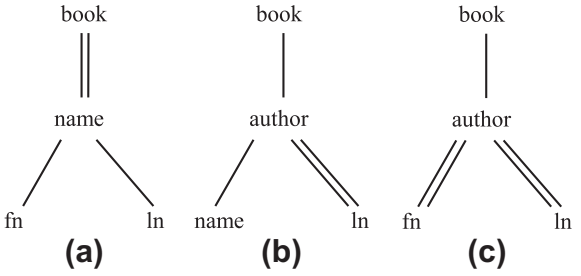

Fig. 6. Three cases of XML query.

– By Theorem 1, for any two different children c_i and c_j of exp node which are not included in the same probability subset of the exp, $n_1 \nsim n_2$ holds if n_1 is included in the c_i-rooted subtree and n_2 is included in the c_j-rooted subtree.

By checking *cannot exist simultaneously* relationships between data nodes, we can find not-possible answers before joining the results of binary structural relationships. In order to find not-possible answers, we examine an instance of matched data nodes for a branch node and all its children in a query tree.

Consider the branch node q_r and its all children $q_1, q_2, \ldots, q_n$ in a part of probabilistic XML query tree in Fig. 8a. Each of the edges between q_r and $q_1, \ldots, q_n$ can be a single line or a double line. Let the matched data nodes be $r, d_1, \ldots, d_n$ as in Fig. 8b. Then for all i ($i = 1, \ldots, n$), r.startPos < d_i.startPos and d_i.endPos < r.endPos hold. At this time, if there is neither mux nor exp distributional node which is descendant of r, then these data nodes can be a possible answer. Otherwise, that is if there is mux or exp distributional node, we can find not-possible answers by checking *cannot exist simultaneously* relationships occurred by mux or exp. The distributional node mux or exp can include all $d_i (1 \leqslant i \leqslant n)$ as in Fig. 8c or only a part of $d_i (1 \leqslant i \leqslant n)$ as in Fig. 8d. Therefore, we should check *cannot exist simultaneously* relationships for the d_i's whose `startPos` and `endPos` are included in the `startPos` and `endPos` of the distributional node.

The mux or exp distributional nodes which are descendants of r can be found by checking whether distributional node's position is included in the r's position (r.startPos, r.endPos). That is checking the ancestor–descendant relationship between r and mux or exp. For those mux or exp nodes, we can find not-possible answers by checking the following conditions. At this time, mux or exp node has a `ChildList` such as $(s_1; s_2; \ldots; s_m)$ and let s_{m+1} be the `endPos` of the distributional node. Note that exp node has probability subsets.

- In the case of mux
  - If there are d_i and d_j ($d_i \neq d_j$, mux.startPos < d_i.startPos, d_i.endPos < mux.endPos, mux.startPos < d_j.startPos and d_j.endPos < mux.endPos) such that $s_k \leqslant d_i$.startPos, d_i.endPos $\leqslant s_{k+1}$ and $s_l \leqslant d_j$.startPos, d_j.endPos $\leqslant s_{l+1}$ and $s_k \neq s_l (1 \leqslant k, l \leqslant m)$, then $(r, d_1, \ldots, d_n)$ is not a possible answer.
  - If there is only one $j (1 \leqslant j \leqslant m)$ such that for all d_i (mux.startPos < d_i.startPos and d_i.endPos < mux.endPos), $s_j \leqslant d_i$.startPos and d_i.endPos $\leqslant s_{j+1}$, then $(r, d_1, \ldots, d_n)$ is a possible answer.
- In the case of exp
  - If there are d_i and d_j ($d_i \neq d_j$, exp.startPos < d_i.startPos, d_i.endPos < exp.endPos, exp.startPos < d_j.startPos, d_j.endPos < exp.endPos) such that $s_k \leqslant d_i$.startPos, d_i.endPos $\leqslant s_{k+1}$ and $s_l \leqslant d_j$.startPos, d_j.endPos $\leqslant s_{l+1} (1 \leqslant k, l \leqslant m)$ and s_k and s_l are not included commonly in any probability subset, then $(r, d_1, \ldots, d_n)$ is not a possible answer.
  - If there is a probability subset set_p such that for all d_i (exp.startPos < d_i.startPos and d_i.endPos < exp.endPos), there is s_k in set_p such that $s_k \leqslant d_i$.startPos and d_i.endPos $\leqslant s_{k+1}$, then $(r, d_1, \ldots, d_n)$ is a possible answer.

After checking *cannot exist simultaneously* relationships, we should evaluate the probability of each data match. If a query is a simple path query, then the result probability is the existence probability in the label of the result's leaf node. However, a query tree with branch nodes is different. Consider the matched data nodes in Fig. 8b. If $d_1, \ldots, d_n$ are leaf nodes in the matched data tree and there is neither mux nor exp distributional node which is descendant of r, then the result probability is the product of the existence probabilities of $d_1, \ldots, d_n$ divided by $(n - 1)$ times of the existence probability of r. This means that the repeatedly multiplied part is eliminated by the division.

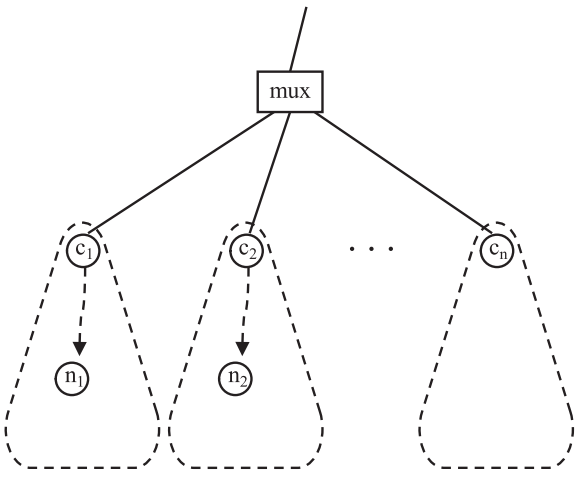

Fig. 7. mux node and descendants.

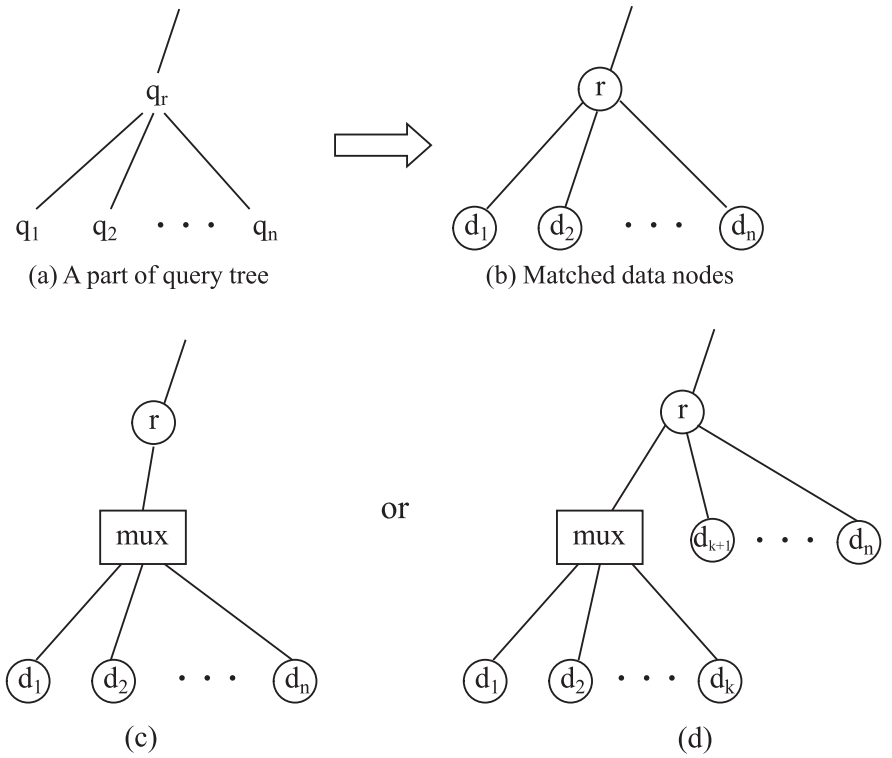

Fig. 8. mux's position in matched data nodes.

However, if there is a mux or exp distributional node which is a descendant of r and an ancestor of some d_i's, then we consider the distributional node in the result probability calculation. The following are the methods to evaluate the result probability in these cases. At this time, the mux or exp node has a `ChildList` such as (s_1: s_2: ... : s_m), s_{m+1} is the `endPos` of the distributional node, $d_1, \ldots, d_k$ are included in (r.startPos, r.endPos) and $d_{k+1}, \ldots, d_n$ are not included in it as in Fig. 8d.

- In the case of mux
  - Let s_j be the one among $s_1, \ldots, s_m$ such that for all d_i ($1 \leqslant i \leqslant k$), $s_j \leqslant d_i$.startPos and d_i.endPos $\leqslant s_{j+1}$.

$$P = \frac{d_1.eprob \times \cdots \times d_n.eprob}{s_j.eprob^{k-1} \times r.eprob^{n-k}}$$

- In the case of exp
  - Let S be a set $\{e_1, \ldots, e_p\}$ such that for all $d_i (1 \leqslant i \leqslant k)$, there is s_l in S such that $s_l \leqslant d_i$.startPos and d_i.endPos $\leqslant s_{l+1}$, and let $s_{d_i} = s_l$ for each d_i.

$$P = \frac{d_1.eprob \times \cdots \times d_n.eprob \times eprob(\{e_1, \ldots, e_p\})}{s_{d_1}.eprob \times \cdots \times s_{d_k}.eprob \times r.eprob^{n-k}}$$

For an example of the case in which there is a mux distributional node between the branch node and its children, if the query in Fig. 6c is applied to the data in Fig. 5, the probability of an instance (1, 8, 21, 24), where numbers are the `startPos`, in the result of (`book`, `author`, `fn`, `ln`) is 0.28. This is calculated by 0.28 × 0.28/0.28 = 0.28 which means that the product of the existence probability of the `fn` with `startPos` 21 and the existence probability of the `ln` with `startPos` 24 is divided by the existence probability of the `name` with `startPos` 20. This is the case in which there is a mux distributional node between the branch node and its children.

For an example of the other case in which there is an exp distributional node between the branch node and its children, if a query //chapter/section[/title]/author is applied to the data in Fig. 5, the probability of an instance (42, 47, 49, 52), where numbers are the `startPos`, in the result of (chapter, section, title, author) is 0.28. This is calculated by (0.63 × 0.49/(0.63 × 0.49)) × 0.28 which means that the product of the existence probability of the title with `startPos` 49 and the existence probability of the author with `startPos` 52 is divided by the product of the existence probability of the title with `startPos` 49 as the exp's child including the title and the existence probability of the author with `startPos` 52 as the exp's

child including the author, and then multiplied by the existence probability 0.28(0.7×0.4) of the subset that includes the title and the author.

Fig. 9 shows the procedure for probabilistic XML query processing. The QueryProcessing procedure calls the subQueryProcessing procedure in line 2 which has two parameters, the list of data nodes' labels for the root node of the query tree and the query tree itself. After calling the subQueryProcessing procedure, the QueryProcessing procedure evaluates the probability of each data match in the result in lines 3 and 4. In lines 7–20, for each label in the list, the subQueryProcessing procedure finds a list of combinations of data nodes' labels matched for the root node's children in the given query tree. At this time, the procedure finds possible answers among the list of data nodes' labels combinations by calling the getPossibleAnswer procedure in line 18 and appends them to the output. Then, in lines 22–26, the subQueryProcessing is recursively called for each child of the root node of the query tree. After all nodes in the query tree are traversed, the result of the query is in the output parameter.

Fig. 10 shows the procedure to get possible answers. This procedure is called by the subQueryProcessing procedure in order to get the set of combinations $(r, d_1, \ldots, d_n)$ which can be the result. In lines 5–12, the getPossibleAnswer procedure shows the case when both mux nodes and exp nodes exist. Also, lines 13–17 show the case when only mux nodes exist and lines 18–22 show the case when only exp nodes exist. The procedure calls the getDistList procedure in Fig. 10 and the checkPossibleAnswer procedure in Fig. 11. The first is to get the list of distributional nodes' labels which are descendants of the given node r, and the second is to check the distributional nodes to determine whether the given combination can be included in the result. The checkPossibleAnswer procedure also evaluates the values necessary to calculate the probability of

```
procedure QueryProcessing(QT)
//QT is the probabilistic XML query tree
//output is a global variable which has the result
//output is a list of data matches and each one has prob
  that contains the probability
begin
1.   qr = root of QT;
2.   subQueryProcessing(qr.List, QT);
     //List is a list of positional labelings
       of corresponding nodes in the probabilistic tree
3.   foreach data match dm in output do
4.     dm.prob = Product of each leaf node's eprob
       divided by the product of each leaf node's bprob;
end

procedure subQueryProcessing(rootList, QT)
begin
1.    qr = root of QT;
2.    r = rootList.firstNode;
3.    outputList = null;
4.    foreach child qrc of qr do
      //qrc is the ith child of qr
5.      c_i = qrc.List.firstNode;
6.      cbegin_i = c_i;
7.    while(r != null)
8.      foreach i do
9.        c_i = cbegin_i;
10.       coutputList_i = null;
11.       while(c_i.startPos < r.startPos)
12.         c_i = c_i.nextNode;
13.       cbegin_i = c_i;
14.       while(c_i ≠ null and c_i.endPos < r.endPos)
15.         append c_i to coutputList_i;
16.         c_i = c_i.nextNode;
            //Remember the position to start
17.     D = coutputList_1 × ... × coutputList_n;
        //make the cartesian product
18.     temp = getPossibleAnswer(r, D);
19.     append temp to outputList;
20.     r = r.nextNode;
      //end of while
21.   append outputList to output;
22.   foreach child qrc of qr do
23.     if qrc is not leaf node
24.       qrcList = list of qrc's labels in the outputList;
25.       subQT = sub query tree rooted by qrc;
26.       subQueryProcessing(qrcList, subQT);
end
```

Fig. 9. Query processing procedure.

the result. In the checkPossibleAnswer procedure, the case where the type of d is mux is in lines 7–13 and the case where the type of d is exp is in lines 14–28. The bprob for each d_i is the value that is used for calculating the probability of the result.

Our query processing procedure in Fig. 9 uses the algorithm Tree-Merge-Anc (Srivastava et al., 2002) to find ancestor–descendant or parent–child relationships. According to Srivastava et al. (2002), the time and space complexities of the algorithm Tree-Merge-Anc are both $O(|AList| + |DList| + |OutputList|)$, where $|AList|$ is the size of the ancestor or parent list, $|DList|$ is the size of the descendant or child list and $|OutputList|$ is the size of the output. Our query processing procedure processes a unit of query tree at a time, where a unit is composed of a node q_r and all children q_1, q_n of the node q_r as (a) in Fig. 8. The time complexity of a unit is $O(\sum_{for\ all\ nodes\ in\ a\ unit}|inputList| + |outputList|)$, where $|inputList|$ is the size of the input list for a node in the query tree and $|outputList|$ is the size of the result for the unit. The output list will be used for the input lists of the nodes which are the roots of units in the next level. Therefore, the time complexity of our query processing is $O(\sum_{for\ all\ nodes\ in\ the\ query\ tree}|inputList| + \sum_{for\ all\ units\ in\ the\ query\ tree}|outputList| + |OutputList|)$, where $|OutputList|$ is the size of the query result, that is derived by $\sum_{for\ all\ units\ in\ the\ query\ tree}(\sum_{for\ all\ nodes\ in\ a\ unit}|inputList| + |outputList|) + |OutputList|$. This means that the time complexity depends on the size of all the intermediate results including the sum of all input lists' sizes and the size of the result of the query. This derived time complexity is consistent to the results of experiments shown in Section 6. For the space complexity, the space for the output list of each unit can be reused. Therefore, the space complexity is $O(\sum_{for\ all\ nodes\ in\ the\ query\ tree}|inputList| + \max|outputList| + |OutputList|)$.

5.2. With probability conditions in nodes

Each node in a probabilistic XML query tree can have a condition on the value of its conditional or existence probability. This probability condition can be applied to a list of labels which is accessed by a node in the query tree. However, a part of the list of labels which does not satisfy the probability condition is not necessary to be considered. Therefore, it can lead to the reduction of the time for data loading to access only the part of data that satisfies the probability condition instead of accessing the whole list of labels for the node. In order to achieve this objective, we could consider the index on the probability. Firstly, we considered the B+-tree index. The list of labels is sorted in startPos, not in the probability, so the index has

```
procedure getPossibleAnswer(r, D)
begin
1.    muxResult = null;
2.    expResult = null;
3.    mList = getDistList(r, mux.List);
4.    eList = getDistList(r, exp.List);

5.    if mList ≠ null and eList ≠ null then
6.       foreach (d1, d2, ..., dn) in D do
7.          if checkPossibleAnswer(r, m, d1, d2, ..., dn)
               for all node m in mList then
8.             append (r, d1, d2, ..., dn) to muxResult;

9.       foreach (r, d1, d2, ..., dn) in muxResult do
10.         if checkPossibleAnswer(r, e, d1, d2, ..., dn)
               for all node e in eList then
11.            append (r, d1, d2, ..., dn) to expResult;
12.      return expResult;

13.   else if eList = null then
14.      foreach (d1, d2, ..., dn) in D do
15.         if checkPossibleAnswer(r, m, d1, d2, ..., dn)
               for all node m in mList then
16.            append (r, d1, d2, ..., dn) to muxResult;
17.      return muxResult;

18.   else //mList = null
19.      foreach (d1, d2, ..., dn) in D do
20.         if checkPossibleAnswer(r, e, d1, d2, ..., dn)
               for all node e in eList then
21.            append (r, d1, d2, ..., dn) to expResult;
22.      return expResult;

procedure getDistList(r, distList)
begin
1.   output = null;
2.   d = distList.firstNode;
3.   while(d != null and d.startPos < r.endPos)
4.      d = d.nextNode;

5.   while(d != null and d.endPos < r.endPos)
6.      if(r.startPos < d.startPos and d.endPos < r.endPos)
7.         append d to output;
8.   return output;
end
```

Fig. 10. Get possible answer procedure.

```
procedure checkPossibleAnswer(r, d, d_1, d_2, ..., d_n)
begin
1.  s_1, s_2, ..., s_m = childList of d;
2.  s_{m+1} = d.endPos;
3.  d_1, ..., d_k = d_i's included in (d.startPos, d.endPos);
4.  d_{k+1}, ..., d_n = d_i's not included in (d.startPos, d.endPos);

5.  foreach d_i (k+1 ≤ i ≤ n) do
6.    d_i.bprob = r.eprob;

7.  if d.type = mux then
8.    s = s_k (s_k ≤ d_1.startPos, d_1.endPos ≤ s_{k+1});
9.    foreach d_i (2 ≤ i ≤ k) do
10.     t = s_l (s_l ≤ d_i.startPos, d_i.endPos ≤ s_{l+1});
11.     if s ≠ t then
12.       return not-possible;
13.     d_i.bprob = s.eprob;

14. else // d.type = exp
15.   S = {s_k | s_k ≤ d_1.startPos, d_1.endPos ≤ s_{k+1}};
16.   s_{d_1} = s_k;
17.   foreach d_i (2 ≤ i ≤ k) do
18.     S = S ∪ {s_l | s_l ≤ d_l.startPos, d_i.endPos < s_{l+1}};
19.     s_{d_i} = s_l;
20.     if there is no subset in dist of d including S
21.       then return not-possible;
22.   if S includes only one s then
23.     foreach d_i (2 ≤ i ≤ k) do
24.       d_i.bprob = s.eprob;
25.   else // S = {e_1, ..., e_p}
26.     d_1.bprob = s_{d_1}.eprob / eprob({e_1,...,e_p});
27.     foreach d_i (2 ≤ i ≤ k) do
28.       d_i.bprob = s_{d_i}.eprob ;
29. return possible;
end
```

Fig. 11. Check possible answer procedure.

no choice but to be a unclustering index. Therefore, with the B+-tree index on the probability, the data access cannot be efficient because the index is not clustered, and additionally, we should sort the list in startPos after accessing the part of the list by the B+-tree index.

Secondly, we consider a new lightweight index which partitions $(0, 1]$ into several blocks and keeps the information of each block to access the labels having the probability value within the interval of the block. This method can access any part of labels in the list while keeping them sorted in startPos. To apply this new index, we should derive the method to partition $(0, 1]$ into several blocks and the method to access the proper data in the list.

In partitioning $(0, 1]$, we could consider a method which uses the distribution of probabilities and queries. However, in order to calculate the distribution of probabilities, we should access the whole list of labels and need to do additional tasks such as sorting the whole list of probabilities, which can be a significant overhead. Therefore, we assume that the distribution of probabilities and queries are not given, and we take the method that partitions $(0, 1]$ into blocks with the same interval.

To access the labels in the list whose probabilities are in a probability interval of the index, we consider two methods. The first is keeping the start position and the end position for each partition, and the second is keeping the start position and the linked list for each partition. In the first method, the start position and the end position are the links to the first label and the last one whose probabilities are included in the interval of the partition. In this case, when we consider the data of one partition, we should access all the labels from the start position to the end position which can include unnecessary labels not included in the interval. Therefore, if the data is distributed uniformly and the size of the list is large, we are apt to access a wide range of the list. Also, in the worst case, one partition has pointers which result in accessing the whole list.

In the second one, keeping the start position and the linked list for each partition, the items of the linked list are the labels whose probabilities are included in the interval of the partition. Using this method, although we should manage the linked list for each partition, we can access only the proper data. In this paper, taking the second one, we propose a lightweight index for the probability and call it the *p-index*.

In the p-index, the values of the probability are partitioned into eleven blocks such as $(0.0, 0.1)$, $[0.1, 0.2)$, $[0.2, 0.3)$, $\ldots$, $[0.9, 1.0)$, $[1.0)$ and the start pointer and count are managed for each block. And a linked list is kept to link all data included in one block. The common feature of all data in one block is that the first number under the decimal point is the same. The start pointer of a block b is the pointer to the first label whose probability is included in the interval of the block b and the count of a block b has the number of all data nodes whose labels are included in the block b. Fig. 12 shows the p-index constructed for the list of labels of `name`. For each distinct node name, the p-index for conditional probability and the p-index for existence probability can be constructed. The data in Fig. 12 is not matched to the data in Fig. 5. In Fig. 12, we add more data to represent the characteristics of the p-index sufficiently.

An insertion in the p-index occurs when a new data node is inserted in a list of labels, and the position of the inserted data node is inserted in the linked list of the p-index. The insert processing of the p-index is almost the same as the insert processing of the linked list. Similarly, a deletion in the p-index occurs when a data node corresponding to a label is deleted. This is almost the same as the delete processing of the linked list.

When nodes in a query tree have probability conditions, we can reduce the time of data access by using the p-index. To get the list of labels for each node in a query tree, we can access some part of the whole list by accessing the p-index with the probability condition. For example, for the data in Fig. 12, when the probability condition of name is $p_c = 0.5$, we can access only three data nodes (10, 23, 74) by using the p-index. Otherwise, we should access the whole eleven data nodes. If the probability condition is a range, several linked lists should be considered such that they are combined by a merge sort in `startPos`. After data accessing, the query processing is similar to that in the previous subsection. Fig. 13 shows the procedure to get the list of data nodes' labels which satisfy the given probability condition.

When data updates occur, the efficient update of p-indexes should be considered because the re-computation of existence probabilities needs the update of the p-indexes which are the p-indexes for the existence probabilities. We consider it as another future work.

6. Experimental results

Our experiments were carried out on an Intel Pentium 1.7 GHz with 1 GB memory, running Windows XP. All experiments were repeated 10 times independently and the average processing time was calculated disregarding the maximum and minimum values. We implemented all procedures in Java and used the file system to store the list of labels and p-index.

The experiments described in this section use three sets of test data. They are XMark 50%, XMark 100% (XMark, 2001) and Mondial (1999). The two XMark data sets contain information about auctions and they are synthetic benchmark data sets generated by the XML Generator from XMark (2001). XMark 50% is generated with parameter 0.5 which means that 50% XMark data is generated. And XMark 100% is the full XMark data generated with parameter 1. The Mondial data set represents geographical web data in the XML format. Characteristics of these data sets are summarized in Table 1. The size in Table 1 is the disk space used to store the original XML file, the nodes is the number of nodes, and the depth is the length of the longest simple path.

In order to perform the probabilistic XML query processing, we need probabilistic XML documents. Therefore, we implemented a procedure that generates a probabilistic XML document from an ordinary one. The procedure traverses the original XML document in the depth-first order. At each node v in the original XML document, it randomly chooses the number of distributional nodes which become children of v. Then it randomly chooses the new parent of each original children of v which is chosen among v and the newly created distributional nodes. And it determines probability distributions of new created distributional nodes randomly such that the distributions satisfy the constraints of the distributional nodes.

After generating a probabilistic XML document, we apply the extended interval-based labeling scheme to the created probabilistic XML document along with evaluating the existence probability of each node. Also we construct the p-index for each list of labels that is given to each distinct node name. By one traversing of the probabilistic XML document, we can apply node labeling scheme, evaluate existence probabilities of each node, and construct the p-index simultaneously.

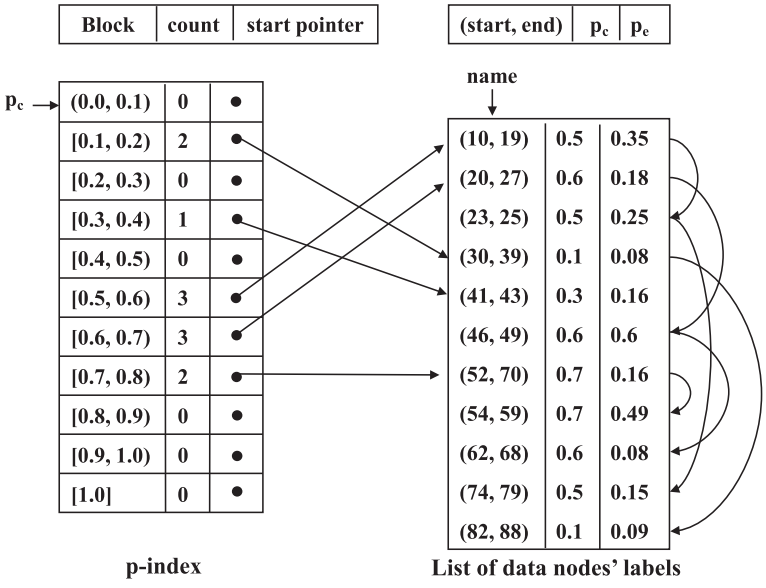

Fig. 12. Lightweight index for probability.

```
procedure getList(node, probCondition)
//Suppose that probCondition is u ≤ p ≤ v
begin
1.  b_s = index.interval(u);
2.  b_e = index.interval(v);
3.  if u == v then
4.     return b_s.list;
5.  foreach b_i from b_s to b_e
6.     linkedlist_i = b_i.list;
7.  result = mergesort(linkedlist_1, ..., linkedlist_n);
8.  return result;
```

Fig. 13. Get data using p-index procedure.

Table 1
Experimental data.

	Mondial	XMark 50%	XMark 100%
Size	1.4 MB	57.6 MB	115,7 MB
Nodes	124,763	833,157	1,666,315
Depth	5	12	12

Table 2
Query set for the comparison of extended and EvalDP.

Data set	Query	Query expression	Results
Mondial	Q1	mondial//mountain[.//height]//name	65
	Q2	mondial//country [.//government]//name	4829
	Q3	//country//city[.//name]//population	2477
	Q4	mondial//country[.//border]//population	26,022
XMark 50%	Q5	//categories//category [.//name]//text	743
	Q6	//closed-auctions//closed-auction[.//price]//date	3830
	Q7	//people//person[.//name]//address	5518
	Q8	//site//person[.//profile]//watch	10,612

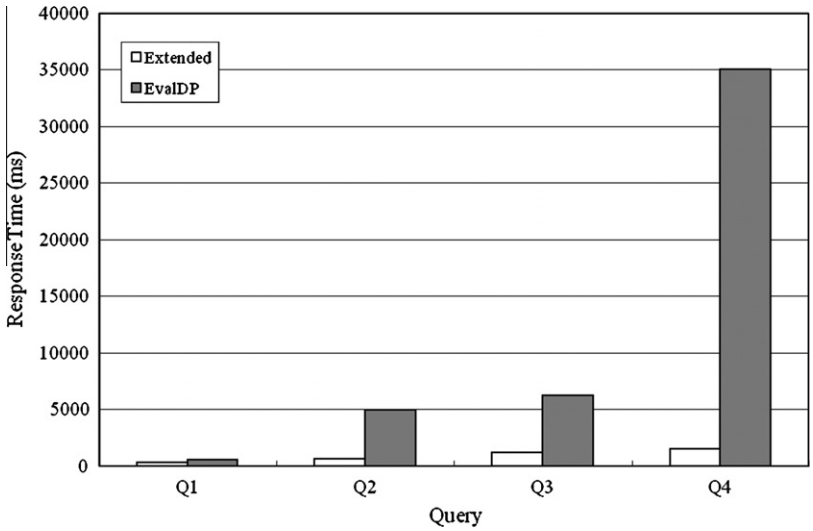

Fig. 14. Query processing for Mondial (1).

Each list of labels is stored in one distinct file, and we can access the list of labels for each distinct node name from the file. All p-indexes are stored in two files, one is for p-indexes for conditional probabilities and the other is for p-indexes for existence probabilities. We construct B+-tree index for each list of labels.

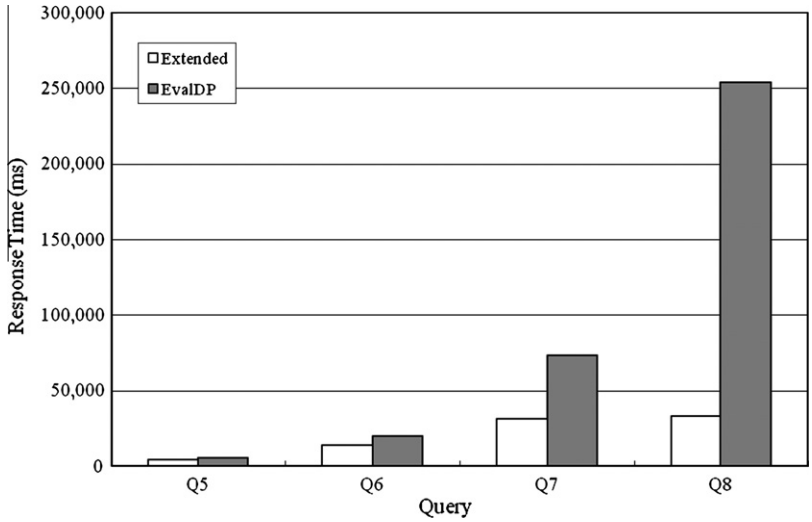

Fig. 15. Query processing for XMark 50% (1).

The size of a p-index is about 140 bytes, and it is not changed by the size of the data set because the structure of the p-index is always the same. However, the total size of all p-indexes for one XML document can be changed by the number of distinct element names, distinct attribute names and distinct text values. In our experiments, the total size of all p-indexes for the Mondial data set with 1.4 MB in size and 44 distinct nodes is about 14 KB ($\approx 140 \times 44 \times 2$). It is about 24 KB ($\approx 140 \times 78 \times 2$) for the XMark 100% data set with 115.7 MB in size and 78 distinct nodes.

6.1. Query processing without the p-index

In order to analyze the efficiency of the proposed node labeling scheme and query processing, we compare ours, which is called Extended from now, with the query processing in Kimelfeld et al. (2008), which is called EvalDP.

Since Kimelfeld et al. (2008) does not provide the query set, we create queries such that they cover different query types, various elements, diverse probability conditions, and various result sizes. We use several query trees under the two data sets, Mondial and XMark 50%, to analyze the query performance. Table 2 shows the information about experimental queries. For the Mondial data set, we use four queries from Q1 to Q4, and for the XMark 50% data set, we use four queries from Q5 to Q8.

Table 3
Query Set for the Comparison of Extended and EvalDP (Mondial data).

Query	Query expression	Results
M6-1	/mo ndial/country/province[city][name="Alsace"]	1
M6-2	/mondial/country [province/city/name]/border	263
M6-3	/mo ndial/country/province[area]/city [name]	2387
M8-1	/mondial[continent] [organization]/country/province[city][name="Alsace"]	1
M8-2	/mondial[continent] [organization]/country [province/city/name]/border	263
M8-3	/mondial[continent] [organization]/country/province[area]/city [name]	2387
M10-1	/mondial[continent] [organization] [sea] [river]/country /province[city] [name="Alsace"]	1
M10-2	/mondial[continent] [organization] [sea] [river]/country [province/city/name]/border	263
M10-3	/mondial[continent] [organization] [sea] [river] /country/province[area]/city[name]	2387
M12-1	/mondial[continent] [organization] [sea] [river] [lake] [island] /country/province[city] [name="Alsace"]	1
M12-2	/mondial[continent] [organization] [sea] [river] [lake] [island] /country [province/city/name]/border	263
M12-3	/mondial[continent] [organization] [sea] [river] [lake] [island] /country/province [area]/city [name]	2387
M14-1	/mondial[continent] [organization] [sea] [river] [lake] [island] [mountain] [desert]/country/province [city] [name="Alsace"]	1
M14-2	/mondial[continent] [organization] [sea] [river] [lake] [island] [mountain] [desert]/country [province/city/name]/border	263
M14-3	/mondial[continent] [organization] [sea] [river] [lake] [island] [mountain] [desert]/country/province [area]/city [name]	2387
M16-1	/mondial[continent] [organization] [sea] [river] [lake] [island] [mountain] [desert]/country/province [city] [name="Alsace"] [area] [population]	1
M16-2	/mondial[continent] [organization] [sea] [river] [lake] [island] [mountain] [desert]/country [name] [population] [province/city/name] /border	263
M16-3	/mondial[continent] [organization] [sea] [river] [lake] [island] [mountain] [desert]/country/province [name] [area] [population]/city [name]	2387

Table 4
Query set for the comparison of extended and EvalDP (XMark 50% data).

Query	Query expression	Results
X6-1	/site[regions]/people/person[name="Wissal Rathsack"]	1
X6-2	/site[regions]/people/person/profile/business	5839
X6-3	/site[regions]/people/person/watches/watch	23,657
X8-1	/site[regions] [categories] [catgraph]/people/person[name="Wissal Rathsack"]	1
X8-2	/site[regions] [categories] [catgraph]/people/person/profile/business	5839
X8-3	/site[regions] [categories] [catgraph]/people/person/watches/watch	23,657
X10-1	/site[regions] [categories] [catgraph] [open-auctions] [closed-auctions]/people/person[name="Wissal Rathsack"]	1
X10-2	/site[regions] [categories] [catgraph] [open-auctions] [closed-auctions]/people/person/profile/business	5839
X10-3	/site[regions] [categories] [catgraph] [open-auctions] [closed-auctions]/people/person/watches/watch	23,657
X12-1	/site[regions] [categories] [catgraph] [open-auctions] [closed-auctions]/people/person[name="Wissal Rathsack"] [emailaddress] [creditcard]	1
X12-2	/site[regions] [categories] [catgraph] [open-auctions] [closed-auctions]/people/person[name] [emailaddress]/profile/business	5839
X12-3	/site[regions] [categories] [catgraph] [open-auctions] [closed-auctions]/people/person[name] [emailaddress]/watches/watch	23,657
X14-1	/site[regions/africa/item] [categories] [catgraph] [open-auctions] [closed-auctions]/people/person[name="Wissal Rathsack"] [emailaddress] [creditcard]	1
X14-2	/site[regions/africa/item] [categories] [catgraph] [open-auctions] [closed-auctions]/people/person[name] [emailaddress] / profile/business	5839
X14-3	/site[regions/africa/item] [categories] [catgraph] [open-auctions] [closed-auctions]/people/person[name] [emailaddress] / watches/watch	23,657
X16-1	/site[regions/africa/item] [categories/category/name] [catgraph] [open-auctions] [closed-auctions]/people/ person[name="Wissal Rathsack"] [emailaddress] [creditcard]	1
X16-2	/site[regions/africa/item] [categories/category/name] [catgraph] [open-auctions] [closed-auctions]/people/person[name] [emailaddress] /profile/business	5839
X16-3	/site[regions/africa/item] [categories/category/name] [catgraph] [open-auctions] [closed-auctions]/people/person[name] [emailaddress] /watches/watch	23,657

In Table 2, the Results shows the number of combinations in query results for 1.4 MB Mondial data and 57.6 MB XMark 50% data.

Figs. 14 and 15 show the results of query processing for Mondial data set and XMark 50% data set, respectively. The performance of Extended is far better than EvalDP. Especially, as the number of combinations in query results increases, the query processing time of EvalDP increases remarkably. In EvalDP, the query processing procedure is composed of two phases. The first phase is to find data matches without considering distributional nodes. Accordingly the data matches contain not possible answers. In the second phase, EvalDP repeatedly accesses the probabilistic XML document to evaluate the probability of each data match. Therefore, as the number of combinations in query results increases, the performance of EvalDP is degraded.

While the experimental queries in Table 2 have only 4 nodes, we additionally use harder and more complex queries for experiments which have 6, 8, 10, . . . , 16 nodes as in Kimelfeld et al. (2008). For two data sets, Mondial and XMark 50%, we create 18 queries for each data set that are three queries for each query node size. In other words, there are three 6-nodes queries, three 8-nodes queries, three 10-nodes queries, etc. Tables 3 and 4 show the information about these experimental queries. For the Mondial data set, we use 18 queries from M6-1 to M16-3 as shown in Table 3, and Mx-y means that the query has x nodes and if $y = 1$, then the number of the query result is 1, if $y = 3$, then the number of the query result is large, and if $y = 2$, then the number of the query result is more than the case $y = 1$ and less than the case $y = 3$. For the depth property, shallow, medium and deep queries are spread over the whole queries. For the ratio of the descendant edges property, we choose parent–child edges because the EvalDP is more efficient in the case of parent–child edges and our approach shows similar performance for both parent–child relationships and ancestor–descendant relationships. For the XMark 50% data set, we use 18 queries from X6-1 to X16-3 as shown in Table 4, and the meanings of x and y of the Xx-y are same as those of Mx-y. Figs. 16 and 17 show the results of query processing from M6-1 to M16-3 in Table 3 over the Mondial data set and from X6-1 to X16-3 in Table 4 over the XMark 50% data set, respectively. The performance of Extended is far better than EvalDP. It is similar to the case of Figs. 14 and 15 that the query processing time of EvalDP increases remarkably as the number of combinations in query results increases. In case the size of the result is one, the performance of Extended is similar to that of EvalDP.

6.2. Query processing with the p-index

In the previous researches about probabilistic XML data and query, the condition for the probability value of each node has not been considered. In this paper, we propose the condition for the probability value and a lightweight index to deal with the probability conditions. In this subsection, we analyze the performance of queries with the condition for the probability value and the efficiency of the p-index.

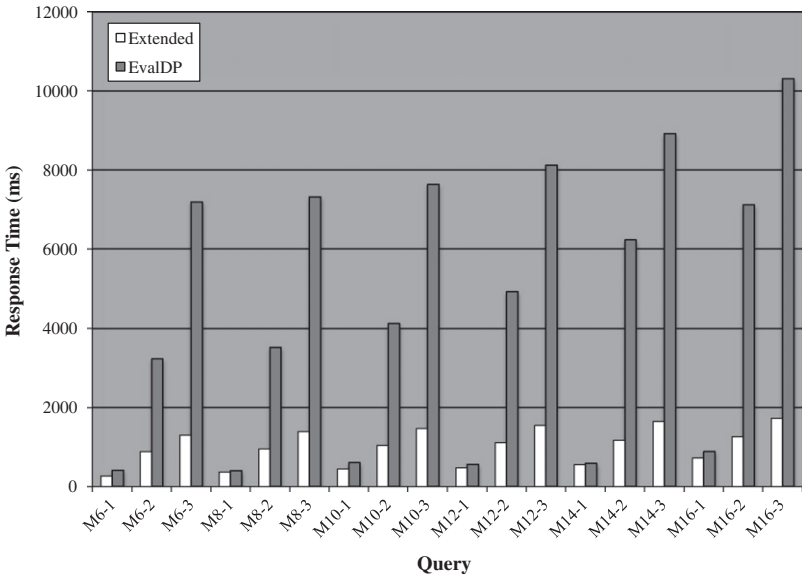

Fig. 16. Query processing for Mondial (2).

We use some simple path queries and query trees under the XMark 100% data set. Table 5 shows the information about experimental queries. Query P1 and P2 are simple path queries and Query P3 is a query tree. Queries from P11 to P13 derived from P1 have probability conditions for query nodes. For example, P12 is the ancestor–descendant relationship between the person and the phone in which the phone is restricted to the ones that have 0.3 or more and 0.6 or less conditional probability. The Results shows the number of combinations in query results.

Figs. 18 and 19 show the results of query processing with the p-index against without the p-index for XMark 100% data set. The performance of the p-index is far better than no index. Especially, as the number of data nodes restricted by the probability conditions increases, the time for query processing decreases.

If there is a constraint for the value of conditional or existence probability of a node, our proposed index, the p-index, can avoid accesses to the disk blocks which store labels with the value of probability which does not satisfy the constraint. If the number of disk blocks is N and the labels satisfying the constraint are in m ($<N$) disk blocks, then it is possible to access only m blocks for getting the list of labels that satisfy the constraint. However, if $m = N$ which is the worst case, then we should

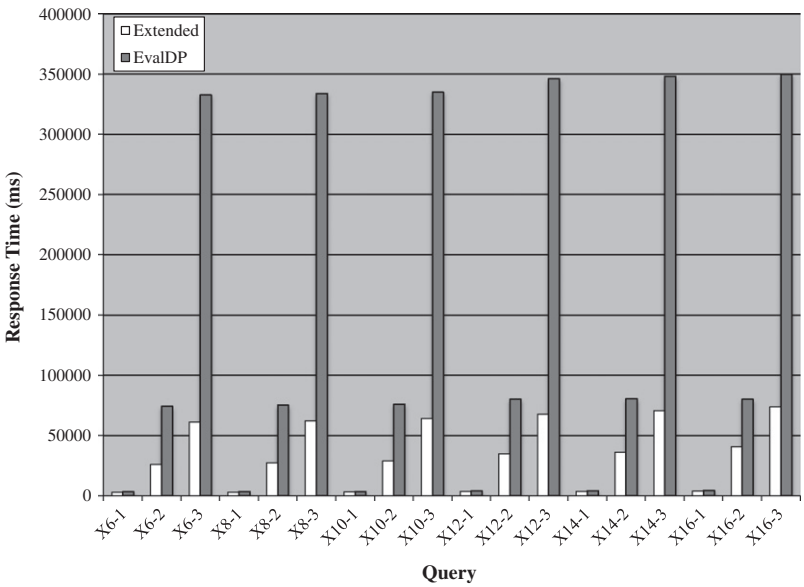

Fig. 17. Query processing for XMark 50% (2).

Table 5
Query set for the comparison of p-index and no p-index.

Query	Query expression	Results
P1	person//phone	12,679
P11	person//phone ($p_c = 0.5$)	574
P12	person//phone ($0.3 \leqslant p_c \leqslant 0.6$)	4167
P13	person ($0.4 \leqslant p_c \leqslant 0.7$)//phone ($0.3 \leqslant p_c \leqslant 0.6$)	620
P2	category//listitem	1267
P21	category//listitem ($p_c = 0.3$)	81
P22	category//listitem ($0.2 \leqslant p_c \leqslant 0.9$)	729
P23	category ($0.4 \leqslant p_c \leqslant 0.7$)//listitem ($0.2 \leqslant p_c \leqslant 0.9$)	245
P3	site//person[.//watch]//phone	12,679
P31	site//person ($0.4 \leqslant p_c \leqslant 0.6$)[.//watch]//phone	2,078
P32	site//person ($0.4 \leqslant p_c \leqslant 0.6$)[.//watch ($0.2 \leqslant p_c \leqslant 0.6$)]//phone	798
P33	site//person ($0.4 \leqslant p_c \leqslant 0.6$)[.//watch ($0.2 \leqslant p_c \leqslant 0.6$)]//phone ($p_c = 0.5$)	36

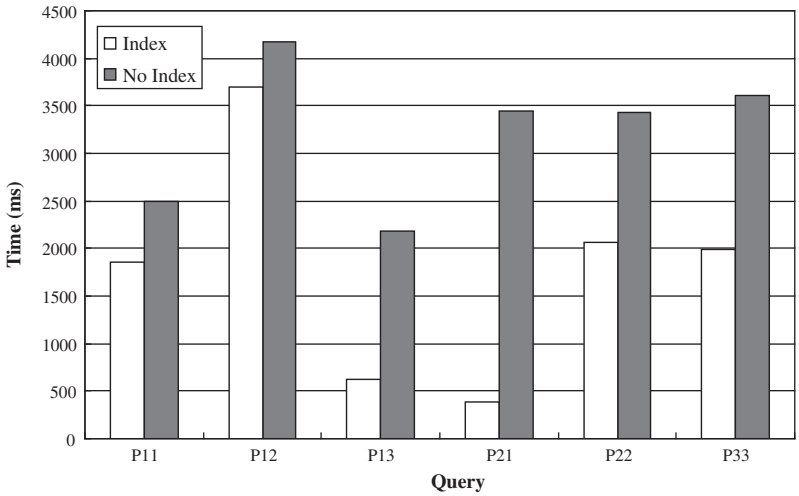

Fig. 18. Query processing with p-index (1).

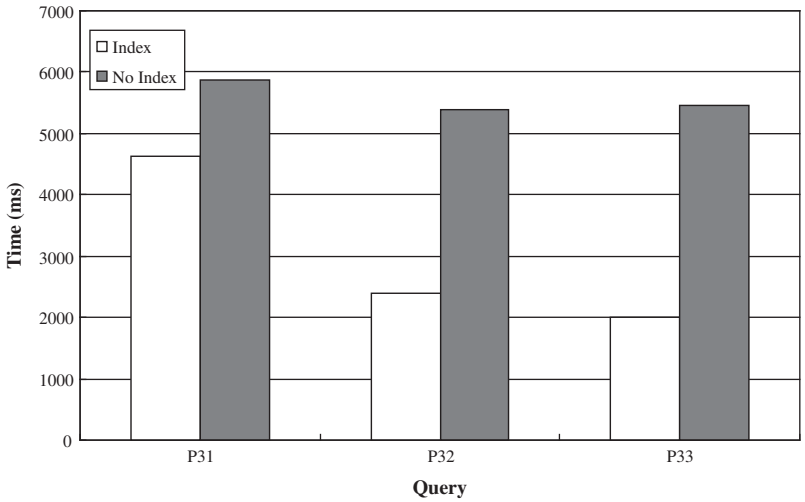

Fig. 19. Query processing with p-index (2).

access all the disk blocks. For example, when there is a query node such as title (pc > 0.7) and the list of labels for title elements are stored in *N* disk blocks, we firstly access the p-index for conditional probabilities of title elements and then we follow four linked lists from the start pointers of [0.7, 0.8), [0.8, 0.9), [0.9, 1.0) and [1.0]. At this time, we do not know whether we should access all disk blocks or some parts of the disk blocks. In the worst case, we should access *N* disk blocks. The number of disk blocks which should be accessed is dependent on the data. But in many cases, the p-index can decrease the number of disk blocks which should be accessed, and also make the size of input list to be decreased in the time and space complexities of our query processing. The size of the result set is one of the factors of time and space complexities, but it is not affected by using the p-index.

7. Conclusions

In this paper, we propose an extended interval-based labeling scheme to provide efficient probabilistic XML query processing. The extended interval-based labeling scheme not only takes the advantages of the previous interval-based labeling scheme but also solves weak points of previously proposed probabilistic XML query processing methods. It accesses only the labels of data specified in queries and finds data matches simultaneously with evaluating the probability of results in order to eliminate unnecessary data matches.

Also, we present an extended probabilistic XML query model with the predicates for the values of probabilities and a lightweight index for those probabilities. The index helps not to access the data which should be eliminated by the probability condition. Therefore the index reduces the time of data access and improves the performance of probabilistic XML query processing.

Experimental results show that our approach is significantly better than EvalDP, which is the most recent approach for probabilistic XML query processing, and improves the performance of query processing when the predicates for the values of probabilities are given.

Acknowledgements

We would like to thank the editor and anonymous reviewers. This work was supported in part by WCU (World Class University) program under the National Research Foundation of Korea funded by the Ministry of Education, Science and Technology of Korea (No. R31-30007), and in part by the National Research Foundation of Korea grant funded by the Korea government (MEST) (No. 2011-0000377).

References

Abiteboul, S., & Senellart, P. (2006). Querying and updating probabilistic information in XML. In *Proceedings of the international conference on extending database technology (EDBT)* (pp. 1059–1068).

Abiteboul, S., Kimelfeld, B., Sagiv, Y., & Senellart, P. (2009). On the expressiveness of probabilistic XML models. *The VLDB Journal, 18*, 1041–1064.

Bruno, N., Koudas, N., & Srivastava, D. (2002). Holistic twig joins: Optimal XML pattern matching. In *Proceedings of the ACM SIGMOD international conference on management of data* (pp. 310–321).

Chen, Y., Davidson, S., & Zheng, Y. (2004). BLAS: An efficient XPath processing system. In *Proceedings of the ACM SIGMOD international conference on management of data* (pp. 47–58).

Chien, S., Vagena, Z., Zhang, D., Tsotras, V. J., & Zaniolo, C. (2002). Efficient structural joins on indexed XML documents. In *Proceedings of 28th international conference on very large data bases (VLDB)* (pp. 263–274).

Cohen, S., Kimelfeld, B., & Sagiv, Y. (2009). Incorporating constraints in probabilistic XML. *ACM Transactions on Database Systems, 34*, 18:1–18:45.

Deay, D., & Sarkar, S. (1996). A probabilistic relational model and algebra. *ACM Transactions on Database Systems (TODS), 21*, 339–369.

Eiter, T., Lukasiewicz, T., & Walter, M. (2000). Extension of the relational algebra to probabilistic complex values. In *Proceedings of the first international symposium on foundations of information and knowledge systems* (pp. 94–115).

Fuhr, N., & Rölleke, T. (1997). A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems (TOIS), 15*, 32–66.

Hung, E., Getoor, L., & Subrahmanian, V. S. (2003a). Probabilistic interval XML. In *Proceedings of the international conference on database theory (ICDT)* (pp. 358–374).

Hung, E., Getoor, L., & Subrahmanian, V. S. (2003b). PXML: A probabilistic semistructured data model and algebra. In *Proceedings of IEEE international conference on data engineering (ICDE)* (pp. 467).

Kimelfeld, B., & Sagiv, Y. (2007). Matching twigs in probabilistic XML. In *Proceedings of 28th international conference on very large data bases (VLDB)* (pp. 27–38).

Kimelfeld, B., Kosharovsky, Y., & Sagiv, Y. (2008). Query efficiency in probabilistic XML models. In *Proceedings of the ACM SIGMOD international conference on management of data* (pp. 701–714).

Kimelfeld, B., Kosharovsky, Y., & Sagiv, Y. (2009). Query evaluation over probabilistic XML. *The VLDB Journal, 18*, 1117–1140.

Lakshmanan, L. V. S., Leone, N., Ross, R., & Subrahmanian, V. S. (1997). ProbView: A flexible probabilistic database system. *ACM Transactions on Database Systems (TODS), 22*, 419–469.

Li, Q., & Moon, B. (2001). Indexing and querying XML data for regular path expressions. In *Proceedings of 28th international conference on very large data bases (VLDB)* (pp. 361–370).

Li, T., Shao, Q., & Chen, Y. (2006). PEPX: A query-friendly probabilistic XML database. In *Proceedings of the 15th ACM conference on information and knowledge management (CIKM)* (pp. 848–849).

Mondial (1999). <http://www.dbis.informatik.uni-goettingen.de/Mondial/>.

Nierman, A., & Jagadish, H. V. (2002). ProTDB: Probabilistic data in XML. In *Proceedings of 28th international conference on very large data bases (VLDB)* (pp. 646–657).

Srivastava, D., Al-Khalifa, S., Jagadish, H. V., Koudas, N., Patel, J. M., & Wu, Y. (2002). Structural joins: A primitive for efficient XML query pattern matching. In *Proceedings of IEEE international conference on data engineering (ICDE)* (pp. 141–152).

Tatarinov, I., Viglas, S. D., Beyer, K., Shanmugasundaram, J., Shekita, E., & Zhang, C. (2002). Storing and querying ordered XML using a relational database system. In *Proceedings of the ACM SIGMOD international conference on management of data* (pp. 204–215).

Wu, X., Lee, M. L., & Hsu, W. (2004). A prime number labeling scheme for dynamic ordered XML trees. In *Proceedings of IEEE international conference on data engineering (ICDE)* (pp. 66–78).

XMark (2001). <http://monetdb.cwi.nl/xml/index.html>.

Zhang, C., Naughton, J., Dewitt, D., Luo, Q., & Lohman, G. (2001). On supporting containment queries in relational database management systems. In *Proceedings of the ACM SIGMOD international conference on management of data* (pp. 425–436).