Swinburne Research Bank
http://researchbank.swinburne.edu.au

Author:	Li, Jianxin; Liu, Chengfei; Zhou, Rui; Yu, Jeff X.
Title:	Quasi-SLCA based keyword query processing over probabilistic XML data
Year:	2014
Journal:	IEEE Transactions on Knowledge and Data Engineering
Volume:	26
Issue:	4
Pages:	957-969
URL:	http://hdl.handle.net/1959.3/378709

Copyright:

© 2013 IEEE. The accepted manuscript is reproduced in accordance with the copy right policy of the publisher. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copy righted component of this work in other works.

This is the author's version of the work, posted here with the permission of the publisher for your personal use. No further distribution is permitted. You may also be able to access the published version from your library.

The definitive version is available at: http://doi.org/10.1109/TKDE.2013.67

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Quasi-SLCA based Keyword Query Processing over Probabilistic XML Data

Jianxin Li, Chengfei Liu *Member, IEEE*, Rui Zhou and Jeffrey Xu Yu *Member, IEEE*

Abstract—The probabilistic threshold query is one of the most common queries in uncertain databases, where a result satisfying the query must be also with probability meeting the threshold requirement. In this paper, we investigate probabilistic threshold keyword queries (PrTKQ) over XML data, which is not studied before. We first introduce the notion of quasi-SLCA and use it to represent results for a PrTKQ with the consideration of possible world semantics. Then we design a probabilistic inverted (PI) index that can be used to quickly return the qualified answers and filter out the unqualified ones based on our proposed lower/upper bounds. After that, we propose two efficient and comparable algorithms: Baseline Algorithm and PI index-based Algorithm. To accelerate the performance of algorithms, we also utilize probability density function. An empirical study using real and synthetic data sets has verified the effectiveness and the efficiency of our approaches.

Index Terms—Probabilistic XML, Threshold Keyword Query, Probabilistic Index.

1 Introduction

Uncertainty is widespread in many web applications, such as information extraction, information integration, web data mining, etc. In uncertain database, probabilistic threshold queries have been studied extensively where all results satisfying the queries with probabilities equal to or larger than the given threshold values are returned [1], [2], [3], [4], [5]. However, all of these works were studied based on uncertain relational data model. Because the flexibility of XML data model allows a natural representation of uncertain data, uncertain XML data management has become an important issue and lots of works have been done recently. For example, many probabilistic XML data models were designed and analyzed [6], [7], [8], [9], [10]. Based on different data models, query evaluation [7], [10], [11], [12], [13], algebraic manipulation [8] and updates [6], [10] were studied. However, most of these works concentrated on structured query processing, e.g., twig queries. In this paper, we propose and address a new interesting and challenging problem of *Pr*obabilistic *T*hreshold *K*eyword *Q*uery (PrTKQ) over uncertain XML databases, which is not studied before as far as we know.

In general, an XML document could be viewed as a rooted tree, where each node represents an element or contents. XIRQL [14] supports keyword search in XML based on structured queries. If users prefer keyword queries, they can directly type in a virtual root node (i.e., //*[.ftcontains()]) with query keywords. In this case, XIRQL can be considered equivalent to XML keyword query. The keyword search results consist of a set of XML elements that contain at least one occurrence of all the query keywords, after excluding the occurrences of the keywords in sub-elements that already contain all the query keywords. The set of XML elements can be sorted based on their relevance to the given keyword query. This keyword search result semantics has been well studied in [15], [16], denoted as *Lowest Common Ancestor* (LCA) semantics. However, one intuitive observation is that if a sub-element already contains all of the query keywords, it (or one of its descendants) will be a more specific result for the query, and thus should be returned in lieu of the parent element. As such, Xu and Papakonstantinou[17] proposed the concept of *Smallest Lowest Common Ancestor* (SLCA), where a node v is regarded as an SLCA if (a) the subtree rooted at the node v, denoted as $T_{sub}(v)$, contains all the keywords, and (b) there does not exist a descendant node v' of v such that $T_{sub}(v')$ contains all the keywords. In other words, if a node is an SLCA, then its ancestors will be definitely excluded from being SLCAs. In this paper, we consider the SLCA semantics to study the problem of *Pr*obabilistic *T*hreshold *K*eyword *Q*uery (PrTKQ).

Based on the SLCA semantics, [18] discussed top-k keyword search over a probabilistic XML document. Given a keyword query q and a probabilistic XML document (PrXML), [18] returned the top k most relevant SLCA results (PrSLCAs) based on their probabilities. Different from the SLCA semantics over deterministic XML documents, a node v being a PrSLCA can only exclude its ancestors from being PrSLCAs

- *Jianxin Li, Chengfei Liu and Rui Zhou are with the Faculty of Information & Technology, Swinburne University of Technology, Australia. {jianxinli, cliu, rzhou}@swin.edu.au*
- *Jeffrey Xu Yu is with the Department of Systems Engineering & Engineering Management, The Chinese University of Hong Kong, China. yu@se.cuhk.edu.hk*

Digital Object Indentifier 10.1109/TKDE.2013.67 1041-4347/13/$31.00 © 2013 IEEE

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

by a probability. This probability can be calculated by aggregating the probabilities of the deterministic documents (called possible worlds) W implied in the PrXML where v is an SLCA in each deterministic document $\in W$.

However, it is not suitable to directly utilize the PrSLCA semantics for evaluating PrTKQs because the PrSLCA semantics are too strong. In some applications, users tend to be confident with the results to be searched, so relatively high probability threshold values may be given. Consequently, it is very likely that no qualified PrSLCA results will be returned. To solve this problem, we propose and utilize a so-called *quasi-SLCA* semantics to define the results of a PrTKQ by *relaxing* the semantics of PrSLCA with regards to a given threshold value, i.e., besides the probability of v being a PrSLCA in PrXML, the probability of a node v being a quasi-SLCA in PrXML may also count the probability of v's descendants being PrSLCAs in PrXML if their probabilities are below the specified threshold value. In other words, a node v being a quasi-SLCA will exclude its ancestors from being quasi-SLCAs by a probability only when this probability is no less than the given threshold; otherwise, this probability will be included for contributing to its ancestors.

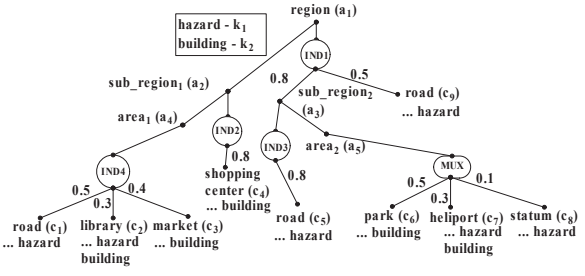


Fig. 1. A probabilistic XML data tree

Example 1: Consider an aircraft-monitored battlefield application, where the useful information will be taken as Aerial photographies. Through analysing the photographies, we can extract the possible objects (e.g., road, factory, airport, etc.) and attach some text description to them with probabilities, which can be stored in the format of PrXML. Figure 1 is a snapshot of an aircraft-monitored battlefield XML data. By issuing a keyword query $\{hazard, building\}$, a military department would find the potential areas containing hazard buildings above a probability threshold.

Based on the semantics of PrSLCA, it is obvious to see that the probability of the node $library$ being a PrSLCA is 0.3. For the node $area_1$, its probability can be calculated by 0.5*0.4*(1-0.3) = 0.14 because $area_1$ can become a PrSLCA only if the nodes $road$ and $market$ exist while the node $library$ does not exist. Similarly, we can get the probabilities of the other nodes being PrSLCAs, e.g., sub_region_1(= 0.168), $heliport$(= 0.24), sub_region_2(= 0.32) and $region$(= 0.088). As we know, the users generally specify a threshold value σ as the confidence score with their issued query, e.g., $\sigma = 0.40$ representing that the users prefer to see the answers with their probabilities up to 0.40. In this condition, no results can be returned.

However, from Figure 1, we can see that if the probabilities of $library$ and $area_2$ could contribute to their parent nodes, $area_1$ and sub_region_2 would become quasi-SLCA results. Unfortunately, the PrSLCA semantics exclude them from being results. This motivates us to relax the PrSLCA semantics to the quasi-SLCA semantics. According to the quasi-SLCA semantics, the probabilities of $area_1$ and sub_region_2 being the quasi-SLCA results are 0.44 and 0.56 with the contributions of their child nodes $library$ and $area_2$, respectively. As such, $area_1$ and sub_region_2 are deemed as the interesting places to be returned.

The contributions of this paper are summarized:

- Based on our proposed quasi-SLCA result definition, we study probabilistic threshold keyword query over uncertain XML data, which satisfies the possible world semantics. To the best of our knowledge, this problem has not been studied before.
- We design a probabilistic inverted (PI) index that can quickly compute the lower bound and upper bound for a threshold keyword query, by which lots of unqualified nodes can be pruned and qualified nodes can be returned as early as possible. To keep the effectiveness of pruning, the probability density function is employed based on the assumption of Gaussian distribution.
- We propose two algorithms, a comparable baseline algorithm and a PI-based Algorithm, to efficiently find all the quasi-SLCA results meeting the threshold requirement.
- Experimental evaluation has demonstrated the efficiency and effectiveness of the proposed approaches.

The rest of this paper is organized as follows. In Section 2, we introduce the probabilistic XML model and the problem definition. Section 3 provides the overview of the proposed approachs. Section 4 presents the procedure of building PI index. In Section 5, we propose a comparable baseline algorithm and a PI-based algorithm to find the qualified quasi-SLCA results. We report the experimental results in Section 6. Section 7 discusses related works and Section 8 concludes the paper.

2 Probabilistic Data Model and Problem Definition

Probabilistic Data Model: A PrXML document defines a probability distribution over a space of deterministic XML documents. Each deterministic document belonging to this space is called a possible world. A PrXML document represented as a labeled

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

tree has *ordinary* and *distributional* nodes. Ordinary nodes are regular XML nodes and they may appear in deterministic documents, while distributional nodes are only used for defining the probabilistic process of generating deterministic documents and they do not occur in those documents.

In this paper, we adopt a popular probabilistic XML model, PrXML$^{\{ind,mux\}}$ [12], [18], which was first discussed in [7]. In this model, a PrXML document is considered as a labeled tree where distributional nodes have two types, IND and MUX. An IND node has children that are *independent* of each other, while the children of a MUX node are *mutually-exclusive*, that is, at most one child can exist in a random instance document (called a *possible world*). A real number from (0,1] is attached on each edge in the XML tree, indicating the conditional probability that the child node will appear under the parent node given the existence of the parent node. An example of a PrXML document is given in Fig. 1. Unweighted edges have 1 as the default conditional probability.

The Semantics of PrSLCA in PrXML: According to the semantics of possible worlds, the global probability of a node v being a PrSLCA with regard to a given query q in the possible worlds is defined as follows:

$$Pr^G_{slca}(q,v) = \sum_{i=1}^{m}\{Pr(w_i)|slca(q,v,w_i) = true\} \quad (1)$$

where $w_1, \ldots, w_m$ denote the possible worlds implied in the PrXML; $slca(q,v,w_i) = true$ indicates that v is an SLCA in the possible world w_i for the query q; $Pr(w_i)$ is the existence probability of the possible world w_i; The symbol G means $Pr^G_{slca}(q,v)$ is the *global* probability of a node v being an SLCA w.r.t. q in all possible worlds.

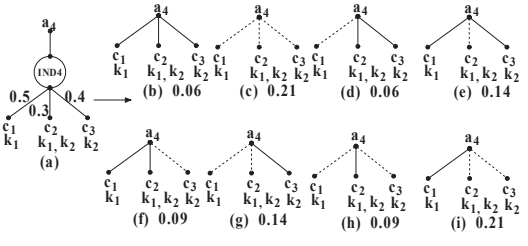


Fig. 2. A small PrXML and its possible worlds

Example 2: Consider a small PrXML in Figure 2.a and all generated possible worlds in Figure 2.{b,c,d,e,f,g,h,i} where the solid line represents the existence of the edge while the dashed line represents the absence of the edge. Given a possible world, we can compute its global probability based on the existence/absence of the edges in the possible world, e.g., $Pr(w_d) = (1-0.5)*0.3*0.4 = 0.06$.

The Semantics of quasi-SLCA in PrXML:

Definition 1: **Quasi-SLCA:** Given a keyword query q and a threshold value σ, a node v is called a quasi-SLCA if and only if (1) v or its descendants are SLCAs in a set W of possible worlds; (2) the aggregated probability of v and its descendants to be SLCAs in W is no less than σ; (3) no descendant nodes of v satisfy both of the above conditions in any set of possible worlds that overlaps with W.

In other words, if a descendant node v_d of v is a quasi-SLCA, then the probability of v_d has to be excluded from the probability of v being a quasi-SLCA. It means that the set of possible worlds that v_d appears does not overlap with the set of possible worlds that v or its other descendants appear.

Given a query q, we can compute $Pr^L_{slca}(q,v)$ in a bottom-up manner, where $Pr^L_{slca}(q,v)$ stands for the local probability for v being an SLCA in the probabilistic subtree rooted at v. For example, a_4 in Figure 2(a) is a subtree of Figure 1. $Pr^L_{slca}(q,a_4)$ can be used to compute the PrSLCA probability of a_2 and a_1. From $Pr^L_{slca}(q,v)$, we can easily get $Pr^G_{slca}(q,v)$ by $Pr^G_{slca}(q,v) = Pr(path_{r\to v}) \times Pr^L_{slca}(q,v)$ where $Pr(path_{r\to v})$ indicates the existence probability of v in the possible worlds. It can be computed by multiplying the conditional probabilities along the path from the root r to v.

Now, we define quasi-SLCA based on PrSLCA and the parent-child relationship. For an IND node v, we have:

$$Pr^G_{quasi-slca}(q,v) = Pr^G_{slca}(q,v) + Pr(path_{r\to v})\times (1 - \prod_{v'\in child(v)\wedge Pr^G_{quasi-slca}(q,v')<\sigma}(1 - Pr^L_{slca}(q,v'))) \quad (2)$$

where the child node v' of v is an SLCA node, but not a quasi-SLCA node. σ is the given threshold value.

For MUX node v, we have:

$$Pr^G_{quasi-slca}(q,v) = Pr^G_{slca}(q,v) + Pr(path_{r\to v})\times \sum\{Pr^L_{slca}(q,v')|v' \in child(v) \wedge Pr^G_{quasi-slca}(q,v') < \sigma\} \quad (3)$$

Note, IND or MUX nodes are normally not allowed to be SLCA result nodes because they are only distributional nodes. As such, for the above IND or MUX node v, we may use its parent node v_p (with v as a sole child) to represent the SLCA result node.

Example 3: Let's consider Example 2 again. First assume the specified threshold value is 0.40, then the global probability of a_4 being a quasi-SLCA result can be calculated by using $Pr^G_{quasi-slca}(q,a_4)$ = $Pr^G_{slca}(q,a_4)$ + $Pr(path_{r\to a_4})$ * $(1-(1-Pr^L_{slca}(q,c_2)))$ = 0.14 + 0.30 = 0.44 because child c_2 is an SLCA node but not a quasi-SLCA node w.r.t. the given threshold. So c_2's SLCA probability contributes to its parent node a_4. If the threshold is decreased to 0.30, then c_2 will be taken as a qualified quasi-SLCA result and will not contribute to a_4. In this case, a_4 cannot become a quasi-SLCA result because $Pr^G_{quasi-slca}(q,a_4)$ = $Pr^G_{slca}(q,a_4)$ = 0.14 $<$ 0.30. If the threshold is further decreased to 0.14, both c_2 and a_4 are qualified quasi-SLCA results.

Definition 2: **Probabilistic Threshold Keyword Query:** (PrTKQ) Given a keyword query q and a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

threshold σ, the results of q over a probabilistic XML data T is a set R of quasi-SLCA nodes with their probabilities equal to or larger than σ, i.e., $Pr^G_{quasi-slca}(q, v) \geq \sigma$ for $\forall v \in R$.

In this work, we are interested in how to efficiently compute the quasi-SLCA answer set for a PrTKQ over a probabilistic XML data.

3 Overview of this Work

A naive method to answer a PrTKQ is to enumerate all possible worlds and apply the query to each possible world. Then, we can compute the overall probability of each quasi-SLCA result and return the results meeting the probability threshold. However, the naive method is inefficient due to the huge number of possible worlds over a probabilistic XML data. Another method is to extend the work in [18] to compute the probabilities of quasi-SLCA candidates. Although it is much more efficient than the naive method, it needs to scan the keyword node lists and calculate the keyword distributions for all relevant nodes. Therefore, that motivates our development of efficient algorithms which not only avoids generating possible worlds, but also prunes more unqualified nodes.

To accelerate query evaluation, in this paper we propose a prune-based probabilistic threshold keyword query algorithm, which determines the qualified results and filters the unqualified candidates by using off-line computed probability information. To do this, we need to first calculate the probability of each possible query term within a node, which is stored as an off-line computed probabilistic index. Within a node, any two of its contained terms may appear in the IND or MUX ways. To precisely differentiate IND and MUX, we utilize different parts to represent the probabilities of possible query terms appearing in MUX way, while the terms in each part hold IND relationships. In other words, the different parts of terms in a node are mutual-exclusive (MUX), e.g., a_1 and a_5 in Figure 3 consists of three parts.

Given a keyword query and a threshold value, we first load the corresponding off-line computed probabilistic index w.r.t. the keyword query and then on-the-fly calculate the range of probabilities of a node being a result of the keyword query using the pre-computed probabilistic index in a bottom-up strategy. Here, the range of probabilities can be represented by two boundary values: lower bound and upper bound. By comparing the lower/upper bounds of candidates, the qualified results can be efficiently identified.

The followed two examples briefly demonstrate how we calculate the lower/upper bounds based on a given keyword query and the off-line computed probabilistic index, and how we apply the on-line computed lower/upper bounds to prune the unqualified candidates and determine the qualified ones.

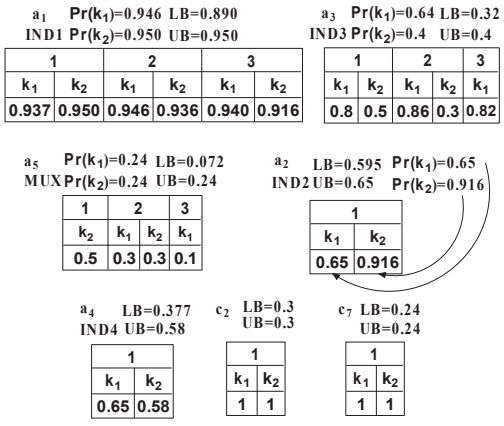


Fig. 3. PI index and Lower/Upper Bound for a query $\{k_1, k_2\}$ over the given PrXML

Figure 3 shows the lower/upper bounds of each node in Figure 1 where the probability of each individual term is calculated offline while the lower/upper bounds are computed on-the-fly based on the given query keywords. Let's first introduce the related concepts briefly: the probability of a term in a node represents the total local probability of the term appearing in all possible worlds to be generated for the probabilistic subtree rooted at the node, e.g., $Pr(k_1, a_2)$ = 0.65 and $Pr(k_2, a_2)$ = 0.916; the lower bound value represents the minimal total local probability of the given query keywords appearing in all the possible worlds w.r.t. the probabilistic subtree, e.g., LB(k_1k_2, a_2)=0.65*0.916=0.595; the upper bound value represents the maximal total local probability of the given query keywords appearing in all the possible worlds w.r.t. the probabilistic subtree because the keywords may be independent or co-occur, e.g., UB(k_1k_2, a_2) = min{0.65, 0.916} = 0.65 no matter whether they are independent. By multiplying the path probability, the local probability can be transformed into the global probability. For the nodes containing MUX semantics, we group the probabilities of its terms into different parts, any two of which are mutually-exclusive as shown in a_1, a_3 and a_5 in Figure 3. The details of computing the lower/upper bounds for the IND and MUX semantics in the following section.

Example 4: Consider a PrTKQ $\{k_1, k_2\}$ with σ=0.40 again. a_5, c_2 and c_7 can be pruned directly without calculation because their upper bounds are all lower than 0.40. We need to check the rest nodes a_1, a_2, a_3 and a_4. For a_4, after computation, the probability of a_4 being a quasi-SLCA result is 0.44, which is larger than the specified threshold value 0.40, so a_4 will be taken as a result. After that, the result of a_4 can be used to update the lower bound and upper bound of a_2, (LB=0.595, UB=0.65) $\rightarrow$ (LB=0.155, UB=0.21). As a consequence, a_2 should be filtered due to $UB(a_2) = 0.21 < \sigma = 0.40$. Similarly, a_3 can be computed and selected as a result because its

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

probability is 0.56. Since a_3 and a_4 having been the quasi-SLCA results, the bounds of a_1 can be updated as (LB=0.890, UB=0.950) $\rightarrow$ (LB=0.136, UB=0.196). As such, a_1 can be pruned because its upper bound is lower than 0.40. From this example, we can find that many answers can be pruned or returned without the need to know their accurate probabilities, and the effectiveness of pruning would be accelerated greatly with the increase of users' search confidence.

As an acute reader, you may find that we have to compute the probability of a_4 being a quasi-SLCA because it cannot determine whether or not a_4 is a qualified result to be output only based on its lower/upper bound values. To exactly calculate the probability of a_4 being a quasi-SLCA, we have to access its child/descendant nodes, e.g., c_1, c_2, c_3, although c_2 has been recognized as a pruned node before we start to process a_4. If an internal node depends on a larger number of pruned nodes, the effectiveness of pruning will be degraded to some extent. To fix this challenging problem, we will introduce *Probability Density Function* PDF that can be used to approximately compute the probability of a node, the result of which can be used to update the lower bound and upper bound of its ancestor nodes further. The details are provided and discussed with algorithms later.

4 Probabilistic Inverted Index

In this section, we describe our *P*robabilistic *I*nverted (PI) index structure for efficiently evaluating PrTKQ queries over probabilistic XML data. In keyword search on certain XML data, inverted indexes are popular structures, e.g., [15], [17]. The basic technique is to maintain a list of lists, where each element in the outer list corresponds to a domain element (i.e., a keyword). Each inner list stores the ids of XML nodes in which the given keyword occurs, and for each node, the frequencies or the weight at which the keyword appears or takes. In this work, we introduce a probabilistic version of this structure, in which we store for each keyword a list of node-ids. Along with each node-id, we store the *probability values* that the subtree rooted at the node may contain the given keyword. The probability values in inner lists can be used to compute lower bound and upper bound on-the-fly during PrTKQ evaluation.

Figure 4 shows an example of a probabilistic inverted index of the data in Figure 1. At the base of the structure is a list of keywords storing pointers to lists, corresponding to each term in the XML data T. This is an inverted array storing, for each term in T, a pointer to a list of triple tuples. In the list $k_i.list$ corresponding $k_i \in T$, the triple $\boxed{(v\_id, \text{Pr}(\text{path}_{r\rightarrow v}), \{p_1, ...\})}$ records the node v [1], the conditional probability from the root to v, and the probability set that may contain *single probability value* or *multiple probability value*. Single probability value represents that all the keyword instances in the subtree can be considered as independent in probability, e.g., the confidence of a_2 containing k_1 is $\{0.65\}$, while multiple probability value means that the keyword instances belonging to different sets occur mutually, e.g., the confidence of a_3 containing k_1 is a set $\{0.8, 0.86, 0.82\}$, that represents the different possibilities of k_1 occurring in a_3.

1. The symbol v is used to represent a node's name or a node's id without confusions in the following sections. Here, v is the id of the node v

4.1 Basic Operations of Building PI Index

To build PI index, we need to traverse the given XML data tree once in a bottom-up method. During the data traversal, we will apply the following operations that may be used solely or in their combinations. The binary operation X $\rhd\lhd^{/}$ Y promotes the probability of Y to its parent node X. The binary operation X $\rhd\lhd^{sibling}$ Y promotes the probabilities of two sibling nodes X and Y to their parent node. The n-ary case can be processed by calling for the corresponding binary cases one by one.

Assume v_1 contains the keywords $\{k_1, k_2, ..., k_i, ..., k_m\}$ and the conditional probability $Pr(path_{v_p->v_1})$ is λ_1; and v_2 contains the keywords $\{k_i, k_{i+1}, ..., k_{m_i}\}$ and the conditional probability $Pr(path_{v_p->v_2})$ is λ_2.

Operator1-$\mathbf{v}_1 \bowtie^{sibling,IND} \mathbf{v}_2$: If v_1 and v_2 are independent sibling nodes, we can directly promote their probabilities to their parent v_p, then we have,

$$Pr(k_j, v_p) = \begin{cases} \lambda_1 * Pr(k_j, v_1) & j < i; \\ 1 - (1 - \lambda_1 * Pr(k_j, v_1)) & \\ (1 - \lambda_2 * Pr(k_j, v_2)) & i \le j \le m \le m_i; \\ \lambda_2 * Pr(k_j, v_2) & m \le j \le m_i; \end{cases} \quad (4)$$

Operator2-$\mathbf{v}_1 \bowtie^{/,IND} \mathbf{v}_2$: If v_2 is an independent child of v_1, we can directly promote the probability of v_2 to v_1, then we have,

$$Pr(k_j, v_1) = \begin{cases} Pr(k_j, v_1) & j < i; \\ 1 - (1 - Pr(k_j, v_1)) & \\ (1 - \lambda_2 * Pr(k_j, v_2)) & i \le j \le m \le m_i; \\ \lambda_2 * Pr(k_j, v_2) & m \le j \le m_i; \end{cases} \quad (5)$$

Example 5: Let's show the procedure of computing $c_1 \bowtie^{sibling,IND} c_2 \bowtie^{sibling,IND} c_3$ in Figure 2 using Operator1 and Operator2. Firstly, we compute $c_1 \bowtie^{sibling,IND} c_2$ and promote the probability of keywords to their parent a_4 by Operator1, i.e., $Pr(k_1, a_4)$ = 1 - (1 - 0.5*1.0)*(1 - 0.3*1.0) = 0.65 and $Pr(k_2, a_4)$ = 0.3. And then, we compute $a_4 \bowtie^{/,IND} c_3$ using operator2, i.e., $Pr(k_2, a_4)$ = 1 - (1 - 0.3)*(1 - 0.4) = 0.58 while $Pr(k_1, a_4)$ do not change because c_3 only contains k_2 here. And the conditional probability from the root to a_4 is 1.0. Therefore, $k_1 \rightarrow (a_4, 1.0, 0.65)$ and $k_2 \rightarrow (a_4, 1.0, 0.58)$ will be inserted in PI index, respectively.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

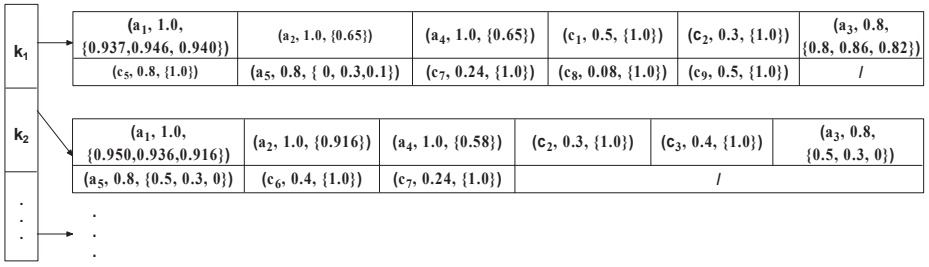


Fig. 4. A probabilistic Inverted Index

Operator3-v$_1$ $\bowtie^{sibling,MUX}$ **v**$_2$: If v_1 and v_2 are two mutually-exclusive sibling nodes and v_p is their parent, then we generate two parts in v_p by $v_p \bowtie^{/,IND} v_1$ and $v_p \bowtie^{/,IND} v_2$, respectively.

Operator4-v$_1$ $\bowtie^{/,MUX}$ **v**$_2$: If v_2 is a mutually-exclusive child node of v_1, then we can get the aggregated probability by $v_1 \bowtie^{/,IND} v_2$.

In the above four basic operators, we assume the terms independently appear in v_1 and v_2. When the nodes v_1 and v_2 contain mutually-exclusive parts, we need to deal with each part using the four basic operators.

Given two independent sibling nodes v_1 (λ_1) and v_2 (λ_2) where only v_2 contains a set of mutually-exclusive parts $\{p_{m_1}, p_{m_2}, ...\}$ with conditional probability λ_{m_i}. In this case, we can apply the operation $v_1 \bowtie^{sibling,IND} p_{m_i}$ for each part p_{m_i}. The computed results are maintained in different parts in their parent v_p.

Example 6: Consider an independent node c_5 and a node a_5 consisting of c_6, c_7 and c_8 in Figure 1. We first promote c_6, c_7 and c_8 to a_5 that consists of three parts:

1	2		3
k_2	k_1	k_2	k_1
0.5	0.3	0.3	0.1

, as shown in Figure 3 - a_5. Because c_5 and a_5 are independent sibling nodes, the operation $c_5 \bowtie^{sibling,IND} a_5$ can be called to compute the probability with regards to their parent a_3. To do this, we apply $c_5 \bowtie^{sibling,IND}$ part$_i$ for each part$_i \in a_5$ using Operator1. The results are shown in Figure 3 - a_3. After that, we can insert $k_1 \rightarrow (a_3, 0.8, 0.8, 0.86, 0.82)$ and $k_2 \rightarrow (a_3, 0.8, 0.5, 0.3, 0)$ into PI index.

If both v_1 and v_2 contain a set of mutually-exclusive parts, respectively, then we can do pairwise aggregations across the two sets of parts. To build the PI index, we can use any numbering schemes as long as the basic document navigation operations are provided, such as Pre/Size/Level encoding (equivalent to the Pre/Post encoding) in [19] and Dewey encoding scheme in [20]. The probability of each node will be appended to its encoding as an additional value. The basic idea of building PI index is to progressively process the document nodes in ascending order, i.e., the data can be loaded and processed in a streaming strategy. When a leaf node v_l is coming, we will compute the probability of each term in the leaf node v_l. After that, the terms with their probabilities in v_l will be written into PI index. Next, we need promote the terms and their probabilities of v_l to the parent v_p of v_l based on the operation types in Section 4.1. After the node stream is scanned completely, the building algorithm of PI index will be terminated. We don't provide the detailed building algorithm in this paper.

4.2 Pruning Techniques using PI Index

In this subsection, we first show how to prune the unqualified nodes using the proposed lower/upper bounds. And then, we explain how to compute lower/upper bounds, and how to update the upper/lower bounds based on intermediate results during the query evaluation.

By default, the node lists in PI index are sorted in the document order. $Pr(k_i, v)$ represents the overall probability of a keyword k_i in a node v. It is obvious that the overall probability of a keyword appearing in a node is larger than or equal to that of the keyword appearing in its descendant nodes. And the overall probability value for each keyword in a node can be computed and stored in PI index.

Consider a node v and a PrTKQ q containing a set of keywords $\{k_1, k_2, ..., k_t\}$. If all the terms in v are independent, then we have,

$$LB(q, v) = \prod_{i=1}^{t} Pr(k_i, v) \tag{6}$$

$$UB(q, v) = min\{Pr(k_i, v) | 1 \leq i \leq t\} \tag{7}$$

Most of the time, v consists of a set of parts $\{vp_1, vp_2, ..., vp_m\}$ that are mutually-exclusive. In this case, the lower bound of v would be generated from a part vp_j that gives the highest lower bound value while the upper bound of v would be generated from another part vp_i that gives the highest upper bound value, in which j may be equal to or not equal to i.

$$LB(q, v) = max_{1 \leq j \leq m}\{\prod_{i=1}^{t} Pr(k_i, vp_j)\} \tag{8}$$

$$UB(q, v) = max_{1 \leq j \leq m}\{min\{Pr(k_i, vp_j) | 1 \leq i \leq t\}\} \tag{9}$$

Where vp_j must satisfy the criteria: (1) $LB(q, v) > 0$; (2) cannot find another part vp'_j having $\prod_{i=1}^{t} Pr(k_i, vp'_j) > 0$ and $min\{Pr(k_i, vp'_j) | 1 \leq$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

$i \leq t\} > min\{Pr(k_i, vp_j)|1 \leq i \leq t\}$. Otherwise, UB($q, v$) and LB($q, v$) will be set as zero.

Example 7: Let's consider a_3 in Figure 3 as an example. The first and second parts can generate lower and upper bounds: Part 1 $\rightarrow$ LB($\{k_1, k_2\}$,a_3)=0.32, UB($\{k_1, k_2\}$,a_3)=0.4; and Part 2 $\rightarrow$ LB($\{k_1, k_2\}$,a_3)=0.206, UB($\{k_1, k_2\}$,a_3)=0.24. Because Part 1 can produce a higher upper bound than Part 2, the lower and upper bounds of a_3 will come from Part 1, which guarantees that a_3 can be a quasi-SLCA candidate with a higher probability. Since Part 3 does not contain full keywords, i.e., missing k_2, it cannot generate lower and upper bounds.

Property 1: **[Upper Bound Usage]** A node v can be filtered if the overall probability $Pr(k_i, v)$ of any keyword k_i ($k_i \in v$ and $k_i \in q$) is lower than the given threshold value σ, i.e., $\exists k_i$, $Pr(k_i, v) < \sigma$.

Proof: Since $Pr^G_{quasi-slca}(q, v) \leq min\{Pr(k_i, v)|k_i \in q\} \leq Pr(\forall k_i \in q, v)$, we have $min\{Pr(k_i, v)|k_i \in q\}$ as the upper bound probability of v becoming a qualified quasi-SLCA node. Therefore, if a node v holds the inequation $Pr(k_i, v) < \sigma$, then $Pr^G_{quasi-slca}(q, v)$ must be lower than σ. As such, v can be filtered. □

Property 2: **[Lower Bound Usage]** The nodes v can be returned as required results if we have $LB(q, v) \geq \sigma$ and $UB(q, v_d) < \sigma$ where v_d is any child or descendant node of v.

Proof: $UB(q, v_d) < \sigma$ means that all the keyword nodes in the subtree rooted at v will contribute their probabilities to node v. In other words, no descendant node of v could be a quasi-SLCA so the lower bound probability $LB(q, v)$ will not be deducted. Therefore, if we have for $LB(q, v) \geq \sigma$, then $Pr^G_{quasi-slca}(q, v) \geq \sigma$. As such, v can be returned as a quasi-SLCA result. □

Example 8: Let's continue Example 7. a_3 can be directly returned as a qualified answer for the given threshold σ(= 0.4). This is because c_2, c_7 and a_5 are filtered due to their upper bound less than the threshold σ(= 0.4).

To update the lower/upper bound values during query evaluation, one way is to treat the different types of nodes differently, by which the updated lower/upper bounds may obtain better precision. But the disadvantage of this way is to easily affect the efficiency of bound update. This is because, given a current node having multiple quasi-SLCA nodes as its descendant nodes, it is required to know the detailed relationships (IND or MUX) among the multiple quasi-SLCA nodes. To avoid the disadvantage, we do not separate the different types of distributional nodes, under which the multiple quasi-SLCA nodes appear. In other words, we unify them into a uniform formula based on the following two properties.

Property 3: No matter node v is an IND or ordinary or MUX node, we can update their upper bound values as follows:

$$UB^{'}(q, v) = UB(q, v) - 1 + \prod_{i=1}^{m}(1 - Pr^G_{quasi-slca}(q, v_{c_i})) \quad (10)$$

Where $Pr^G_{quasi-slca}(q, v_{c_i}) \geq \sigma$ should be held.

Proof: According to the definition of upper bound, UB(q, v) represents the maximal probability of v being a quasi-SLCA node, which comes from the overall probability of a specific keyword. Therefore, the problem of updating upper bound can be alternatively considered as the percentage of the probability of the keyword has been used for the v' descendant nodes becoming qualified quasi-SLCA nodes. If we know there are m qualified descendant nodes of v as returned answers, then we can compute their aggregated probabilities by $1-\prod_{i=1}^{m}(1-Pr^G_{quasi-slca}(q, v_{c_i}))$. Therefore, the upper bound can be updated as $UB(q, v) - 1 + \prod_{i=1}^{m}(1 - Pr^G_{quasi-slca}(q, v_{c_i}))$.

Does the above update equation hold for MUX node? To answer this question, we utilize the properties in [18], from which we can compute the aggregated probability by using $\sum_{i=1}^{m} Pr^G_{quasi-slca}(q, v_{c_i})$. Therefore, we have $UB^{'}(q, v) = UB(q, v) - \sum_{i=1}^{m} Pr^G_{quasi-slca}(q, v_{c_i})$. The equation can be converted into $UB(q, v)-1+[1-\sum_{i=1}^{m} Pr^G_{quasi-slca}(q, v_{c_i})]$.

Since $\prod_{i=1}^{m}(1 - Pr^G_{quasi-slca}(q, v_{c_i}))$ can be expressed as $1 - \sum_{i=1}^{m} Pr^G_{quasi-slca}(q, v_{c_i}) + \Delta$ where Δ is a positive value, i.e., ≥ 0, we can derive that $1 - \sum_{i=1}^{m} Pr^G_{quasi-slca}(q, v_{c_i}) \leq \prod_{i=1}^{m}(1 - Pr^G_{quasi-slca}(q, v_{c_i}))$. As a consequence, we can obtain that $UB^{'}(q, v) = UB(q, v) - \sum_{i=1}^{m} Pr^G_{quasi-slca}(q, v_{c_i})$ = $UB(q, v) - 1 + [1 - \sum_{i=1}^{m} Pr^G_{quasi-slca}(q, v_{c_i})] \leq UB(q, v) - 1 + \prod_{i=1}^{m}(1 - Pr^G_{quasi-slca}(q, v_{c_i}))$.

Therefore, $UB^{'}(q, v)$ = $UB(q, v)$ - 1 + $\prod_{i=1}^{m}$(1 - $Pr^G_{quasi-slca}$ (q, v_{c_i})) holds for IND, ordinary and MUX nodes. □

Property 4: No matter node v is an IND or ordinary or MUX node, we can update their lower bound values as follows:

$$LB^{'}(q, v) = LB(q, v) - \sum_{i=1}^{m} Pr^G_{quasi-slca}(q, v_{c_i}) \quad (11)$$

Where $Pr^G_{quasi-slca}(q, v_{c_i}) \geq \sigma$ should be held.

Proof: For the lower bound update, we need to deduct the confirmed probability $[1 - \prod_{i=1}^{m}(1 - Pr^G_{quasi-slca}(q, v_{c_i}))]$ for IND nodes or $\sum_{i=1}^{m} Pr^G_{quasi-slca}(q, v_{c_i})$ for MUX nodes, from the original lower bound $LB(q, v)$. According to the procedure of the above proof, we have $\prod_{i=1}^{m}(1 - Pr^G_{quasi-slca}(q, v_{c_i})) \geq 1 - \sum_{i=1}^{m} Pr^G_{quasi-slca}(q, v_{c_i})$. Consequently, we have the inequation, $1 - \prod_{i=1}^{m}(1 - Pr^G_{quasi-slca}(q, v_{c_i})) \leq 1 - (1 - \sum_{i=1}^{m} Pr^G_{quasi-slca}(q, v_{c_i})) = \sum_{i=1}^{m} Pr^G_{quasi-slca}(q, v_{c_i})$. Therefore, it is safe to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

use the right side to update the lower bound values. □

Example 9: Consider a_4 that has been computed and its probability is 0.44. Given threshold σ (=0.4), a_4 is returned as a quasi-SLCA result. Consequently, we can update the lower/upper bound values of its ancestor a_2, i.e., $UB'(\{k_1, k_2\}, a_2)$ = 0.65 - 1 + (1 - 0.44) = 0.21 and $LB'(\{k_1, k_2\}, a_2)$ = 0.595 - 0.44 = 0.155. Since $UB'(\{k_1, k_2\}, a_2) < \sigma$, a_2 can be filtered out effectively without computation.

Property 3 is used to filter the unqualified nodes by reducing the upper bound value while Property 4 is used to quickly find the qualified required results by comparing the reduced lower bound value (for the probability of the remaining quasi-SLCAs) with the threshold value.

Sometimes, we need to calculate the probability distributions of keywords in a node if the given threshold σ is in the range $(LB(q, v), UB(q, v)]$. The basic computational procedure is similar to the PrStack algorithm in [18]. Different from the PrStack algorithm, we will introduce probability density function (PDF) to approximately calculate the probability for a node if the node depends on a large number of pruned descendant nodes. To decide when to invoke the PDF while avoiding the risk of reducing precision significantly, we would like to select and compute some descendant nodes that may contribute large probabilities to the node v. For the remaining descendant nodes, we may choose to invoke the PDF, by which we can reduce the time cost while still guarantee the precision to some extent. The detailed procedure will be introduced in the next section.

5 Prune-Based Probabilistic Threshold Keyword Query Algorithm

A key challenge of answering a PrTKQ is to identify the qualified result candidates and filter the unqualified ones as soon as possible. In this work, we address this challenge with the help of our proposed probabilistic inverted (PI) index. Two efficient algorithms are proposed, a comparable Baseline Algorithm and a PI-based Algorithm.

5.1 Baseline Algorithm

In keyword search on certain XML data, it is popular to use keyword inverted index retrieving the relevant keyword nodes, by which the keyword search results are generated based on different algorithms, e.g., [17], [15], [21], [22]. In probabilistic XML data, [18] proposed PrStack Algorithm to compute top-k SLCA nodes. In this section, we propose an effective Baseline Algorithm that is similar the idea of PrStack Algorithm. To answer PrTKQ, we need to scan all the keyword inverted lists once. Firstly, the keyword-matched nodes will be read one by one based on their document order. After one node is processed, we check if its probability can be up to the given threshold value σ. If it is true, the node can be output as a quasi-SLCA node and its remaining keyword distributions (i.e., containing partial query keywords) can be continuously promoted to its parent node. Otherwise, we promote its complete keyword distributions (i.e., containing both all keywords or partial keywords) to its parent node. After that, the node at the top of the stack will be popped. Similarly, the above procedures will be repeated until all nodes are processed. The basic algorithm can be terminated when all nodes are processed. The detailed procedure is shown in Algorithm 1.

Algorithm 1 Baseline Algorithm

input: a query $q = \{k_1, k_2, ..., k_m\}$ with threshold σ, keyword inverted (KI) index
output: a set R of quasi-SLCA nodes

```
1: load keyword node lists L = {l_1, l_2, ..., l_m} from KI index;
2: get the smallest node v w.r.t. document order from L;
3: initiate a stack S_1 using v;
4: while L ≠ φ do
5:    get the next smallest node v w.r.t. document order from L;
6:    while (S_1.top() ≺^not v) do
7:       x = S_1.pop();
8:       if x contains full keywords and Pr^G_{quasi-slca}(x) ≥ σ then
9:          output x into R;
10:      promote the rest keyword distributions of x to its parent x_p using CombineProb(x, x_p);
11:   S_1.push(v);
12: while S_1 ≠ φ do
13:    a new node v ← S_1.pop();
14:    if v contains full keywords and Pr^G_{quasi-slca}(v) ≥ σ then
15:       output v into R;
16:    promote the rest keyword distributions of v to its parent v_p using CombineProb(v, v_p);
17: return R;
```

5.2 PI-based Algorithm

To efficiently answer PrTKQ, the basic idea of PI-based Algorithm is to read the nodes from keyword node lists one by one in a bottom-up strategy. For each node, we quickly compute its lower bound and upper bound by accessing PI index, which is far faster than computing the keyword distributions of the node directly. After comparing its lower/upper bounds with the given threshold value, we can decide if the node should be output as a qualified answer, skipped as an unqualified result, or cached as a potential result candidate. For example, if the current node's lower bound is larger than or equal to the threshold value, then the node can be output directly without further computation. This is because all its descendants have been checked according to the bottom-up strategy. If its upper bound is lower than the threshold value,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Algorithm 2 PI-based Algorithm

input: a query $q = \{k_1, k_2, ..., k_m\}$ with threshold σ, keyword inverted (KI) index, PI index
output: a set R of quasi-SLCA nodes

1: load keyword node lists $L = \{l_1, l_2, ..., l_m\}$ from KI index;
2: load probability node lists $PIL = \{PIL_1, PIL_2, ..., PIL_m\}$;
3: get the smallest node w.r.t. document order v from L;
4: initiate a stack S_1 using v and an empty stack S_2;
5: **while** $L \neq \phi$ **do**
6: get the next smallest node w.r.t. document order v from L;
7: **while** (S_1.top() $\prec^{not}$ v) **do**
8: x = S_1.pop();
9: UB(q,x) and LB(q,x) $\leftarrow$ ComputeBound(x, $\{PIL_i(x)\}$);
10: **if** LB(q,x)$\geq \sigma$ **then**
11: output x into R;
12: UpdateBound($\{v_a \in S_1 | v_a \prec x\}$, LB($q$,$x$), UB($q$,$x$));
13: S_2.pop($v_d \in S_2 | v_d \succ x$);
14: **else if** UB(q,x) $\geq \sigma >$ LB(q,x) **then**
15: $Prob(x) \leftarrow$ ComputeProbDist(x, S_2);
16: **if** $Prob(x) \geq \sigma$ **then**
17: output x into R;
18: UpdateBound($\{v_a \in S_1 | v_a \prec x\}$, $Prob(x)$);
19: S_2.pop($v_d \in S_2 | v_d \succ x$);
20: **else**
21: $S_2.push(x)$;
22: S_1.push(v);
23: **while** $S_1 \neq \phi$ **do**
24: a new node $v \leftarrow S_1$.pop();
25: UB(q,v) and LB(q,v) $\leftarrow$ ComputeBound(v, $\{PIL_i(v)\}$);
26: process the node v using the same codes in Line 10 - Line 21;
27: **return** R;

then the node can be filtered out. Otherwise, it will be temporarily cached for further checking. Based on different cases, different operations would be applied. Only the nodes identified as potential result candidates need to be computed. Compared with Baseline Algorithm, PI-based algorithm can be accelerated significantly because Baseline Algorithm has to compute the keyword distributions for all nodes. The detailed procedure has been shown in Algorithm 2.

5.2.1 Detailed Procedure of PI-based Algorithm

In Algorithm 2, Line 1-Line 4 show that the procedures of initiating PI-based Algorithm. We first load the keyword node lists L from KI index and probability node lists PIL from PI index. And then we take the smallest node v (w.r.t. document order) from L to initiate a stack S_1. Another stack S_2 is used to maintain the temporary filtered nodes. After that, the PI-based Algorithm is ready to start.

Next, we need to check each node in L in document order. Different from Baseline Algorithm, we only compute the keyword distribution probabilities for a few nodes that are first identified using the lower bound and upper bound in PIL. Consider v be the next smallest node (w.r.t. document order) to be processed. We compare it with the node x in stack S_1. If v is the descendant node of x, then v will be pushed into S_1 and get the next smallest node (w.r.t. document order) from L. Otherwise, we pop out x from S_1 and check if it is a qualified quasi-SLCA answer. In Baseline Algorithm, it will compute the keyword distributions of x and combine its remaining distributions and the distribution of its parent based on promotion operations. Different from Baseline Algorithm, PI-based Algorithm will quickly compute the upper bound UB(q,x) and lower bound LB(q,x) using PIL, which is used to differentiate the nodes as qualified nodes - output, unqualified nodes - filter and uncertain nodes - to be further checked. By doing this, only a few nodes need to be computed. Since bound computation is far faster than computation of keyword distribution, lots of run time cost can be saved in PI-based Algorithm. Line 10-Line 21 show the detailed procedures. If the lower bound LB(q,x) is larger than or equal to the given threshold value σ, then x can be output as a qualified quasi-SLCA answer without computation. At this moment, the lower bound LB(q,x) can be taken as the temporary probability of x being a quasi-SLCA result because the exact probability of x is delayed until we need to calculate its exact probability value. Subsequently, the temporary probability value LB(q,x) and the probabilities of x' descendant quasi-SLCA results can be used to update the lower/upper bounds of the ancestors of x in stack S_1 based on Equation 11 and Equation 10, respectively. If the lower bound LB(q,x) is lower than σ while the upper bound UB(q,x) is larger than or equal to σ, then we need to compute the keyword distributions of x using the cached descendant nodes in S_2. Based on the computed probability $Prob(x)$ of x, it can be decided to be output as a qualified answer or filtered as an unqualified candidate. If the upper bound UB(q,x) is lower than σ, then x will be pushed into S_2 for the possible computation of its ancestors.

There are two main functions in PI-based Algorithm. The first one is ComputeProbDist(v, S_2) for computing the probability of full keyword distribution of v using the descendant nodes in S_2. The second is UpdateBound($\{v_a \prec v | v_a \in S_1\}$, LB($q$,$v$) or Prob($q$,$v$)) for updating the bounds of the nodes to be processed.

5.2.2 Function ComputeProbDist()

The function ComputeProbDist(v, S_2) can be implemented in two ways, *Exact Computation* or *Approximate Computation*.

Exact Computation is to actually calculate the probability of v being a quasi-SLCA node by scanning all the nodes in the stack S_2 that maintains the descendant nodes of v. The processing strategy is similar to Baseline Algorithm in Section 5.1. In other words, it

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

needs to visit the nodes in S_2 one by one and compute the local keyword distribution of each node, and then promotes the intermediate results to its parent. After all nodes in S_2 are processed, the probability of v will be obtained because it aggregates all the probabilities from its descendant nodes.

Approximate Computation is to approximately calculate the probability of v being a quasi-SLCA node based on a partial set of nodes in the stack S_2 that maintains the descendant nodes of v. The approximate computation can be made according to different distribution types, e.g., uniform distributions, piecewise polynomials, Poisson distributions, etc. In this work, we consider normal or Gaussian distributions in more detail.

As we know Gaussian distribution is considered the most prominent probability distribution in statistics. However, the PDF of Gaussian distribution cannot be applied to PrTKQ over probabilistic XML data directly due to two main challenges. The first challenge is to simulate the continuous distributions using discrete distributions based on the real conditions in order to reduce the approximate errors as much as possible, and the second is to embody the multiple keyword variables in the PDF.

Generally, the probability density function of a Gaussian distribution $N(\mu, \sigma^2)$ of mean μ and variance σ^2 is:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/(2\sigma^2)} \tag{12}$$

Addressing Challenge 1: The density function has a shape of a bell centered in the mean value μ with variance σ^2. Based on the definition of Gaussian distribution, the Gaussian distribution is often used to describe, at least approximately, measurements that tends to cluster around the mean. Therefore, consider the mean μ be the partial computed probability value of v be a quasi-SLCA node, which guarantees the real probability value will not be significantly different from the probability base that has already been calculated based on promising descendant nodes. The value of the variance σ^2 can be chosen from the range [1-#computed descendant nodes/#total descendant nodes, 1] based on the visited/unvisited descendant nodes in S_2. This is because the more the descendant nodes are actually computed, the higher the percentage of the values would be drawn within one standard deviation σ away from the mean. Extremely, if all descendant nodes are computed actually, 100% of values can be drawn within one standard deviation. Therefore, we select and compute a few descendant nodes of v from S_2, which can contribute relatively higher probabilities to make v a quasi-SLCA node. In this work, we use heuristic method to select a few descendant nodes with the higher probabilities of single keywords in the descendant nodes of v. And then, we take the partially computed probability as the base of the probability density function of a Gaussian distribution.

Consider v be a node to be evaluated and UB(q,v) [2] be its current upper bound value. We have,

$$Pr^{G,Gaussian}_{quasi-slca}(q, v) = \int_0^{UB(q,v)} f(x)dx \tag{13}$$

After substituting Equation 12 into Equation 13, we get,

$$Pr^{G,Gaussian}_{quasi-slca}(q, v) = \int_0^{UB(q,v)} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \tag{14}$$

Where μ is the partially computed probability, σ^2 is set as 1-#computed descendant nodes/#total descendant nodes.

Addressing Challenge 2: To embody all the keyword variables in the PDF, we introduce the joint/conditional Gaussian distribution based on the work in [23]. Assume a PrTKQ contains two keywords k_x and k_y. We have the conditional PDF as follows.

$$f_{Y|X}(y|x) = \frac{1}{\sqrt{2\pi(1-\rho^2)\sigma_Y^2}} e^{-\frac{[(y-\mu_Y)-(\rho(\frac{\sigma_Y}{\sigma_X})(x-\mu_X))]^2}{2\sigma_Y^2}} \tag{15}$$

Since $f(x, y) = f_X(x) * f_{Y|X}(y|x)$, after substituting Equation 12 into Equation 15, we get

$$f(x,y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} e^{-\frac{(x-\mu_X)^2}{2\sigma_X^2} - \frac{[(y-\mu_Y)-(\rho(\frac{\sigma_Y}{\sigma_X})(x-\mu_X))]^2}{2\sigma_Y^2}} \tag{16}$$

If we make an assumption that x and y are independent keyword variables i.e., $\rho = 0$, and assume $\mu_X = \mu_Y = \mu$ and $\sigma_X = \sigma_Y = \sigma$, then we have

$$f(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-\mu)^2+(y-\mu)^2}{2\sigma^2}} \tag{17}$$

Therefore, Equation 17 can be easily extended to multiple keyword variables that are assumed as independent. We can compute the probability of v w.r.t. a PrTKQ $\{k_1, k_2, ..., k_t\}$.

$$Pr^{G,Gaussian}_{quasi-slca}(q, v) = \int_0^{UB(q,v)} ... \int_0^{UB(q,v)} \frac{e^{-\frac{(x_1-\mu)^2+...+(x_t-\mu)^2}{2\sigma^2}}}{(2\pi)^{t/2}\sigma^t} dx_1...dx_t \tag{18}$$

Where μ is the partially computed probability, σ^2 is set as 1-#computed descendant nodes/#total descendant nodes.

In the experiments, we call Matlab from Java to calculate Equation 18. The estimated results are used to show the comparison between the actual computation

2. Note that UB(q,v) has been updated if v has descendant nodes that are qualified answers, i.e., it minus the probability contributions of the qualified answers.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

and approximation computation. The results verify the usability of Gaussian distribution to measure the probability.

5.2.3 Function UpdateBound()

For each ancestor node $v_a \in S_1$ ($v_a \prec v$), we need to update the upper bounds and lower bounds using Function UpdateBound() based on Equation 10 and Equation 11, respectively. To guarantee the completeness of the answer set, the parameters of the function may be different based on conditions. For example, if LB(q,v) is larger than or equal to the threshold value σ as shown in Algorithm 2: Line 12, then the probability value LB(q,v) is used to update the upper bounds of v' ancestors while the probability value UB(q,v) is used to update the lower bounds of v' ancestors; if LB(q,v) is smaller than σ and UB(q,v) is larger than or equal to σ as shown in Algorithm 2: Line 18, the actual or approximate probability value $Prob(v)$ computed by Function ComputeProbDist(v, S_2) will be utilized to update the upper/lower bounds of v' ancestors together.

Here, we use two hashmaps to implement Function UpdateBound(). For a node, one hashmap is used to cache the node encoding number as a key, and the lower/upper bounds as a value where the bounds are computed based on PI index. Another hashmap is used to record the probability that the descendants of the node having been identified as qualified quasi-SLCA answers. When a node is coming, we can quickly get the updated lower/upper bounds based on the two hashmaps.

6 Experimental Studies

We conduct extensive experiments to test the performance of our algorithms: Baseline Algorithm (BA); PI-based Exact-computation Algorithm (PIEA) that implements Function ComputeProbDist() by exactly computing the probability distributions of the keyword matched nodes; and PI-based Approximate-computation Algorithm (PIAA) that makes approximated computation based on the Gaussian distribution of keywords while still exactly computing the probability distributions of the keyword matched nodes that have the higher probabilities. All these algorithms were implemented in Java and run on a 3.0GHz Intel Pentium 4 machine with 2GB RAM running Windows 7.

6.1 Dataset and Queries

We use three real datasets, DBLP [24], Mondial [25], and INEX [26] and a synthetic XML benchmark dataset XMark [27] for testing the proposed algorithms. For XMark, we also generate four datasets with different sizes. The four types of datasets are selected based on their features. DBLP is a relatively shallow dataset of large size; Mondial is a deep and complex, but small dataset; INEX is a large dataset with complex structures; XMark is a balanced dataset with varied depth, complex structure and varied size. Therefore, they are chosen as test datasets.

TABLE 1
Properties of PrXML data

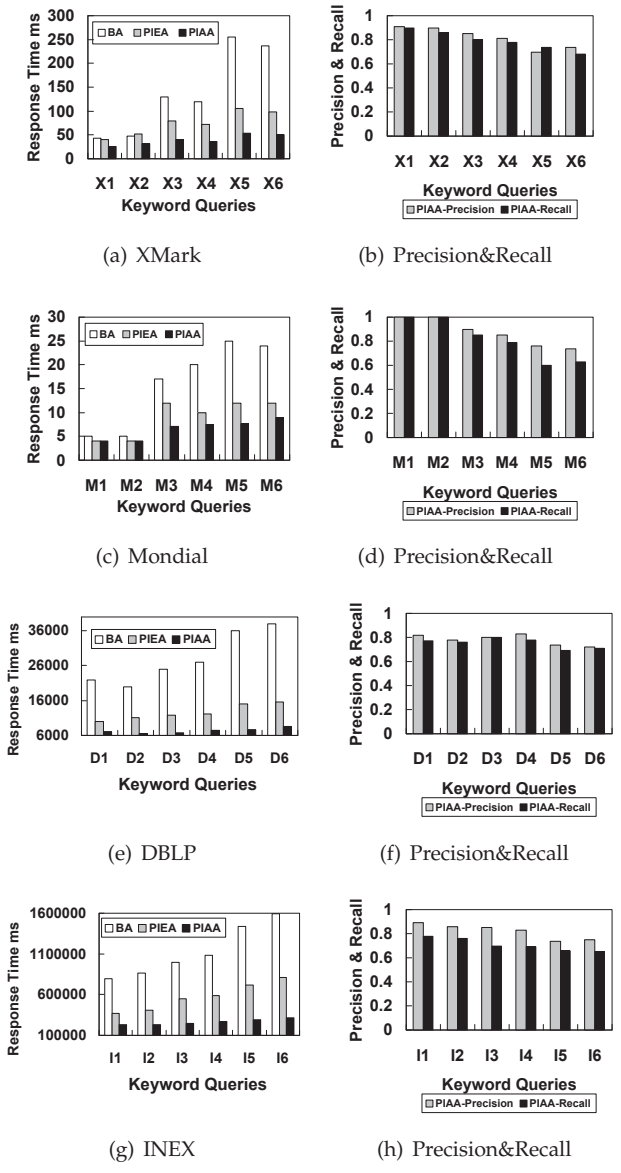
ID	name	size	#IND	#MUX	#Ordinary
Doc1	XMark	10M	26k	26k	145k
Doc2		20M	54k	52k	200k
Doc3		40M	98k	100k	606k
Doc4		80M	329k	368k	1M
Doc5	Modial	1.2M	8k	9k	20k
Doc6	DBLP	136M	759k	589k	3M
Doc7	INEX	5,898M	13M	10M	52M

For each XML dataset used, we generate the corresponding probabilistic XML tree, using the same method as used in [12]. We visit the nodes in the original XML tree in pre-order way. We first set the random ratio of IND:MUX:Ordinary as 3:3:4. For each node v visited, we randomly generate some distributional nodes with "IND" or "MUX" types as children of v. Then, for the original children of v, we choose some of them as the children of the new generated distributional nodes and assign random probability distributions to these children with the restriction that the sum of them for a MUX node is no greater than 1. The generated datasets are described in Table 1. And we select terms and construct a set of keyword queries to be tested for each dataset. Due to the limited space, we only show six of these queries for each dataset. For each different sets of queries, the terms in the first two queries have small size of keyword matched nodes; the terms of the middle two queries relate to a medium size of keyword matched nodes; the terms of the last queries are based on the computation of a larger number of keyword matched nodes.

6.2 Varying Keyword Queries

Figure 5 shows the experimental results of six queries for each selected datasets where X represents the queries over 20MB XMark dataset, M represents the queries over Mondial dataset, D represents the queries over DBLP dataset and I represents the queries over INEX dataset. And the required threshold value σ is set as 0.3. From the results, we can find that compared with the BA algorithm, most of time the PIEA algorithm can reduce the response time by nearly 40% using the pruning techniques based on the updated lower/upper bounds. The PIAA algorithm can further improve the time efficiency by about 20% with the assumption of probability distribution of keywords. For $X1$, $X2$ and M_1, M_2, the response time of BA algorithm is approaching to the time cost of the other two algorithms. Especially for query $X2$, PIEA algorithm is overwhelmed by BA algorithm. This is because both the number of keyword-matched nodes and the size of answer sets are smaller than the other queries. From the four figures on the left

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.



(a) XMark (b) Precision&Recall

(c) Mondial (d) Precision&Recall

(e) DBLP (f) Precision&Recall

(g) INEX (h) Precision&Recall

Fig. 5. Evaluation of Keyword Queries over XMark20M, Mondial, DBLP, INEX where σ=0.3

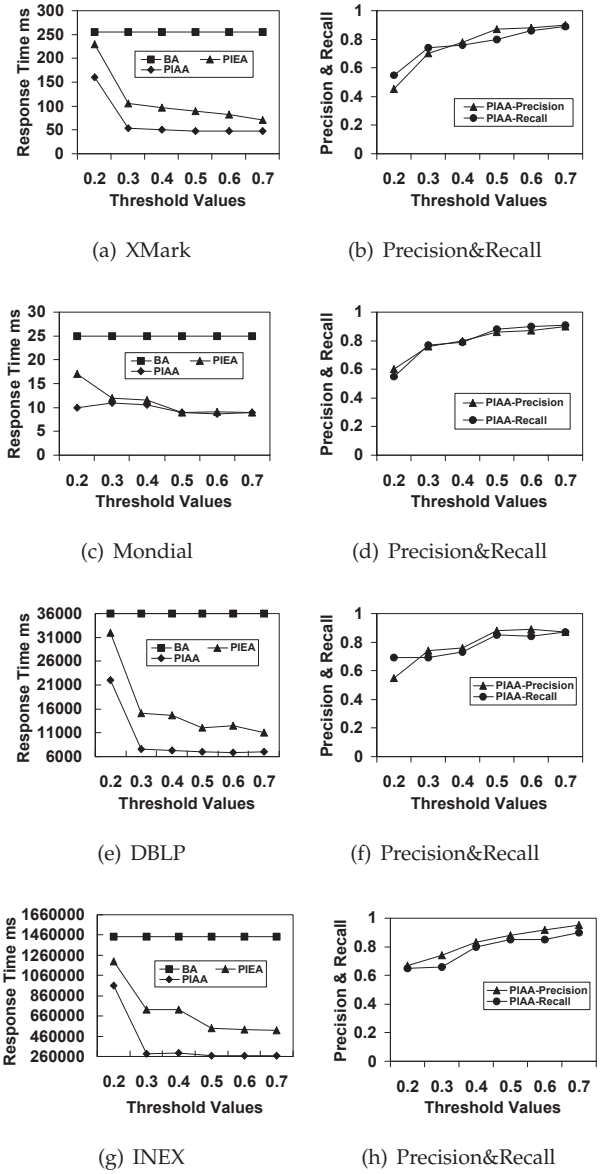
side of Figure 5, we find that the scalability of PIEA and PIAA algorithms is much better than that of BA algorithm by testing the queries with different sizes of answer sets.

To measure the precision and recall of PIAA algorithm, we utilize the P&R equations in information retrieval area as follows.

$Precision = \frac{|R_{BA} \cap R_{PIAA}|}{|R_{PIAA}|}$; $Recall = \frac{|R_{BA} \cap R_{PIAA}|}{|R_{BA}|}$

Because PIEA algorithm can find the same results with BA algorithm by exactly computing the required probability distributions, Figure 5 only demonstrates the precision and recall of PIAA algorithm for different queries over each dataset. From the experimental results, we find that the precision and recall can reach up to at least 0.7 for XMark, 0.6 for Mondial, 0.7 for DBLP, and 0.66 for INEX, respectively. Sometimes, it can be up to 0.9 at most, e.g., X1, X2, M1, M2, I1, I2, etc. Comparing all the tested queries, we can get a general conclusion that the precision and recall will be decreased with the increase of potential result size. However, from the experiments, they will not be lower than 0.6 because (1) the results with higher probabilities are exactly selected and computed, which does not need to depend on the Gaussian assumption; (2) the rest minor results are estimated by using Gaussian assumption over the keyword distributions that have been excluded by the results with higher probabilities. In other words, PIAA strategy can return the percentage ($\geq$0.6) of significant results, but may underestimate the minor results.

6.3 Varying Threshold Values



(a) XMark (b) Precision&Recall

(c) Mondial (d) Precision&Recall

(e) DBLP (f) Precision&Recall

(g) INEX (h) Precision&Recall

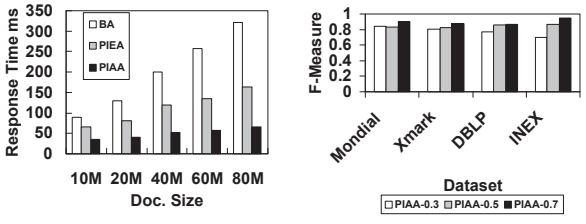
Fig. 6. Time, Precision and Recall vs. Varied Threshold

To test the adaptability of the proposed algorithms

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

to threshold query, we test the changes of response time and precision&recall with the increase of threshold value. Figure 6 shows the experimental results when the threshold value varies from 0.2 to 0.7 for queries X5, M5, D5 and I5. The left four figures in Figure 6 show that PIEA and PIAA algorithms can overwhelm BA algorithm greatly with the increase of threshold value. This is because BA algorithm has to scan and compute all the relevant nodes while PIEA and PIAA algorithms can skip more nodes when the threshold value becomes large. However, when the threshold value is up to 0.5, the change of the time cost will be smooth because once a quasi-SLCA node is identified, its ancestor nodes can be skipped definitely, which is true for the threshold values larger than 0.5. From the right four figures in Figure 6, we can find that the precision and recall will be affected by the change of threshold values. When the threshold value reaches up to 0.5, the precision and recall can be up to 0.8 at least. On the contrary, if the threshold value is lower than 0.2, the precision and recall would be decreased to 0.5 based on the selected datasets.

6.4 Varying Probabilistic Document Size



(a) XMark (X3, σ=0.3) (b) A Variant of F-Measure

Fig. 7. Response Time and F-Measure for different datasets

We firstly take XMark dataset as an example to test the performance of the three algorithms when we increase the document size. We test all the six queries of XMark dataset, but in this paper, we only show the results of the query $X3$ where the threshold value is specified as 0.3. From Figure 7(a), we can see that the response time of all the three algorithms will be increased when the document size increases from 10MB to 80MB. However, the increase of PIEA and PIAA algorithms is much slower. Particularly, PIAA just changes a bit. The comparison illustrates that PIEA and PIAA algorithms can obtain much better performance than BA algorithm. In addition, all algorithms show linear degradation, i.e., they have the similar scalability.

Secondly, we evaluate the precision and recall of PIAA algorithm using a variant of F-measure that aggregates the precision and recall of all queries together.

$$F - measure = 2 * \frac{\overline{P(q_i)} * \overline{R(q_i)}}{\overline{P(q_i)} + \overline{R(q_i)}}$$

Where $\overline{P(q_i)} = \sum_1^6 (P(q_i))/6$, and $\overline{R(q_i)} = \sum_1^6 (R(q_i))/6$.

To evaluate the F-measure of PIAA algorithm, we test 24 queries with different threshold values: 0.3, 0.5 and 0.7. From the results in Figure 7(b), we can find that the F-measure can be over 0.75 for all datasets.

7 Related Work

The topic of probabilistic XML has been studied recently. Many models have been proposed, together with structured query evaluations. Nierman et al. [7] first introduced a probabilistic XML model, ProTDB, with the probabilistic types IND - *independent* and MUX - *mutually-exclusive*. Hung et al. [8] modeled the probabilistic XML as directed acyclic graphs, supporting arbitrary distributions over sets of children. Keulen et al. [9] used a probabilistic tree approach for data integration where its probability and possibility nodes are similar to MUX and IND, respectively. Cohen et al. [28] incorporated a set of constraints to express more complex dependencies among the probabilistic data. They also proposed efficient algorithms to solve the constraint-satisfaction, query evaluation, and sampling problem under a set of constraints. In [12], Kimelfeld et al. summarized and extended the probabilistic XML models previously proposed, the expressiveness and tractability of queries on different models are discussed with the consideration of IND and MUX. [11] studied the problem of evaluating twig queries over probabilistic XML that may return incomplete or partial answers with respect to a probability threshold to users. [13] proposed and addressed the problem of ranking top-k probabilities of answers of a twig query. All the above work focused on the discussions of probabilistic XML data model and/or structured XML query, e.g., twig query. The most closely related work is [18] that proposed two algorithms to answer top-k keyword queries over probabilistic XML data. However, compared with [18], in this work we propose a probabilistic inverted index that can be used to efficiently answer threshold keyword queries by reducing the computational cost of unqualified nodes. In addition, we also take into account the relaxation (i.e., quasi-SLCA) of results for keyword search w.r.t. a threshold value while [18] focused on the strict SLCA semantics of results.

8 Conclusions

In this work, we first proposed and investigated the problem of finding *quasi*-SLCA for PrTKQs over probabilistic XML data. And then we designed a PI index and analyzed the pruning features of PI index. Based on the lower and upper bounds computed from PI index, the proposed PI-based algorithm can quickly identify the qualified results and filter the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

unqualified ones. Our experimental results demonstrated the comparison of Baseline algorithm, PI-based Exact-computation Algorithm (PIEA) and PI-based Approximate-computation Algorithm (PIAA), which verified our motivation and approaches.

9 Acknowledgments

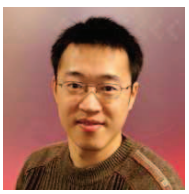
This work was supported by the ARC Discovery Projects under Grant No. DP110102407.

References

[1] R. Cheng and S. Prabhakar, "Managing uncertainty in sensor database," *SIGMOD Record*, vol. 32, no. 4, pp. 41–46, 2003.
[2] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter, "Efficient indexing methods for probabilistic threshold queries over uncertain data," in *VLDB*, 2004, pp. 876–887.
[3] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar, "Indexing multi-dimensional uncertain data with arbitrary probability density functions," in *VLDB*, 2005, pp. 922–933.
[4] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: a probabilistic threshold approach," in *SIGMOD Conference*, 2008, pp. 673–686.
[5] Y. Qi, R. Jain, S. Singh, and S. Prabhakar, "Threshold query optimization for uncertain data," in *SIGMOD Conference*, 2010, pp. 315–326.
[6] P. Senellart and S. Abiteboul, "On the complexity of managing probabilistic xml data," in *PODS*, 2007, pp. 283–292.
[7] A. Nierman and H. V. Jagadish, "ProTDB: Probabilistic data in xml," in *VLDB*, 2002, pp. 646–657.
[8] E. Hung, L. Getoor, and V. S. Subrahmanian, "Pxml: A probabilistic semistructured data model and algebra," in *ICDE*, 2003, pp. 467–.
[9] M. van Keulen, A. de Keijzer, and W. Alink, "A probabilistic xml approach to data integration," in *ICDE*, 2005, pp. 459–470.
[10] S. Abiteboul and P. Senellart, "Querying and updating probabilistic information in xml," in *EDBT*, 2006, pp. 1059–1068.
[11] B. Kimelfeld and Y. Sagiv, "Matching twigs in probabilistic xml," in *VLDB*, 2007, pp. 27–38.
[12] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv, "Query efficiency in probabilistic xml models," in *SIGMOD Conference*, 2008, pp. 701–714.
[13] L. Chang, J. X. Yu, and L. Qin, "Query ranking in probabilistic xml data," in *EDBT*, 2009, pp. 156–167.
[14] N. Fuhr and K. Großjohann, "Xirql: A query language for information retrieval in xml documents," in *SIGIR*, 2001, pp. 172–180.
[15] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, "XRANK: Ranked Keyword Search over XML Documents," in *SIGMOD Conference*, 2003, pp. 16–27.
[16] G. Koloniari and E. Pitoura, "Lca-based selection for xml document collections," in *WWW*, 2010, pp. 511–520.
[17] Y. Xu and Y. Papakonstantinou, "Efficient Keyword Search for Smallest LCAs in XML Databases," in *SIGMOD Conference*, 2005, pp. 537–538.
[18] J. Li, C. Liu, R. Zhou, and W. Wang, "Top-k keyword search over probabilistic xml data," in *ICDE*, 2011, pp. 673–684.
[19] P. A. Boncz, S. Manegold, and J. Rittinger, "Updating the pre/post plane in monetdb/xquery," in *XIME-P*, 2005.
[20] I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang, "Storing and querying ordered xml using a relational database system," in *SIGMOD Conference*, 2002, pp. 204–215.
[21] Z. Bao, T. W. Ling, B. Chen, and J. Lu, "Effective xml keyword search with relevance oriented ranking," in *ICDE*, 2009, pp. 517–528.
[22] J. Li, C. Liu, R. Zhou, and W. Wang, "Suggestion of promising result types for xml keyword search," in *EDBT*, 2010, pp. 561–572.
[23] D. P. Bertsekas and J. N. Tsitsiklis, "Introduction to probability, 1st edition," *Athena Scientific*, vol. 4.7, 2002.
[24] "http://dblp.uni-trier.de/xml/."
[25] "http://www.dbis.informatik.uni-goettingen.de/mondial."
[26] L. Denoyer and P. Gallinari, "The wikipedia xml corpus," in *SIGIR Forum*, 2006.
[27] "http://monetdb.cwi.nl/xml/."
[28] S. Cohen, B. Kimelfeld, and Y. Sagiv, "Incorporating constraints in probabilistic xml," *ACM Trans. Database Syst.*, vol. 34, no. 3, 2009.

Jianxin Li received his BE and ME degrees in computer science, from the Northeastern University, China, in 2002 and 2005, respectively. He received his PhD degree in computer science, from the Swinburne University of Technology, Australia, in 2009. He is currently a postdoctoral research fellow at Swinburne University of Technology. His research interests include XML database, keyword search, and query processing and optimization.

Chengfei Liu received the BS, MS and PhD degrees in Computer Science from Nanjing University, China in 1983, 1985 and 1988, respectively. Currently he is a Professor in the Faculty of Information and Communication Technologies, Swinburne University of Technology, Australia. His current research interests include keywords search on structured data, query processing and refinement for advanced database applications, query processing on uncertain data and big data, and data-centric workflows. He is a member of IEEE.



Rui Zhou is a research fellow currently working at Swinburne University of Technology, Australia, where he finished his PhD. in 2010. He received his BS. and MS. at Northeastern University, China in 2004 and 2006 respectively. His research interest is mainly about query processing on XML data and graph data.

Jeffrey Xu Yu received the BE, ME, and PhD degrees in computer science, from the University of Tsukuba, Japan, in 1985, 1987, and 1990, respectively. Currently he is a Professor in the Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong. His major research interests include graph mining, graph database, social networks, keyword search, and query processing and optimization. He is a senior member of the IEEE, a member of the IEEE Computer Society, and a member of ACM.