Information Sciences 179 (2009) 3745–3762

Contents lists available at ScienceDirect

Information Sciences

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

SAIL: Structure-aware indexing for effective and progressive top-*k* keyword search over XML documents

Guoliang Li [a,*], Chen Li [b], Jianhua Feng [a], Lizhu Zhou [a]

[a] Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China
[b] Department of Computer Science, University of California, Irvine, CA 92697-3435, USA

ARTICLE INFO

Article history:
Received 7 January 2009
Received in revised form 6 April 2009
Accepted 14 June 2009

Keywords:
XML keyword search
Minimal-cost trees
Structure-aware indices
Relevance ranking
Keyword-pair-based ranking
Schema-aware dewey code

ABSTRACT

Keyword search in XML documents has recently gained a lot of research attention. Given a keyword query, existing approaches first compute the lowest common ancestors (LCAs) or their variants of XML elements that contain the input keywords, and then identify the subtrees rooted at the LCAs as the answer. In this the paper we study how to use the rich structural relationships embedded in XML documents to facilitate the processing of keyword queries. We develop a novel method, called SAIL, to index such structural relationships for efficient XML keyword search. We propose the concept of *minimal-cost trees* to answer keyword queries and devise structure-aware indices to maintain the structural relationships for efficiently identifying the minimal-cost trees. For effectively and progressively identifying the top-*k* answers, we develop techniques using link-based relevance ranking and keyword-pair-based ranking. To reduce the index size, we incorporate a numbering scheme, namely *schema-aware dewey code*, into our structure-aware indices. Experimental results on real data sets show that our method outperforms state-of-the-art approaches significantly, in both answer quality and search efficiency.

1. Introduction

Keyword search is a widely accepted mechanism for querying document systems and the World Wide Web. Many researchers have been studying challenges related to keyword search on XML documents [9,15,26,34,36,47,50,51]. One important advantage of keyword search is that it enables users to search information without knowing a complex query language such as XPath or XQuery, or having prior knowledge about the structure of the underlying data.

Many algorithms for XML keyword search use the notion of "lowest common ancestors" (LCAs) in the labeled tree modeled from an XML document [9,15,34,47,50,51]. Intuitively, an LCA of a set of keywords is a lowest node in the tree that is the common ancestor of nodes with these keywords. For a keyword query, these algorithms first retrieve *content nodes* in the XML document that contain the input keywords using inverted indices. They then identify the LCAs of the content nodes, and take the subtrees rooted at the LCAs as the answer to the query. For example, a bibliography XML document is shown in Fig. 1. Suppose a user issues a keyword query "`DB Tom`". Nodes 2, 12, and 15 are LCAs of the keyword query. Notice that node 2 is the LCA of nodes 13 and 17. Evidently, node 2 is less relevant to the query than nodes 12 and 15, as nodes 13 and 17 correspond to values of different papers. To address this limitation of using LCAs as query answers, many methods have been proposed [26,34,36,47,50] to improve search efficiency and effectiveness. They focus on proposing different semantics based on LCAs and their variants to improve answer quality, and studying efficient algorithms for computing query answers.

* Corresponding author. Tel.: +86 10 62789150; fax: +86 10 627871138.
E-mail addresses: liguoliang@tsinghua.edu.cn (G. Li), chenli@ics.uci.edu (C. Li), fengjh@tsinghua.edu.cn (J. Feng), dcszlz@tsinghua.edu.cn (L. Zhou).

0020-0255/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2009.06.025

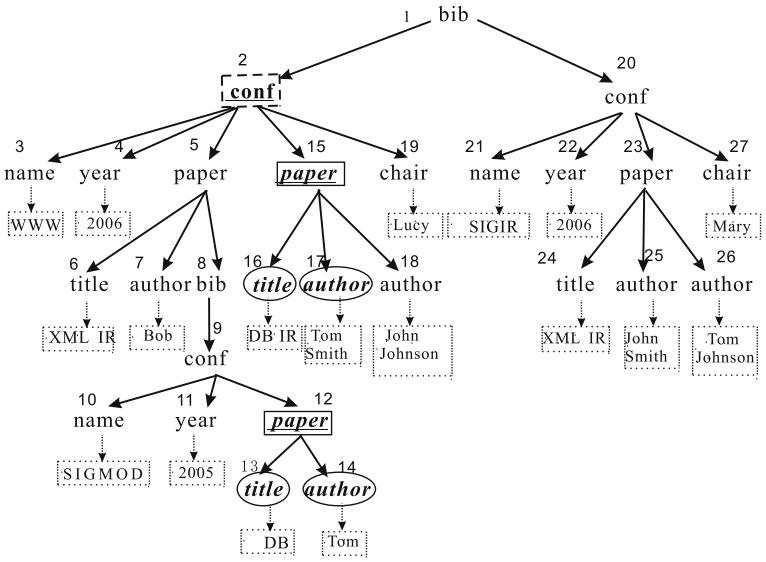

Fig. 1. Keyword search using lowest common ancestors (LCAs).

Existing algorithms have two main limitations. First, they use the default "AND" semantics between the input keywords in a query, thus ignoring nodes that are relevant to some of the query keywords (but not all the keywords). For example, suppose a user issues a keyword query "DB IR Tom" on the document above. The LCAs to the query are nodes 15, 5, and 1. Although nodes 12 and 23 do not have leaf nodes corresponding to all the three keywords, they might still be more relevant than nodes 5 and 1, which contain many irrelevant papers. Second, in order to compute the best results to a query, existing methods find candidate nodes first before ranking them, and this approach is not efficient for computing the best answers. In the example above, existing methods need to first identify the LCAs of the three keywords, i.e., nodes 15, 5, and 1, and then rank them. A more efficient algorithm might be able to find the best answers without generating all candidate nodes.

To address these limitations, we propose a structure-aware indexing method for effective XML keyword search, called SAIL. We develop novel structure-aware ranking techniques and efficient search algorithms. In our approach, each node on the XML tree could be potentially relevant to a keyword query, and we use a ranking function to decide the best answers to the query. For each keyword in the tree, we index not only the nodes containing the keyword, but also those nodes whose descendants contain the keyword. For instance, consider the XML document in Fig. 1. For the keyword "DB", we index nodes 13, 16, 12, 15, 9, 2, 8, 1, and 5 for this keyword. For the keyword "IR", we index nodes 6, 16, 24, 5, 15, 23, 2, 20, and 1. For the keyword "Tom", we index nodes 14, 17, 26, 12, 15, 23, 9, 2, 20, 8, 1, and 5. The nodes are sorted by their relevance to the keyword. We develop different ranking techniques, one based on a link analysis, and another based on keyword-pair relevancy. Our approach automatically supports "OR" semantics.

We can use this index to efficiently compute the best answers to a query. In particular, it allows us to use the classic threshold-based techniques [12] to progressively and efficiently identify the top-k relevant answers to a query. For instance, for the keyword query "DB IR Tom" in our running example, we can use the index to compute nodes 15, 12, and 23 as the top-3 relevant nodes to the query. In addition, we study how to reduce the index size by using a novel numbering scheme, namely *schema-aware dewey code*. To the best of our knowledge, our work is the first study that indexes structural relationships to improve answer quality and efficiency of XML keyword search. To summarize, we make the following contributions:

- We propose the concept of "minimal-cost trees" to answer keyword queries over XML documents. We develop ranking techniques based on link analysis and keyword-pair for finding relevant answers. We rank the minimal-cost trees by taking into account both the tf*idf based document relevance in the IR literature and the structural compactness of minimal-cost trees from a database point of view.
- We devise novel structure-aware index structures by storing the structural relationships embedded in the XML document for efficient keyword search. We adopt threshold-based techniques to progressively and efficiently identify the top-k relevant answers.
- We employ a numbering scheme, namely *schema-aware dewey code*, in our structure-aware indices to reduce the index size.
- We have conducted an extensive experimental study using both real datasets and synthetic datasets. The results show that our method can find relevant results and achieve high search efficiency, and outperforms state-of-the-art approaches significantly.

The remainder of this paper is organized as follows. Section 2 introduces new ranking functions for effective XML keyword search. We propose structure-aware indices in Section 3. Section 4 presents a numbering scheme to reduce the index size. Extensive experimental evaluations are provided in Section 5. We review related work in Section 6, and conclude in Section 7.

2. Relevance-based answers to keyword queries

In this section we introduce a new framework to find relevant answers to a keyword query on an XML document. In the framework, each node on the tree is relevant to the query to some degree. For the node, we define its corresponding answer to the query as its subtree with paths to nodes that include the query keywords. This subtree is called the "minimal-cost tree" (MCT for short) for this node (Section 2.1). Different nodes correspond to different answers to the query, and we will study how to quantify the relevance of each answer to the query for ranking (Section 2.2).

For ease of presentation, we first introduce some notations. An XML document can be modeled as a rooted, ordered, and labeled tree. A node v in the tree corresponds to an element in the XML document, and has a label $\lambda(v)$. A query $\mathcal{K}$ consists of a set of keywords $\{k_1, k_2, \ldots, k_m\}$. For each keyword k_i, we call the nodes in the tree that contain the keyword the *content nodes* for k_i. The ancestor nodes[1] of the content nodes are called the *quasi content nodes* of the keyword. For example, in Fig. 1, `title` (node 16) is a content node for the keyword "`DB`", and `conf` (node 2) is a quasi content node of the keyword.

2.1. Answer to a keyword query for a node

Consider an XML document $\mathcal{D}$, a keyword k_i, and a content node or quasi content node n for k_i. A *pivotal node* for keyword k_i and the node n, is a content node for k_i, which is either a descendant of n or n itself, with a minimal distance to n. The path from node n to this node is called the *pivotal path* of this pivotal node. In general, there can be more than one pivotal node for k_i and n. For example, in the XML document in Fig. 1, for a given keyword `DB`, node 9 is a quasi content node for `DB`. Node 13 is a pivotal node for node 9 and the keyword `DB`, and the path $n_9 \to n_{12} \to n_{13}$ is the corresponding pivotal path, where n_9 denotes node 9. Intuitively, a pivotal node for a node n and k_i is a very relevant node to n for this keyword.

Given a keyword query, each node n in the XML document is potentially relevant to the query. We introduce the notion of *minimal-cost tree* (MCT for short) to define the answer to the query with respect to the node n. Later we will discuss how to use relevance functions to quantify the quality of this answer.

Definition 1 (*MCT*). Given an XML document $\mathcal{D}$, a node n in $\mathcal{D}$, and a keyword query $\mathcal{K} = \{k_1, k_2, \ldots, k_m\}$, the minimal-cost tree of the query and node n is the subtree rooted at n that includes all the pivotal paths for the pivotal nodes with respect to the input keywords and node n.

Example 1. Consider the XML document in Fig. 1 and a keyword query "`WWW DB Tom`". Nodes 3, 13, 14, 16, 17, and 26 are content nodes of the three keywords; nodes 1, 2, 5, 8, 9, 12, 15, 20, and 23 are their quasi content nodes. Node 3 is the pivotal node for node 2 and `WWW`. Node 16 is the pivotal node for node 2 and `DB`. Node 17 is the pivotal node for node 2 and `Tom`. The MCT of node 2 is the subtree rooted at node 2, which contains the paths: $n_2 \to n_3, n_2 \to n_{16}$, and $n_2 \to n_{17}$.

The main advantage of this definition is that, even if a node does not have descendant nodes that include all the keywords in the query, this node could still be considered as a potential answer. In other words, this definition is relaxing the assumption in existing semantics that all the query keywords need to appear in the descendants of an answer node. As we will see in the next section, this definition still allows us to do indexing to answer queries efficiently.

2.2. Ranking query answers

Now we discuss how to rank the MCT for a node n as an answer to the query. Intuitively, we first evaluate the relevance between node n and each input keyword, and then combine these relevance scores as the overall score of the MCT. We will focus on different methods to quantify the relevance of node n to a query keyword, and how to combine these relevance scores.

2.2.1. Method 1: ranking based on tf*idf

Our first ranking method models each node as a document that includes the terms in its subtree. We can then use the idea of tf*idf to score the relevance of the node to a keyword. Given an XML document $\mathcal{D}$, suppose there are p nodes and q keywords in $\mathcal{D}$. Given a node $n \in \mathcal{D}$ and a keyword k_i ($1 \leqslant i \leqslant q$) contained in n, we denote $tf(k_i, n)$ as the term frequency of k_i in n, which is the number of occurrences of k_i in n. We denote $idf(k_i)$ as the inverse document frequency of k_i, i.e., $idf(k_i) = \frac{p+1}{O_{k_i}+1}$, where O_{k_i} is the number of nodes that include k_i. We denote $ntl(n)$ as the normalized term length of n, i.e., $ntl(n) = \frac{|n|}{|n_{max}|}$, where $|n|$ denotes the number of terms in n and $|n_{max}|$ denotes the node with the maximal number of terms.

[1] A node is not an ancestor nor a descendant of itself.

Consider an input keyword query $\mathscr{K} = \{k_1, k_2, \ldots, k_m\}$. Using our first ranking method, the relevance of the node n to a keyword k_i is defined as:

$$Score_1(n, k_i) = \frac{ln(1 + tf(k_i, n)) * ln(idf(k_i))}{(1 - s) + s * ntl(n)}. \quad (1)$$

In the formula, s is a constant (usually set to 0.2 [35]). The relevance of the node n to all the keywords is defined as:

$$Score_1(n, \mathscr{K}) = \sum_{k=1}^{m} Score_1(n, k_i). \quad (2)$$

For instance, consider the XML document in Fig. 1. For node 24, we have $Score_1(n_{24}, \texttt{XML}) = \frac{ln(1+1)*ln(28/3)}{0.2+0.8} = 1.55$ and $Score_1(n_{24}, \texttt{IR}) = \frac{ln(1+1)*ln(28/4)}{0.2+0.8} = 1.35$. For node 25, we have $Score_1(n_{25}, \texttt{John}) = \frac{ln(1+1)*ln(28/3)}{0.2+0.8} = 1.55$.

2.2.2. Method 2: ranking based on ancestor–descendant relationships

Method 1 cannot rank a quasi content node that does not contain a query keyword. To address this issue, we extend it in order to effectively score such quasi content nodes. Given a keyword k_j, a content node c, and a quasi content node n w.r.t. c and k_j, the distance between n and c can indicate how relevant the node n is to keyword k_j. Based on this observation, our second ranking method scores the relevance n w.r.t. k_j as follows:

$$Score_2(n, k_j) = \sum_{p \in P} \alpha^{\delta(n,p)} * Score_1(p, k_j), \quad (3)$$

where P is the set of pivotal nodes for n and k_j, α is a damping factor between 0 and 1, and $\delta(n, p)$ denotes the distance between node n and node p. As the distance between n and p increases, n becomes less relevant to k_j. Our experiments suggested that a good value for α is 0.8 (Section 5). The computation of $Score_{\text{IR}}(p, k_j)$ is using Eq. (1) as p must contain k_j. Accordingly, if n is a content node for k_j, we adopt Eq. (1) to score n w.r.t. k_j; otherwise, if n is a quasi content node for k_j, we use Eq. (3) to compute the score.

Given a keyword query $\mathscr{K} = \{k_1, k_2, \ldots, k_m\}$ and a node n, we take the sum of the scores of node n on every k_i as the overall score of node n on $\mathscr{K}$:

$$Score_{\text{SK}}(n, \mathscr{K}) = \sum_{k=1}^{m} Score_{\text{SK}}(n, k_i), \quad (4)$$

where $Score_{\text{SK}}(n, k_i)$ denotes the single keyword score of keyword k_i to node n. $Score_{\text{SK}}(n, k_i) = Score_1(n, k_i)$ if n is a content node for k_i; $Score_{\text{SK}}(n, k_i) = Score_2(n, k_i)$ if n is a quasi content node for k_i.

Example 2. For instance, consider the XML document in Fig. 1. Given a keyword query $\mathscr{K} = \{\texttt{XML}, \texttt{IR}, \texttt{John}\}$, for node 23, we have

$$Score_{\text{SK}}(n_{23}, \texttt{XML}) = \alpha^1 * Score_{\text{SK}}(n_{24}, \texttt{XML}) = 0.8 * 1.55 = 1.24;$$
$$Score_{\text{SK}}(n_{23}, \texttt{IR}) = \alpha^1 * Score_{\text{SK}}(n_{24}, \texttt{IR}) = 0.8 * 1.35 = 1.08;$$
$$Score_{\text{SK}}(n_{23}, \texttt{John}) = \alpha^1 * Score_{\text{SK}}(n_{25}, \texttt{John}) = 0.8 * 1.55 = 1.24;$$
$$Score_{\text{SK}}(n_{23}, \mathscr{K}) = 1.08 + 1.24 + 1.24 = 3.56.$$

Suppose a user submits a query "IR John Smith". Nodes 15 and 23 have the same score. However, the latter is more relevant to the query, as keywords "John" and "Smith" are in the same node. This example shows that we can improve the ranking functions by considering the structural relationships.

2.2.3. Method 3: ranking based on keyword-pairs

Example 2 suggests that we can improve the second ranking method by considering keyword-pairs. Consider two keywords k_i and k_j, and a node n that contains the two keywords. If k_i and k_j are in the same pivotal node, they are related to each other. Moreover, the smaller the distance between the pivotal nodes w.r.t. k_i and k_j is, they are more relevant to node n. Based on this observation, we can measure the relevance between the two keywords w.r.t. node n as follows:

$$Rel(\langle k_i, k_j \rangle | n) = max\{\alpha^{\delta(p_i, p_j)} | p_i, p_j \text{ are pivotal nodes for } n \text{ w.r.t } k_i \text{ and } k_j\}, \quad (5)$$

where $\delta(p_i, p_j)$ denotes the minimal distance between p_i and p_j. This relevance based on keyword-pairs captures the structural information between two keywords.

Notice that if $Rel\langle k_i, k_t \rangle | n)$ and $Rel(\langle k_t, k_j \rangle | n)$ are large, $Rel(\langle k_i, k_j \rangle | n)$ should also be large, and thus k_i, k_j, and k_t should be very relevant to each other. We can use keyword-pair-based relevance to effectively quantify the compactness of a minimal-cost tree by considering the structural relationships among the keywords. Our third ranking method computes the score of a node n w.r.t. a keyword-pair $\langle k_i, k_j \rangle$ as:

$$Score_{\text{KP}}(n, \langle k_i, k_j \rangle) = Rel(\langle k_i, k_j \rangle | n) * (Score_{\text{SK}}(n, k_i) + Score_{\text{SK}}(n, k_j)). \quad (6)$$

Accordingly, by combining the three ranking methods, we propose Eq. (7) to effectively score node n w.r.t. an input keyword query.

$$Score(n, \mathscr{K}) = \sum_{1 \leqslant i < j \leqslant m} Score_{\mathsf{KP}}(n, \langle k_i, k_j \rangle) + \sum_{l=1}^{m} Score_{\mathsf{SK}}(n, k_l). \tag{7}$$

For instance, consider the query "IR John Smith" in Example 2. We have

$$Score_{\mathsf{KP}}(n_{23}, \langle \mathtt{John}, \mathtt{Smith} \rangle) = 0.8^0 * (1.24 + 1.24) = 2.48;$$
$$Score_{\mathsf{KP}}(n_{15}, \langle \mathtt{John}, \mathtt{Smith} \rangle) = 0.8^2 * (1.24 + 1.24) = 1.58.$$

The keyword-pair-based score of node 23 (using the third ranking method) is larger than that of node 15, since the three input keywords are more relevant to the subtree rooted at node 23. Thus, the keyword-pair-based ranking method can rank the minimal-cost tree rooted at node 23 as the most relevant answer.

3. Efficient indexing for computing answers

In this section, we study how to build an index structure to compute answers to keyword queries, as defined in the previous section. We focus on answering top-k queries using Eq. (7) as the ranking measure.

3.1. Indexing construction

We develop an efficient stack-based method to identify the pivotal paths using a single-pass over the XML document. We traverse the XML document tree in pre-order. When traversing the document, we maintain a stack to keep the paths from the current node to the document root. Each element in the stack is the child of the element directly below it, where the stack grows down-top. We maintain a hash table for each element in the stack, which is used to keep the terms[2] included in the element and the corresponding pivotal paths. The entries of the hash table w.r.t. a node n, denoted by $\mathscr{H}_n$, are the terms included in the node n or n's descendants, and the values are their corresponding pivotal paths, as illustrated by Fig. 2a.

During the traversal, consider the case where we meet the start tag of a node n. Let t be the label of node n or a term in the text of n. We insert $\langle t, n \rangle$ into the hash table of n (i.e., $\mathscr{H}_n$), and push node n with $\mathscr{H}_n$ into the stack. When we see the end tag of the node n, we pop the top element from the stack. For each term t in $\mathscr{H}_n$, if t is not in the hash table of the current top element n_{top} (i.e., $\mathscr{H}_{n_{top}}$), we insert $\langle t, n_{top} \cdot \mathscr{H}_n(t) \rangle$ into $\mathscr{H}_{n_{top}}$, where $\mathscr{H}_n(t)$ is the hash value of t in $\mathscr{H}_n$ (i.e., the path from n to $\kappa_{(n,t)}$), and $n_{top} \cdot \mathscr{H}_n(t)$ denotes the path by concatenating n_{top} with $\mathscr{H}_n(t)$; otherwise, if the length of term t to node n_{top} is the same as that of the current pivotal path w.r.t. $\mathscr{H}_{n_{top}}(t)$, there should be multiple pivotal nodes, and thus we update $\mathscr{H}_{n_{top}}(t)$ by inserting $n_{top} \cdot \mathscr{H}_n(t)$.

Example 3. Consider the XML document in Fig. 1. During the traversal, after visiting the nodes $\langle \mathtt{bib} \rangle(n_1), \langle \mathtt{conf} \rangle(n_2)$, and $\langle \mathtt{name} \rangle(n_3)$, we have $\langle \mathtt{bib}, \mathtt{bib} \rangle$ in $\mathscr{H}_{\mathtt{bib}}$, $\langle \mathtt{conf}, \mathtt{conf} \rangle$ in $\mathscr{H}_{\mathtt{conf}}$, $\langle \mathtt{WWW}, \mathtt{name} \rangle$ and $\langle \mathtt{name}, \mathtt{name} \rangle$ in $\mathscr{H}_{\mathtt{name}}$. These three elements were pushed into the stack, as shown in Fig. 2b. When we see $\langle /\mathtt{name} \rangle(n_3)$ during the traversal, we put $\langle \mathtt{WWW}, \mathtt{conf} \cdot \mathtt{name} \rangle$ and $\langle \mathtt{name}, \mathtt{conf} \cdot \mathtt{name} \rangle$ into $\mathscr{H}_{\mathtt{conf}}$, as shown in Fig. 2c, because name and WWW are not contained in $\mathscr{H}_{conf}$. In general, we compute the pivotal paths after traversing the XML document once.

Then, based on the constructed hash tables, we compute the three scores as follows. First, we compute tf, idf, and ntl, and compute $Score_1$ according to Eq. (1). Then, for each hash table $\mathscr{H}_n$ and each of its entries t, we compute $Score_2(n, t)$ based on the pivotal paths in $\mathscr{H}_n(t)$ according to Eq. (3). Finally, for any two entries k_i and k_j in $\mathscr{H}_n$, we compute $Rel(\langle k_i, k_t \rangle | n)$ based on the pivotal paths $\mathscr{H}_n(k_i)$ and $\mathscr{H}_n(k_j)$ according to Eq. (5). Accordingly, we can compute the three scores.

Next, we devise two structure-aware indices to maintain the pivotal paths and scores. The first one, called "EI-Index", is an inverted index extended to maintain single keyword scores. Each entry of the EI-Index is a term t, and the value is a set of triples $\langle n, p, s \rangle$, where n is a quasi content node or content node for t, p is the corresponding pivotal path, and s is $Score_{\mathsf{SK}}(n, t)$. For example, given the XML document in Fig. 1, the corresponding EI-Index is illustrated in Table 1. The second index structure, called "KP-Index," is used to compute keyword-pair-based scores. Each entry in the index is a keyword-pair with the corresponding nodes that contain the two keywords and their scores, i.e., $Score_{\mathsf{KP}}(n, \langle k_i, k_j \rangle)$. The KP-Index of the XML document in Fig. 1 is illustrated in Table 2.

3.2. Answering queries using indices

Given a keyword query $\mathscr{K} = \{k_1, k_2, \ldots, k_m\}$, we focus on finding the top-k relevant answers based on the scoring function in Eq. (7). There are many threshold-based techniques, such as Fagin's Algorithm (FA) [11], the Threshold Algorithm (TA) [12,3,22,45,13], to progressively identify the top-k answers with the highes scores. We use the TA algorithm in this paper. We first retrieve each single keyword list from the EI-Index and each keyword-pair list from the KP-Index. Then, we identify

[2] In this paper, "keyword" denotes a user input keyword, and "term" denotes a keyword tokenized in an XML document.

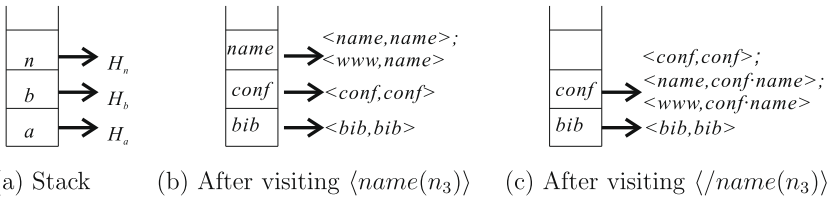

Fig. 2. Stack with a hash table for each element.

Table 1
EI-Index.

Term	[Node, pivotal path, score]
IR	$[n_6, n_6, 1.35]; [n_{16}, n_{16}, 1.35]; [n_{24}, n_{24}, 1.35]; [n_5, n_5{-}n_6, 1.08]; [n_{15}, n_{15}{-}n_{16}, 1.08]; [n_{23}, n_{23}{-}n_{24}, 1.08] \cdots$
XML	$[n_6, n_6, 1.55]; [n_{24}, n_{24}, 1.55]; [n_5, n_5{-}n_6, 1.24]; [n_{23}, n_{23}{-}n_{24}, 1.24]; \cdots$
John	$[n_{18}, n_{18}, 1.55]; [n_{25}, n_{25}, 1.55]; [n_{15}, n_{15}{-}n_{18}, 1.24]; [n_{23}, n_{23}{-}n_{25}, 1.24]; \cdots$
Smith	$[n_{17}, n_{17}, 1.55]; [n_{25}, n_{25}, 1.55]; [n_{15}, n_{15}{-}n_{17}, 1.24]; [n_{23}, n_{23}{-}n_{25}, 1.24]; \cdots$
...	...

Table 2
KP-Index.

Keyword-pair	[Node, score]
⟨IR, XML⟩	$[n_6, 2.90]; [n_{24}, 2.90]; [n_5, 2.32]; [n_{23}, 2.32]; \cdots$
⟨John, Smith⟩	$[n_{25}, 3.10]; [n_{23}, 2.48]; [n_{20}, 1.98]; [n_{15}, 1.58]; \cdots$
⟨IR, John⟩	$[n_{15}, 1.48]; [n_{23}, 1.48]; \cdots$
⟨IR, Smith⟩	$[n_{15}, 1.48]; [n_{23}, 1.48]; \cdots$
⟨XML, John⟩	$[n_{23}, 1.59]; [n_{20}, 1.27]; \cdots$
⟨XML, Smith⟩	$[n_{23}, 1.59]; [n_{20}, 1.27]; \cdots$
...	...

the top-k relevant nodes from the lists. Next, we construct the minimal-cost trees rooted at the identified nodes based on the pivotal paths in the EI-Index. In addition, note that we can efficiently identify all the results in terms of the "AND" predicate as follows. We use the indexed lookup eager algorithm (IL) in [45] to compute the intersection of inverted lists (note that the lists should be sorted by the node ids). We scan every node in the shortest list (with minimal number of nodes), and check whether each node appears in other lists (using binary-search or hash based methods). Different from the IL algorithm, for each node v in the shortest list, we need not find the right match and the left match of node v. Instead, we only need to check whether v appears in every inverted list, which is more efficient than the IL algorithm.

Example 4. Consider a keyword query "`XML IR John`" over the XML document in Fig. 1. We first get the top-1 relevant node, i.e., n_{23}, using the threshold-based method on the EI-Index and the KP-Index. Then, we construct the minimal-cost tree rooted at n_{23} based on the pivotal paths in the EI-Index, i.e., the subtree rooted n_{23} and containing paths: $n_{23} \rightarrow n_{24}$, and $n_{23} \rightarrow n_{25}$. Note that existing methods use their semantics (such as exclusive LCA [15], meaningful LCA [34], and smallest LCA [50]) to identify query results. Instead, we adopt the ranking-based techniques to identify the most relevant answers.

4. Schema-aware dewey code

To reduce the size of the EI-Index, we propose another index, which only maintains the pivotal nodes, as opposed to the pivotal paths. Moreover, we can efficiently deduce the pivotal paths based on the pivotal nodes. To achieve our goal, we introduce a novel numbering scheme, called *schema-aware dewey code* (SADC for short), which is inspired by the dewey code in [25,39,48]. The dewey code assigns to each node a vector that represents the path from the document's root to the node. The dewey code captures sibling order relationships. However, we cannot deduce a label from a dewey code. Instead, our proposed schema-aware dewey code captures schema information, and we can easily deduce the label of a node and its ancestor's labels based on the schema-aware dewey code.

Generally, given an XML document, there is a corresponding DTD (or schema) associated with it, which describes the document type definition. A DTD is typically much smaller than its corresponding XML document, and therefore it is much easier to manipulate. Even if there does not exist a DTD, we can extract one from the XML document[52,7]. For example, in Fig. 3, (a) is an XML document and (b) is the DTD extracted from this XML document.

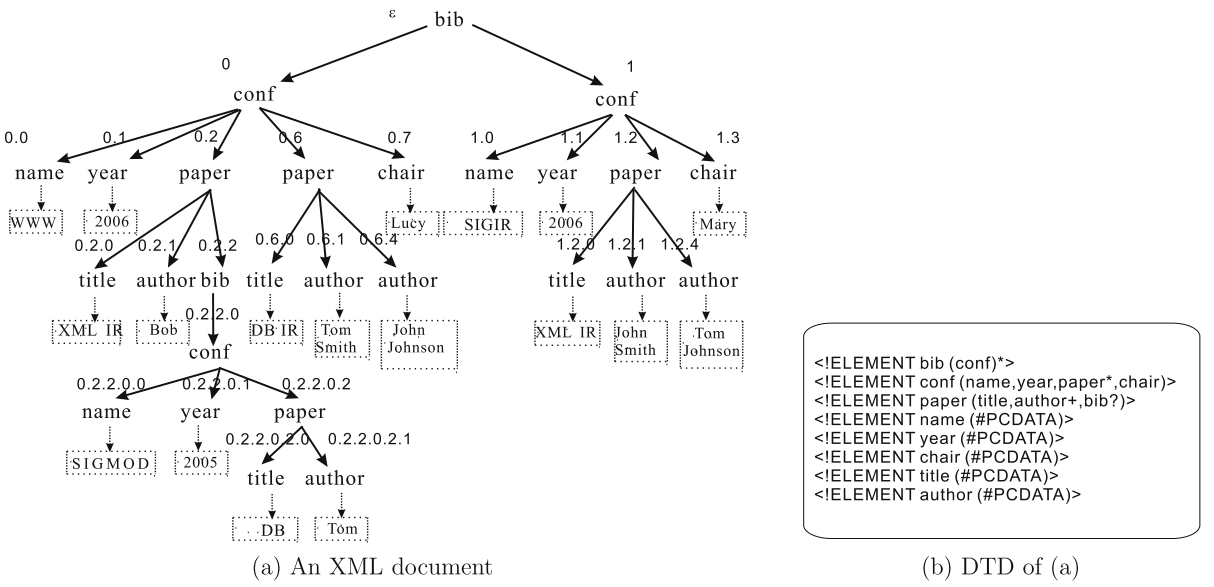

Fig. 3. An example XML document and its DTD.

Given an XML document, we encode its nodes based on the corresponding DTD. Let $parent(n)$ denote the parent of n and $presib(n)$ denote the preceding sibling (neighboring) of n. Suppose the schema-aware dewey code of the root is ϵ and $\mathscr{C}_n$ denotes the schema-aware dewey code of node n. We can encode each node from the root to the leaf as follows: for any node n, its schema-aware dewey code is its parent's schema-aware dewey code, $\mathscr{C}_{parent(n)}$, concatenated with an assigned and ordered number $\mathscr{O}_n$, i.e., $\mathscr{C}_n = \mathscr{C}_{parent(n)} \circ \mathscr{O}_n$. Without loss of generality, let $l_0, l_1, \ldots, l_{m-1}$ denote m distinct labels contained among the children of $parent(n)$ in DTD[3] and suppose $\lambda(n)$ is the kth $(0 \leqslant k \leqslant m-1)$ label. We can compute $\mathscr{O}_n$ and $\mathscr{C}_n$ through Eqs. (8) and (9), respectively. It is obvious that $\mathscr{O}_n \% m = k$, that is, the siblings with the same label will get the same remainder when divided by the total number of distinct labels among their siblings. The schema-aware dewey code captures the following salient features:

(i) Node a is an ancestor of node d, iff, $\mathscr{C}_a$ is a prefix of $\mathscr{C}_d$. a is the parent of d, iff, $\mathscr{C}_a$ is a prefix of $\mathscr{C}_d$ and $|\mathscr{C}_a| = |\mathscr{C}_d| - 1$, where $|\mathscr{C}_v|$ denotes the length of $\mathscr{C}_v$, i.e., the depth of node v in the XML document tree.

(ii) Node a follows (or precedes) node b iff $\mathscr{C}_a$ is greater (or smaller) than $\mathscr{C}_d$ in lexicographical order.

$$\mathscr{O}_n = \begin{cases} k & \text{if } n \text{ is the first child of parent}(n), \\ \mathscr{O}_{presib(n)} + k - \mathscr{O}_{presib(n)} \% m & \text{else if } \mathscr{O}_{presib(n)} \% m < k, \\ \mathscr{O}_{presib(n)} + m + k - \mathscr{O}_{presib(n)} \% m & \text{otherwise}, \end{cases} \tag{8}$$

$$\mathscr{C}_n = \begin{cases} \epsilon & \text{if } n \text{ is the root node}, \\ \mathscr{C}_{parent(n)} \circ \mathscr{O}_n{}^4 & \text{otherwise}. \end{cases} \tag{9}$$

(iii) Given the schema-aware dewey code of a node, we can deduce its ancestors' schema-aware dewey codes and the corresponding labels based on the numbering scheme.

(i) and (ii) are obvious according to the encoding strategy. Based on the two properties, we can efficiently compute the LCA of two or more nodes. (iii) is the salient feature of the schema-aware dewey code, which the general dewey numbering schemes do not capture. Given the schema-aware dewey code of node n, we demonstrate how to infer the labels of the ancestors of n as follows. As the schema-aware dewey code of the root is always ϵ, we deduce the labels iteratively from the root, and here we only introduce how to infer the label of a given node according to its schema-aware dewey code and its parent's label that has already been deduced. Consider that the schema-aware dewey code of node n is $\mathscr{C}_{\text{parent}(n)} \circ \mathscr{O}_n$, and its parent's label $\lambda(\text{parent}(n))$ has been obtained through iteration. Suppose the distinct labels of parent(n)'s children are, in order, $l_0, l_1, \ldots, l_{m-1}$, which can be gotten from the corresponding DTD. We can compute the order

[3] For the "ANY element" in DTD, we need enumerate and preserve all the parsable sub-elements under it.

[4] a ∘ b denotes another code constructed by concatenating a and b with a delimiter (e.g., a dot) between them.

Table 3
SADC-EI-Index.

Term	[Node, SADC, score]
IR	[0.2.0, 0.2.0, 1.35]; [0.6.0, 0.6.0, 1.35]; [1.2.0, 1.2.0, 1.35]; [0.2, 0.2.0, 1.08]; ···
XML	[0.2.0, 0.2.0, 1.55]; [1.2.0, 1.2.0, 1.55]; [0.2, 0.2.0, 1.24]; [1.2, 1.2.0, 1.24]; ···
John	[0.6.4, 0.6.4, 1.55]; [1.2.1, 1.2.1, 1.55]; [0.6, 0.6.4, 1.24]; [1.2, 1.2.1, 1.24]; ···
Smith	[0.6.1, 0.6.1, 1.55]; [1.2.1, 1.2.1, 1.55]; [0.6, 0.6.1, 1.24]; [1.2, 1.2.1, 1.24]; ···
···	···

Table 4
SADC-KP-Index.

Keyword-pair	[Node, score]
⟨IR, XML⟩	[0.2.0, 2.90]; [1.2.0, 2.90]; [0.2, 2.32]; [1.2, 2.32];···
⟨John, Smith⟩	[1.2.1, 3.10]; [1.2, 2.48]; [1, 1.98]; [0.6, 1.58];···
⟨IR, John⟩	[0.6, 1.48]; [1.2, 1.48]; ···
⟨IR, Smith⟩	[0.6, 1.48]; [1.2, 1.48]; ···
⟨XML, John⟩	[1.2, 1.59]; [1, 1.27]; ···
⟨XML, Smith⟩	[1.2, 1.59]; [1, 1.27]; ···
···	·····

of $\lambda(n)$ among these labels, i.e., $\mathscr{O}_n \% m$, and accordingly get its label, i.e., $l_{\mathscr{O}_n \% m}$. Hence, given the schema-aware dewey code of a node, we can deduce the labels of its ancestors from the root to itself iteratively.

Based on this salient feature of the schema-aware dewey code, we only need to maintain the pivotal nodes in the EI-Index, instead of the pivotal paths. The pivotal paths can be deduced based on the schema-aware dewey code. Accordingly, we devise another two structure-aware indices, the schema-aware dewey code based EI-Index (SADC-EI-Index) and the schema-aware dewey code based KP-Index (SADC-KP-Index). The difference between the EI-Index and the SADC-EI-Index is that the former maintains the pivotal paths while the latter preserves the schema-aware dewey codes of pivotal nodes. Tables 3 and 4, respectively, illustrate the SADC-EI-Index and the SADC-KP-Index of the XML document in Fig. 3, where underlined numbers denote the corresponding scores. To better understand the key feature of the SADC-EI-Index, we walk through an example to describe how to compute schema-aware dewey codes and how to deduce the labels based on the schema-aware dewey codes as follows.

Example 5. Consider the XML document in Fig. 3a and its DTD in Fig. 3b. The schema-aware dewey code of `conf` (the first child of the root node(`bib`)) is 0 according to Eq. (9). Node `conf`(0) has four child labels, `name`, `year`, `paper`, `chair`, thus $m = 4$. As `name` is the first child of `conf`(0), $k = 0$ and $\mathscr{O}_{name} = 0$ according to Eq. (8), thus $\mathscr{C}_{name} = 0.0$. As `year` is the second child label of `conf`(0), $k = 1$. As $\mathscr{O}_{name} \% m = 0 < k, \mathscr{O}_{year} = \mathscr{O}_{name} + k - \mathscr{O}_{name} \% m = 1$, thus $\mathscr{C}_{year} = 0.1$. Similarly, $\mathscr{O}_{paper^1} = 2$[5] and $\mathscr{C}_{paper^1} = 0.2$. As $\mathscr{O}_{paper^1} \% m = k = 2, \mathscr{O}_{paper^2} = \mathscr{O}_{paper^1} + m + k - \mathscr{O}_{paper^1} \% m = 6$ based on Eq. (8). Thus, $\mathscr{C}_{paper^2} = 0.6$. Accordingly, we can encode the nodes in the document in Fig. 3a as shown in Tables 3 and 4.

Given a schema-aware dewey code 0.6.1, its ancestors' schema-aware dewey codes are, respectively, $\epsilon, 0, 0.6$. Suppose the level of the root is 0. Let $|0.6|$ denote the level of node 0.6 in the XML document. We have $|0.6| = 2$ and $|0.6.1| = 3$. As $|0.6| = |0.6.1| - 1, 0.6$ is the parent of 0.6.1. We can deduce the labels of 0.6.1's ancestors based on the DTD. As the root is `bib`, the label of 0.6.1's ancestor at level 0 is `bib`. `bib` has only one child label, i.e., `conf`, in the DTD. As the assigned and ordered number of 0.6.1's ancestor at level 1 ($\mathscr{O}_1$) is 0, and $\mathscr{O}_1 \% 1 = 0$, so the ancestor of 0.6.1 at level 1 is `conf`. Node `conf`(0) has four ($m = 4$) distinct child labels, in order `name`, `year`, `paper`, `chair`, according to the DTD. Since the assigned and ordered number of 0.6.1's ancestor at level 2 is 6, and $\mathscr{O}_2 \% m = 6 \% 4 = 2$, so the ancestor of 0.6.1 at level 2 is `paper`. In the same way, as `paper` has three distinct child labels, in order `title`, `author`, `bib`, and $\mathscr{O}_3 \% m = 1 \% 3 = 1$, so the label of 0.6.1 is `author`. Therefore, the labels of 0.6.1's ancestors from the root to itself, are `bib`, `conf`, `paper`, `author`. Given node `conf`(0) and keyword "Tom", their pivotal node is `author`(0.6.1). We only need to maintain the pivotal node (0.6.1). Moreover, the pivotal path conf-paper-author can be deduced according to the schema-aware dewey code.

Note that the schema-aware dewey code can reduce the index size as (1) suppose there are n nodes in the XML document. The EI-Index needs $log(n)$ bits to encode each node by assigning an integer; while the SADC-EI-Index employs a relative encoding scheme (encoding a node among the children of its parent node), which can reduce the space as discussed in [41]; and (2) the EI-Index needs to keep the pivotal paths[6]; while the SADC-EI-Index only maintains the schema-aware dewey code and the DTD.

[5] *paper*[1] refers to the first `paper` of `conf`(0); *paper*[2] refers to the second `paper`.
[6] Even if the EI-Index maintains nodes's IDs in the pivotal paths, it needs to keep a set of mappings from ids to labels.

Table 5
Queries (a part) employed in the experiments.

Query ID	Queries
(a) DBLP dataset	
Q_1	IR Database
Q_2	DB IR XML
Q_3	XML Keyword Search Li
Q_4	XML Relational Keyword Search Yu
Q_5	Amer-Yahia DB IR XML 2006
(b) SIGMOD Record dataset	
Q_6	XML IR
Q_7	Database IR XML
Q_8	Xu Yu XML Search
Q_9	Lin Guo Search XML IR
Q_{10}	Data Mining Han 2002 VLDB
(c) TreeBank dataset	
Q_{11}	NP EMPTY
Q_{12}	IN PP CC
Q_{13}	ADJ NNS NP DT
Q_{14}	VB VP SBAP NN DT
Q_{15}	VBN VBG CC NNS NN PP
(d) XMark dataset	
Q_{16}	Shipping charges
Q_{17}	Money order payment
Q_{18}	Listitem admiration pledges date
Q_{19}	Accompany pardon itemref initial bidder
Q_{20}	Ambitious impotent flourish parlist emph

5. Experimental study

We have designed and performed a comprehensive set of experiments to evaluate the performance of our proposed algorithm. We compared SAIL with state-of-the-art algorithms, XRank [15], smallest LCA (SLCA) [50] and Meaningful LCA (MSLCA) [47]. For XRank, we implemented the ElemRank computation and the hybrid Dewey inverted list (HDIL) algorithm. The inverted lists were implemented using Lucene,[7] and we built a rmB^+-tree over the inverted lists. We set the parameters for ElemRank as in [15], e.g., $d_1 = 0.35, d_2 = 0.25, d_3 = 0.25$, and the convergence threshold is 0.00002. For SLCA, we implemented the indexed lookup eager algorithm which is the best one among the three algorithms proposed in [50].

We used both real and synthetic datasets. The synthetic dataset was generated using the XMark benchmark[8] with a factor of 1.0 and the raw file was about 115 MB. We also employed the real datasets DBLP,[9] SIGMOD Record,[10] and TreeBank dataset[11] from the Washington XML Data Repository to explore the performance of various algorithms. The sizes of DBLP, SIGMOD Record and TreeBank were about 420 MB, 700 KB and 82 MB, respectively. To better understand the performance of our algorithms, we selected various keyword queries with different selectivities. For each dataset, we selected one hundred keyword queries consisting of two to six keywords each. Some of the selected queries are illustrated in Table 5. To evaluate the effectiveness of our algorithm, we also used the INEX dataset (Section 5.2).[12]

We employed four metrics, elapsed time, precision, recall and $\mathscr{F}$*-measure* to evaluate the efficiency and effectiveness of these algorithms. To compute precision and recall, we reformulated the keyword queries into schema-aware XQuery queries according to the schemas of the datasets as discussed in [34], took the results of these transformed queries as the accurate answers, and then computed precision and recall of given queries as follows. Given a keyword query $\mathscr{K}$ and its corresponding transformed XQuery $\mathscr{X}$, the accurate result set of $\mathscr{K}$, i.e., the answer of $\mathscr{X}$, is denoted as AR,[13] and the approximate result set, i.e., the result of an algorithm on $\mathscr{K}$, is denoted as PR. PR is composed of subtrees, path trees, or minimal-cost trees for different algorithms [36]. If two trees are with the same root, we take them as the same result. Accordingly, we can define the precision and recall of an algorithm as follows. Precision of an algorithm is the ratio between $|\mathtt{AR} \cap \mathtt{PR}|$ and $|\mathtt{PR}|$. Recall is the ratio between $|\mathtt{AR} \cap \mathtt{PR}|$ and $|\mathtt{AR}|$, where $\mathtt{AR} \cap \mathtt{PR}$ denotes the set of trees that are in both AR and PR, and $|\mathtt{PR}|$ denotes the number of trees in PR. Let $\mathscr{F}, \mathscr{P}$ and $\mathscr{R}$ denote $\mathscr{F}$*-measure*, precision and recall of an algorithm respectively. If $\mathscr{P} \neq 0$ and $\mathscr{R} \neq 0$, $\mathscr{F} = \frac{2*\mathscr{P}*\mathscr{R}}{\mathscr{P}+\mathscr{R}}$; otherwise, $\mathscr{F} = 0$.

[7] http://lucene.apache.org/java/docs/index.html.
[8] http://www.xml-benchmark.org/.
[9] http://dblp.uni-trier.de/xml/.
[10] http://www.sigmod.org/record/xml/.
[11] http://www.cs.washington.edu/research/xmldatasets/.
[12] http://inex.is.informatik.uni-duisburg.de.
[13] To compute all the accurate results, we consider the "AND" predicate. For the top-k keyword queries, we use the "OR" predicate.

The experiments were conducted on an Intel(R) Pentium(R) 2.0@2GHz computer with 2GB RAM running Windows XP. The algorithms were implemented in Java and the parsing of the XML files was performed using the SAX API of the Xerces Java Parser.

5.1. Search quality

This section evaluates the quality of a search technique in terms of accuracy and completeness using the standard metrics, precision, recall and $\mathscr{F}$-*measure*, borrowed from the IR literature, where the correct results are the answers returned by the corresponding schema-aware languages. Precision measures accuracy, indicating the fraction of results in the approximate answer that are correct; while recall measures completeness, indicating the fraction of all correct results actually captured in the approximate answer.

We used the selected one hundred keyword queries, performed the selected algorithms on them, and then computed the corresponding precision, recall and $\mathscr{F}$-*measure*. We set α to 0.8 as a trade-off in the experiments and SAIL achieves the best performance at this point. This is so because, it will degrade the importance of ancestor nodes for a smaller α and thus may miss some meaningful and relevant results; on the contrary, it will involve many duplicates and less important results for a lager α. As MSLCA mainly improves search efficiency of SLCA and there is no distinct difference between SLCA and MSLCA in terms of search quality, we only report the experimental results of MSLCA in terms of precision and recall in the remainder of this paper.

5.1.1. Search accuracy

We first evaluated the precision of the selected algorithms with different numbers of input keywords and Fig. 4 shows the experimental results obtained. Since SLCA and MSLCA may cause false positives as they will prune some LCAs, they will lead to low search accuracy. In addition, XRank and MSLCA do not consider the structural relevance, and they may involve false positives by integrating irrelevant nodes as the query results and thus lead to low precision. Alternatively, SAIL(EI+KP-Index) achieves much higher precision than other approaches, because it employs better ranking methods based on novel techniques using link-based relevance ranking and keyword-pair ranking to capture the structural information embedded in XML documents.

We then compared the top-k answer relevance of different algorithms by varying the values of k. The top-k answer relevance measures the ratio of the number of relevant answers among the first k answers with the highest scores to k. The experimental results are illustrated in Fig. 5. As MSLCA does not provide a ranking mechanism, we rank them by using traditional tf*idf based IR-style ranking methods as described in Eq. (2). We observe that SAIL(EI+KP-Index) achieves much better top-k precision than state-of-the-art methods, and even leads to 10–40% over XRank and MSLCA. This is because our ranking method is much more effective. Moreover, SAIL(EI+KP-Index) outperforms SAIL(EI-Index), which reflects the benefits of our proposed keyword-pair ranking mechanism.

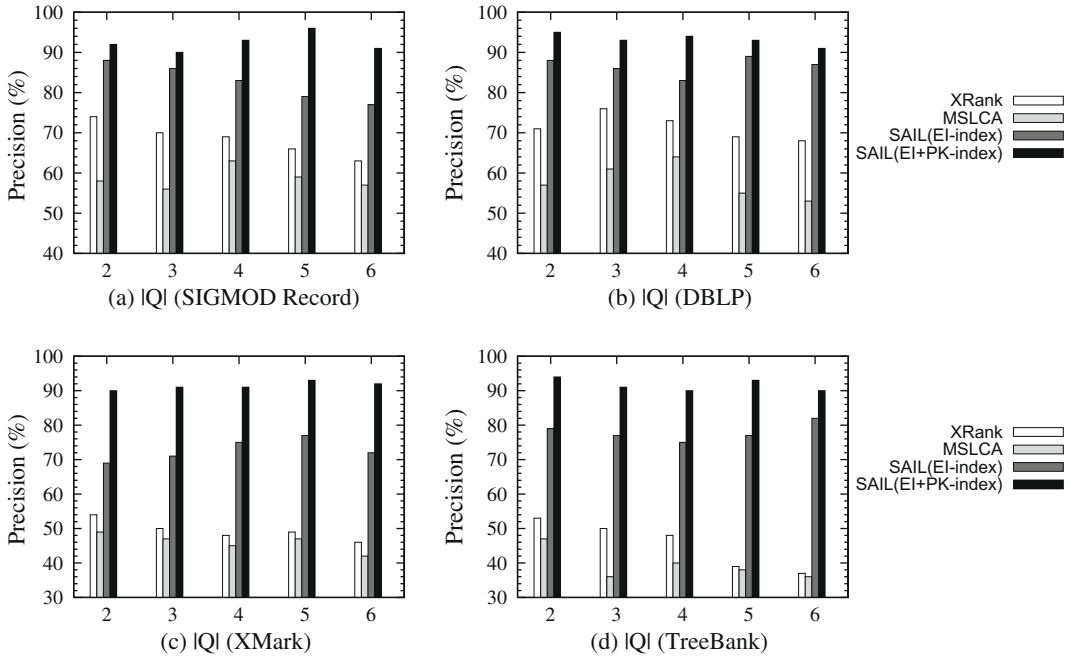

Fig. 4. Search accuracy for different numbers of input keywords.

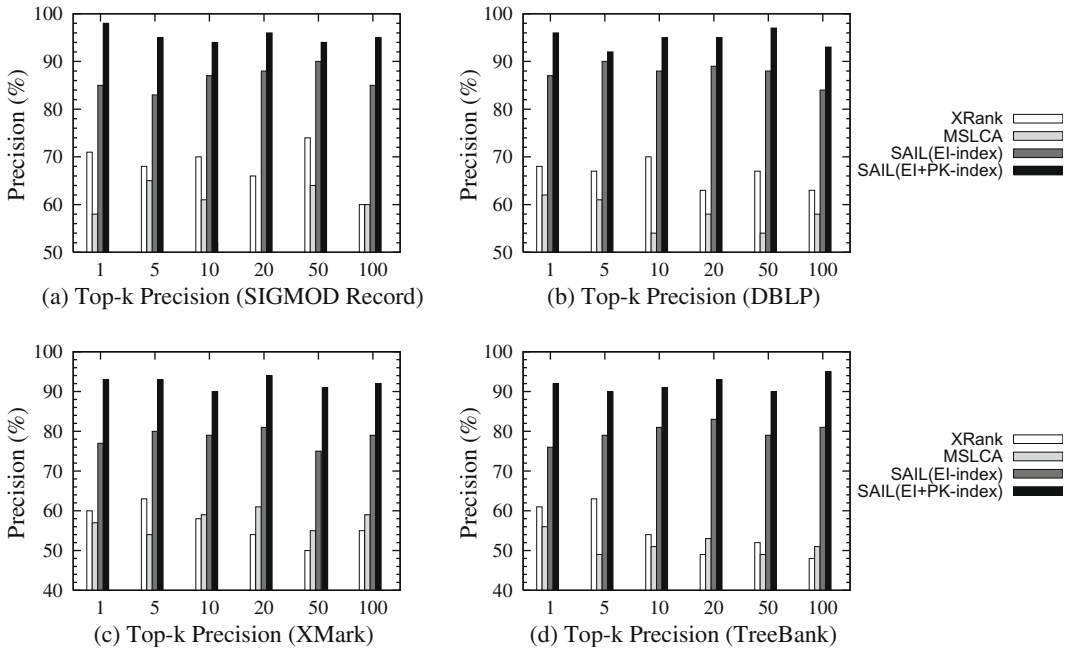

Fig. 5. Top-k precision for different values of k.

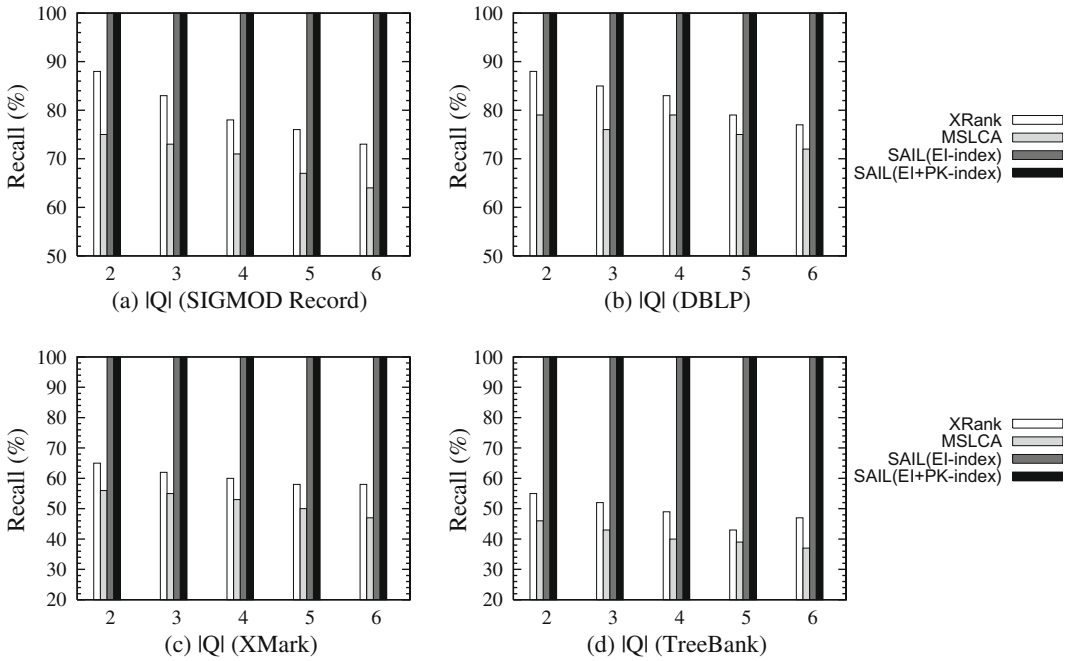

Fig. 6. Search completeness for different numbers of input keywords.

5.1.2. Search completeness

In this section we compare the search completeness of different algorithms, which indicates the fraction of all correct results actually captured in the approximate answer. Fig. 6 illustrates the search completeness with different numbers of input keywords. Table 6 shows the average recall among all the selected queries and Fig. 7 describes the 11-pt precision/recall curves for all the queries. We can see SAIL achieves much higher recall, outperforming XRank and MSLCA by about 20–30%. As existing methods identify the subtrees rooted at LCAs or its variants and cause false negatives. Instead, SAIL identifies the answers ranked by the relevancy with an effective ranking mechanism. This comparison further reflects the effectiveness

Table 6
Average search completeness.

Recall (%)	SIGMOD Record	DBLP	XMark	TreeBank
XRank	84	82	76	68
MSLCA	78	73	65	54
SAIL(EI-index)	100	100	100	100
SAIL(EI+KP-index)	100	100	100	100

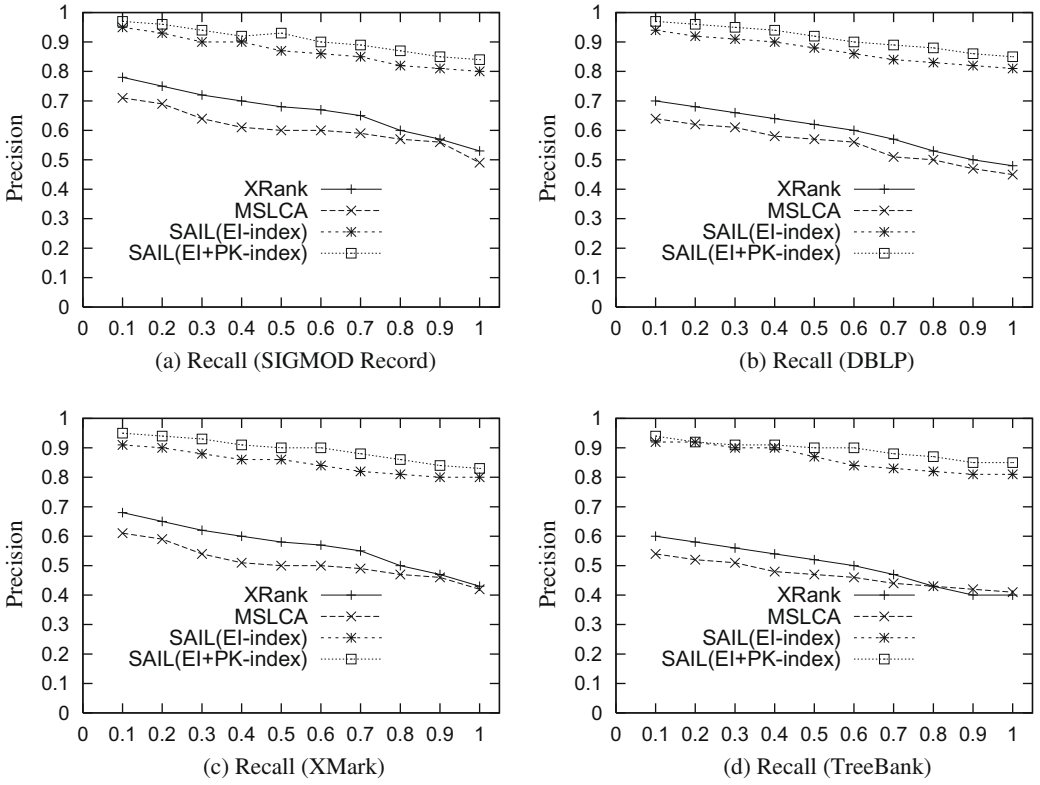

Fig. 7. Recall-precision curves.

of our proposed method. Moreover, in Fig. 7, we observe that SAIL outperforms the existing methods, and always achieves higher precision than the other methods. The precision of other methods falls sharply with the increase of recall, while that of SAIL varies slightly.

5.1.3. $\mathcal{F}$-measure

To further compare the selected algorithms, we employed another good metric, the $\mathcal{F}$-*measure*. We used the selected one hundred keyword queries and compared their average $\mathcal{F}$-*measure*. We can see that SAIL beats the other algorithms and achieves the best $\mathcal{F}$-*measure* as shown in Table 7. For example, on TreeBank, the $\mathcal{F}$-*measure* of SAIL(EI+KP-Index) reaches 92%, the $\mathcal{F}$-*measure* of SAIL(EI-Index) is 81%; while those of the other ones are less than 60%, and especially that of MSLCA is only 53%.

Table 7
$\mathcal{F}$-*measure*.

$\mathcal{F}$-*measure*(%)	XRank	MSLCA	SAIL(EI-Index)	SAIL(EI+KP-Index)
SIGMOD Record	83	75	90	98
DBLP	78	71	88	97
XMark	73	66	86	95
TreeBank	59	53	81	92

5.2. Usability study

To evaluate the usability of different algorithms, we used the INEX dataset. Usability is evaluated by human judgement. The INEX corpus is composed of the full-texts, marked up in XML, consisting of 16819 articles of the IEEE Computer Society's publications from 12 magazines and 6 transactions, covering the period of 1995–2004, and totaling 735 megabytes in size. The collection has a suitably complex XML structure (192 different content models in DTD) and contains scientific articles of varying length. On average an article contains 1532 XML nodes, where the average depth of a node is 6.9.

INEX uses two graded dimensions to express relevance: exhaustivity and specificity. Exhaustivity is defined as a measure of how exhaustively an XML element discusses the topic of request, while specificity is defined as a measure of how focused the element is on the topic of request (i.e., discusses no other, irrelevant topics). Exhaustivity refers to the standard relevance criterion used in IR, whereas specificity provides a measure with respect to the size of a component as it measures the ratio of relevant to non-relevant content within an element. The combination of the two dimensions is used to identify those relevant XML elements.

The INEX collection consists of 30 CAS (content and structure) topics and 30 CO (content only) topics. Here, we focus on the CAS topics. A content and structure topic consists of a NEXI [49] query, an explanation of the requested information in plain English, and finally a narrative that describes the criteria used by the INEX assessors to determine whether an answer to the query is relevant. We issued 30 queries based on the 30 CAS topics. Fig. 8 illustrates the precision computed using the INEX-provided utilities. Fig. 9 gives the average precision-recall graph on the 30 queries. We observe that our method still achieves high precision and recall. This is because we use effective ranking techniques to identify the answers, instead of finding answers based on a restrictive semantics.

5.3. Search efficiency

We evaluated the efficiency of SAIL, XRank, SLCA and MSLCA on the SIGMOD Record, DBLP, XMark and TreeBank datasets in this section, and compared their elapsed time for various queries. We first compared the search efficiency with different numbers of keywords and the experimental results are illustrated in Fig. 10, where SAIL(EI-Index) employs the SADC-EI-Index and SAIL(EI+KP-Index) adopts both the SADC-EI-Index and the SADC-KP-Index. We observe that SAIL achieves high search efficiency and outperforms state-of-the-art methods significantly. More importantly, with the an increasing number of input keywords the search efficiency of XRank, SLCA and MSLCA degrades. This is because it is rather inefficient to discover the rich structural relationships between content nodes on the fly. SAIL can efficiently identify minimal-cost trees by

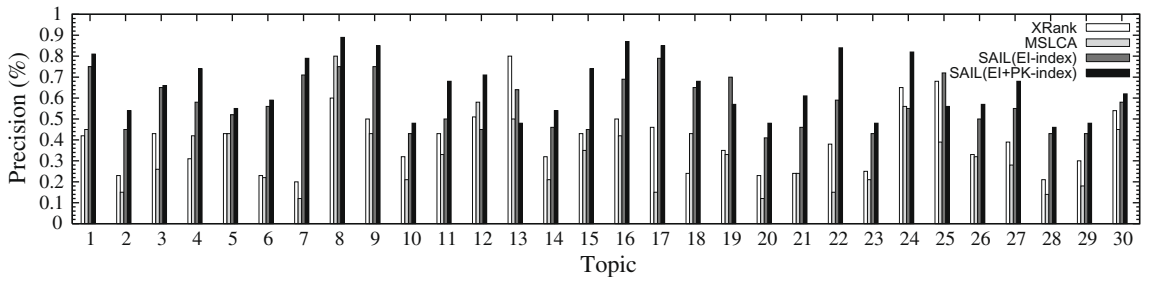

Fig. 8. Precision on INEX.

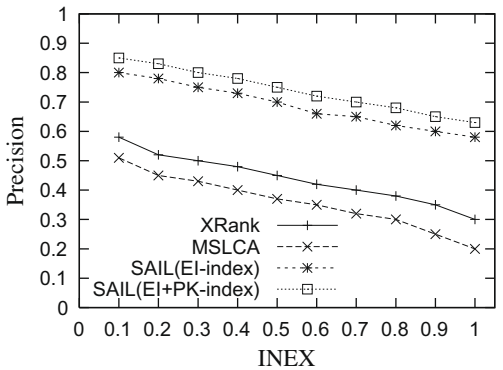

Fig. 9. Recall-precision curves on INEX.

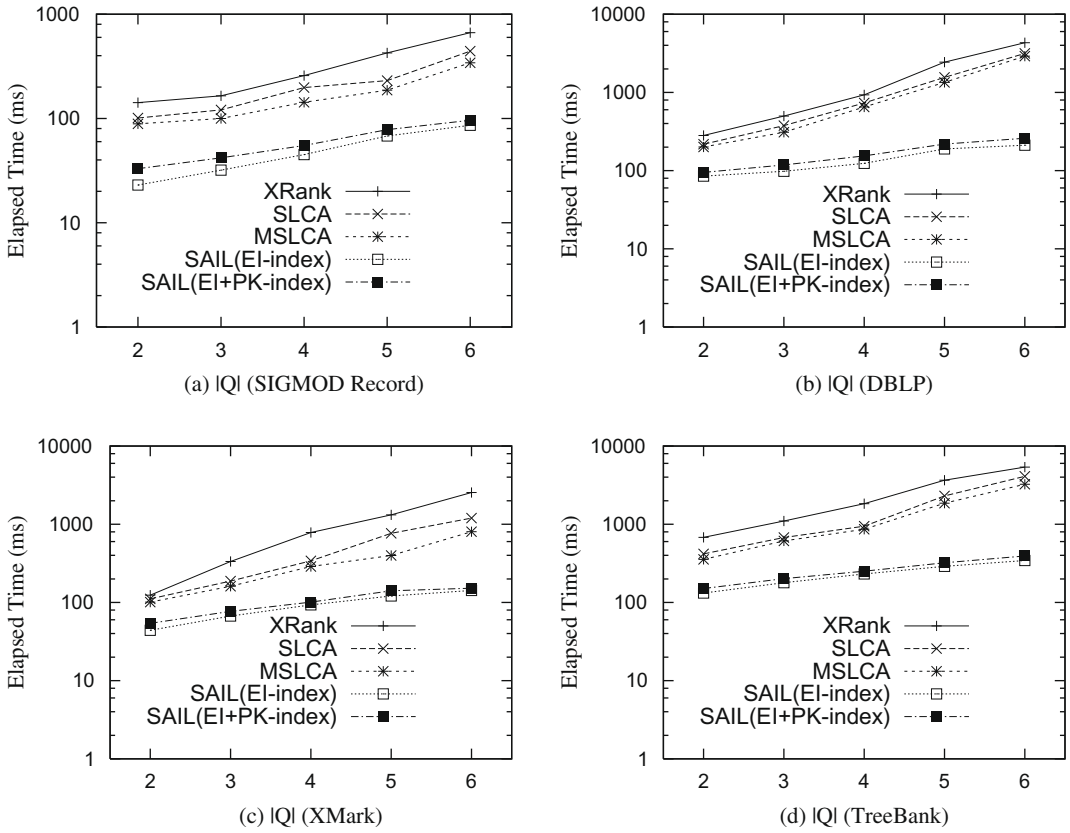

Fig. 10. Search efficiency for different numbers of input keywords.

employing the skip-based method as discussed in Section 3, and in IR literature such methods have been proven to be very effective for retrieving relevant documents based on inverted indices.

We then compared SAIL with state-of-the-art methods by identifying the top-k answers with different values of k, as users are usually interesting in the top-k results. Fig. 11 gives the experimental results obtained. Note that SAIL has a key feature that it can progressively identify the top-k answers by adopting the threshold-based techniques [12], which can efficiently retrieve the top-k answers from multiple inverted lists in a progressive way; while the other methods have to first retrieve all the relevant answers and then rank them. Accordingly, SAIL outperforms the alternative methods significantly, and even is two orders of magnitude faster than the other approaches.

5.4. Index size

In this section, we report the storage size of our proposed structure-aware indices. The experimental results are illustrated in Table 8. We observe that the size of the SADC-EI-Index is smaller than that of the EI-Index. This is because the SADC-EI-Index employs relative encoding scheme and need not maintain pivotal paths as discussed in Section 4. Moreover, the SADC-EI-Index is a bit larger than general inverted indices since it also indexes the quasi content nodes besides the content nodes. Although the SADC-KP-Index is a little larger than the SADC-EI-Index, it achieves higher search quality than the SADC-EI-index as experimentally shown in Section 5.1.

6. Related work

6.1. XML keyword search

In the literature there are different ways to define the answers to a keyword query on an XML document. A commonly used one is based on the notion of "lowest common ancestor" (LCA) [16,44]. Given an XML document $\mathscr{D}$ and some of its nodes $v_1, v_2, \ldots, v_m$, we say a node u in the documents is the LCA of these nodes if $\forall 1 \leqslant i \leqslant m$, node u is node v_i or an ancestor of v_i, and there does not exist another node u' in the document such that $u \prec u'$, and for each v_i, u' is either v_i or an ancestor of v_i. There are many LCA-based studies [15,25,50] to answer keyword queries over XML documents.

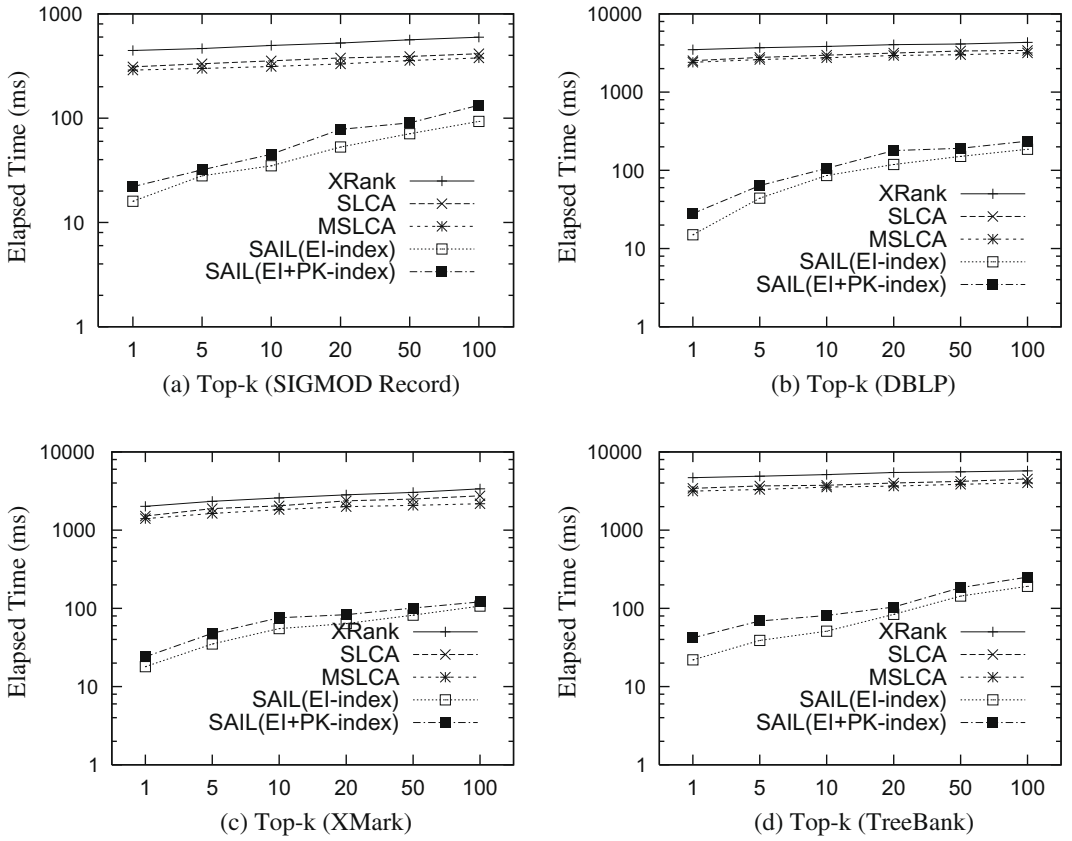

Fig. 11. Search efficiency for different values of *k*.

Table 8
Sizes of indexes.

Space (MB)	SIGMOD Record	DBLP	XMark	TreeBank
EI-Index	1.34	2642	694	641
Inverted-Index	0.18	287	94	61
SADC-EI-Index	0.21	481	136	101
SADC-KP-Index	0.36	768	198	156

To improve the answer quality, many semantics have been proposed [51,50]. Yu et al. [51] proposed the semantics of exclusive LCA (ELCA) to improve result quality. An LCA is an ELCA if it is still an LCA after excluding its LCA descendants. The intuition 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 [15]. Yu et al. [50] proposed the smallest LCA (SLCA) to answer a keyword query. The basic idea behind SLCA is that, if node v contains all the input keywords, its ancestors will be less meaningful than v. Hence, SLCA introduces the concept of *smallest tree*, which is a tree that contains all the keywords but contains no subtrees that also contain all the keywords. Clearly $SLCA(\mathscr{K}) \subseteq ELCA(\mathscr{K}) \subseteq LCA(\mathscr{K})$. The key property of SLCA search is that, given two keywords k_i and k_j and a node v that contains keyword k_i, one need not inspect the whole node list of keyword k_j in order to discover potential solutions. Instead, one only needs to find the left and right match of v in the list of k_j, where the left (right) match is the node with the greatest (least) id that is smaller (greater) than or equal to the id of v. Thus, one can skip many elements to find SLCAs and the property can be generalized to more than two keywords. Moreover, one can scan the shortest lists and find the SLCAs by binary-searching other inverted lists. Guo et al. [15] proposed XRank to improve search efficiency. Both XRank [15] and ELCA [51] employ the semantics of ELCA to answer XML keyword queries. XRank extends PageRank's ranking mechanism to XML by taking the nested structure of XML into account, and proposes two core algorithms, DIL (Dewey Inverted List) and RDIL (Ranked Dewey Inverted List), to return the top-k answers from ELCAs. The DIL algorithm sort-merges the m inverted lists of the query keywords to find the ELCAs. The RDIL algorithm in [15] introduces B^+-trees built on inverted lists sorted by Dewey ids to improve the efficiency. XRank develops a hybrid algorithm which starts using RDIL and switches to DIL when it finds out that RDIL has spent too much time

on answering the query. Multiway SLCA-based (MSLCA) keyword search in XML data [47] generalizes the SLCA query semantics to support keyword search beyond the "AND" semantics to include both "AND" and "OR" boolean operators. For AND-only SLCA query semantics, MSLCA notices that one need not completely scan the smallest keyword list for certain data instances. Instead, certain keyword instances in the smallest keyword list can be skipped for faster processing. Yu et al. [51] extended the SLCA idea to identify ELCAs. They presented an efficient algorithm, named Indexed Stack (IS) based on the ELCA semantics by combing the stack-based mechanism and the skip-based techniques introduced in [50].

Schema-Free XQuery [34] uses the idea of meaningful LCA (MLCA), and proposes a stack-based sort-merge algorithm by considering some structural properties and incorporating a new function `mlcas` into XQuery. XSeek [36] studies the problem of inferring the most relevant return nodes without elicitation of user preferences. XSeek generates two types of nodes: return nodes that can be inferred explicitly by analyzing keyword match patterns; and return nodes that can be inferred implicitly by considering both keyword match patterns and the XML data structure. Cohen et al. [8] proposed interconnection semantics to improve search quality. XSEarch [9] supports extended keyword search and focuses on the semantics and the ranking of the results. It employs the semantics of meaningfully related nodes to answer keyword queries. Two nodes are meaningfully related if they are in a pre-defined set, which can be given by administrators. It uses an all-pairs interconnection index to check whether two nodes are connected and meaningfully related. Li et al. [25] proposed valuable LCA (VLCA), to improve the meaningfulness and completeness of answers beyond XSEarch. VLCA considers the semantics of the XML document structures, which is similar to the meaningfully related semantics. VLCA proposes a more efficient algorithm to identify the answers based on a stack-based algorithm. XKeyword [21] is a system that offers keyword proximity search over XML databases. It models XML documents as graphs by considering IDREFs and is therefore more related to keyword search over graphs. Hristidis et al. [19] proposed the grouped distance minimum connecting tree (GDMCT) to answer keyword queries by grouping the relevant subtrees. Their algorithm first identifies the minimum connected tree, which is a subtree with minimum number of edges, and then groups such trees to answer keyword queries. Shao et al. [46] studied the problem of keyword search over XML views. Liu et al. [37] proposed to reason and identify relevant answers. Huang et al. [38] studies to find snippets to answer keyword queries. Li et al. [26] proposed a more effective ranking mechanism to improve search effectiveness by combining structural compactness and textual relevance to rank the answers. The ranking functions are independent of the search algorithms, and thus could be applied to any search algorithm.

Various XML full-text query languages have also been proposed [6,32,33], and the workshop INEX,[14] INitiative for the Evaluation of XML retrieval, aiming at evaluating XML retrieval effectiveness, has also been organized. More recently, two algebras for keyword search over XML documents have been proposed in [2,42]. Amer-Yahia et al. [2] presented the XFT algebra that accounts for element nesting in the XML document structure to evaluate queries with complex full-text predicates, while Pradhan et al. [42] demonstrated several optimization techniques that guarantee better efficiency for keyword search over tree-structured documents. You et al. [53] studied to rank an answers.

6.2. Keyword search over databases

In addition, the database research community has been studying keyword search over relational databases [1,4,20], graph databases [17,23], and heterogenous data sources [27,31]. The different approaches for keyword search over relational databases can be broadly classified into three categories: the methods based on the candidate network [18,20,1,40], those based on Steiner trees [4,10,17,23,35], and others based on Steiner graphs [31]. Kimelfeld et al. [24] demonstrated keyword proximity search in relational databases, which shows that the answer of keyword proximity search can be enumerated in ranked order with polynomial delay. Sayyadian et al. [43] introduced schema mapping into keyword search and proposed a method to answer keyword queries across heterogeneous databases. Guo et al. [14] proposed data topology search to retrieve meaningful structures from biological databases. Li et al. [28] proposed progressive ranking for effective keyword search over relational databases by indexing the primary–foreign-key relationships. He et al. [17] proposed the BLINK index to improve search efficiency, which employed a keyword-to-node index to facilitate identifying Steiner trees. Dalvi [5] et al. proposed a keyword search method on external memory graphs, which introduces a multi-granular graph representation to support large graphs. Li et al. [29] proposed tuple units, which are composed of the most relevant tuples in the underlying databases, to improve search effectiveness of keyword queries over relational databases. More recently, Li et al. [27] proposed a unified keyword search framework, namely SAILER, for answering keyword queries over unstructured and semi-structured data. Li et al. [31] proposed EASE to adaptively and effectively answer keyword queries over heterogenous data sources composed of unstructured, semi-structured and structured data by summarizing the graphs transformed from the heterogenous data sources. They proposed Steiner subgraphs to effectively answer keyword queries over graphs. Li et al. [30] studied type-ahead search in relational databases.

7. Conclusion

In this paper, we have investigated the problem of effective keyword search over XML documents, with the aim of identifying the most relevant subtrees to answer XML keyword queries. We proposed an effective and progressive top-k keyword

[14] http://inex.is.informatik.uni-duisburg.de/.

search method. As opposed to existing LCA-based methods, we identified and indexed the structural relationships embedded in XML documents into structure-aware indices and used the indices to improve the performance of XML keyword search. We devised several structure-aware indices for efficiently identifying the minimal-cost trees. We developed the techniques of link-based relevance ranking and keyword-pair ranking for effective keyword search over XML documents. We used the threshold-based techniques to progressively identify the top-k answers. We incorporated a numbering scheme into our structure-aware indices to reduce the index size. Our extensive experimental results show that our method achieves high search efficiency and answer quality on both synthetic and real datasets for various queries, and outperforms state-of-the-art approaches significantly.

The experimental study proves that keyword search provides an alternative way to query XML data. One important advantage of keyword search is that it enables users to search information without knowing a complex query language such as XPath or XQuery, or having prior knowledge about the structure of the underlying data and provides a user-friendly way for querying XML data.

Acknowledgements

This work is partly supported by the National Natural Science Foundation of China under Grant No. 60873065, the National High Technology Development 863 Programs of China under Grant Nos. 2007AA01Z152 and 2009AA011906, and the National Grand Fundamental Research 973 Program of China under Grant No. 2006CB303103.

References

[1] S. Agrawal, S. Chaudhuri, G. Das, Dbxplorer: a system for keyword-based search over relational databases, in: Proceedings of the 18th International Conference on Data Engineering (ICDE 2002), 2002, pp. 5–16.
[2] S. Amer-Yahia, E. Curtmola, A. Deutsch, Flexible and efficient xml search with complex full-text predicates, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2006), 2006, pp. 575–586.
[3] B. Arai, G. Das, D. Gunopulos, N. Koudas, Anytime measures for top-k algorithms, in: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB 2007), 2007, pp. 914–925.
[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, S. Sudarshan, Keyword searching and browsing in databases using banks, in: Proceedings of the 18th International Conference on Data Engineering (ICDE 2002), 2002, pp. 431–440.
[5] B. Dalvi, M. Kshirsagar, S. Sudarshan, Keyword search on external memory data graphs, in: Proceedings of the 34rd International Conference on Very Large Data Bases (VLDB 2008), 2008, pp. 1189–1204.
[6] C. Botev, S. Amer-Yahia, J. Shanmugasundaram, Expressiveness and performance of full-text search languages, in: Proceedings of the 10th International Conference on Extending Database Technology (EDBT 2006), 2006, pp. 349–367.
[7] S. Chen, M. Lo, K. Wu, J. Yih, C. Viehrig, A practical approach to extracting DTD-conforming XML documents from heterogeneous data sources, Inf. Sci. 176 (7) (2006) 820–844.
[8] S. Cohen, Y. Kanza, B. Kimelfeld, Y. Sagiv, Interconnection semantics for keyword search in xml, in: Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management (CIKM 2005), 2005, pp. 389–396.
[9] S. Cohen, J. Mamou, Y. Kanza, Y. Sagiv, Xsearch: a semantic search engine for xml, in: Proceedings of the 29th International Conference on Very Large Data Bases (VLDB 2003), 2003, pp. 45–56.
[10] B. Ding, J.X. Yu, S. Wang, L. Qin, X. Zhang, X. Lin, Finding top-k min-cost connected trees in databases, in: Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007), 2007, pp. 836–845.
[11] R. Fagin, Combining fuzzy information from multiple systems, in: Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1996), 1996, pp. 216–226.
[12] R. Fagin, Fuzzy queries in multimedia database systems, in: Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1998), 1998, pp. 1–10.
[13] R. Fagin, Optimal aggregation algorithms for middleware, in: Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2001), 2001, pp. 28–37.
[14] L. Guo, J. Shanmugasundaram, G. Yona, Topology search over biological databases, in: Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007), 2007, pp. 556–565.
[15] L. Guo, F. Shao, C. Botev, J. Shanmugasundaram, Xrank: ranked keyword search over xml documents, in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD 2003), 2003, pp. 16–27.
[16] D. Harel, R.E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM J. Comput. 13 (2) (1984) 338–355.
[17] H. He, H. Wang, J. Yang, P. Yu, Blinks: ranked keyword searches on graphs, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIDMOD 2007), 2007, pp. 305–316.
[18] V. Hristidis, L. Gravano, Y. Papakonstantinou, Efficient ir-style keyword search over relational databases, in: Proceedings of 29th International Conference on Very Large Data Bases (VLDB 2003), 2003, pp. 850–861.
[19] V. Hristidis, N. Koudas, Y. Papakonstantinou, D. Srivastava, Keyword proximity search in xml trees, IEEE Trans. Knowl. Data Eng. 18 (4) (2006) 525–539.
[20] V. Hristidis, Y. Papakonstantinou, Discover: keyword search in relational databases, in: Proceedings of 28th International Conference on Very Large Data Bases (VLDB 2002), 2002, pp. 670–681.
[21] V. Hristidis, Y. Papakonstantinou, A. Balmin, Keyword proximity search on xml graphs, in: Proceedings of the 19th International Conference on Data Engineering (ICDE 2003), 2003, pp. 367–378.
[22] M. Hua, J. Pei, A.W.C. Fu, X. Lin, H.-F. Leung, Efficiently answering top-k typicality queries on large databases, in: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB 2007), 2007, pp. 890–901.
[23] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, H. Karambelkar, Bidirectional expansion for keyword search on graph databases, in: Proceedings of the 31rd International Conference on Very Large Data Bases (VLDB 2005), 2005, pp. 505–516.
[24] B. Kimelfeld, Y. Sagiv, Finding and approximating top-k answers in keyword proximity search, in: Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2006), 2006, pp. 173–182.
[25] G. Li, J. Feng, J. Wang, L. Zhou, Efficient keyword search for valuable LCAs over xml documents, in: Proceedings of the 16th ACM Conference on Information and Knowledge Management (CIKM 2007), 2007, pp. 31–40.
[26] G. Li, J. Feng, J. Wang, L. Zhou, Race: finding and ranking compact connected trees for keyword proximity search over xml documents, in: Proceedings of the 17th International Conference on World Wide Web (WWW 2008), 2008, pp. 1045–1046.
[27] G. Li, J. Feng, J. Wang, L. Zhou, Sailer: an effective search engine for unified retrieval of heterogeneous xml and web documents, in: Proceedings of the 17th International Conference on World Wide Web (WWW 2008), 2008, pp. 1061–1062.

[28] G. Li, J. Feng, L. Zhou, Progressive ranking for efficient keyword search over relational databases, in: Proceedings of the 25th British National Conference on Databases (BNCOD 2008), 2008, pp. 193–197.
[29] G. Li, J. Feng, L. Zhou, Retrieving and materializing tuple units for effective keyword search over relational databases, in: Proceedings of the 27th International Conference on Conceptual Modeling (ER 2008), 2008, pp. 469–483.
[30] G. Li, S. Ji, C. Li, J. Deng, Efficient type-ahead search on relational data: a TASTIER approach, in: Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD 2009), 2009, pp. 695–706.
[31] G. Li, B.C. Ooi, J. Feng, J. Wang, L. Zhou, Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2008), 2008, pp. 903–914.
[32] Y. Li, H. Yang, H.V. Jagadish, Nalix: an interactive natural language interface for querying xml, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2005), 2005, pp. 900–902.
[33] Y. Li, H. Yang, H.V. Jagadish, Constructing a generic natural language interface for an xml database, in: Proceedings of the 10th International Conference on Extending Database Technology (EDBT 2006), 2006, pp. 737–754.
[34] Y. Li, C. Yu, H.V. Jagadish, Schema-free xquery, in: Proceedings of the 30th International Conference on Very Large Data Bases (VLDB 2004), 2004, pp. 72–84.
[35] F. Liu, C. Yu, W. Meng, A. Chowdhury, Effective keyword search in relational databases, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2006), 2006, pp. 563–574.
[36] Z. Liu, Y. Chen, Identifying meaningful return information for xml keyword search, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2007), 2007, pp. 329–340.
[37] Z. Liu, Y. Chen, Reasoning and identifying relevant matches for XML keyword search, in: Proceedings of the 34rd International Conference on Very Large Data Bases (VLDB 2008), 2008, pp. 921–932.
[38] Y. Huang, Z. Liu, Y. Chen, Query biased snippet generation in XML search, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2008), 2008, pp. 315–326.
[39] J. Lu, T.W. Ling, C.-Y. Chan, T. Chen, From region encoding to extended dewey: on efficient processing of xml twig pattern matching, in: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005), 2005, pp. 193–204.
[40] Y. Luo, X. Lin, W. Wang, X. Zhou, Spark: top-*k* keyword query in relational databases, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2007), 2007, pp. 115–126.
[41] Patrick O'Nei, Elizabeth O'Neil et al., ORDPATHs: insert-friendly XML node labels, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2004), 2004, pp. 903–908.
[42] S. Pradhan, An algebraic query model for effective and efficient retrieval of xml fragments, in: Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB 2006), 2006, pp. 295–306.
[43] M. Sayyadian, H. LeKhac, A. Doan, L. Gravano, Efficient keyword search across heterogeneous relational databases, in: Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007), 2007, pp. 346–355.
[44] B. Schieber, U. Vishkin, On finding lowest common ancestors: simplification and parallelization, SIAM J. Comput. 17 (6) (1988) 1253–1262.
[45] K. Schnaitter, J. Spiegel, N. Polyzotis, Depth estimation for ranking query optimization, in: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB 2007), 2007, pp. 902–913.
[46] F. Shao, L. Guo, C. Botev, A. Bhaskar, M. Chettiar, F. Yang, J. Shanmugasundaram, Efficient keyword search over virtual xml views, in: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB 2007), 2007, pp. 1057–1068.
[47] C. Sun, C.Y. Chan, A.K. Goenka, Multiway SLCA-based keyword search in xml data, in: Proceedings of the 16th International Conference on World Wide Web (WWW 2007), 2007, pp. 1043–1052.
[48] I. Tatarinov, S. Viglas, K.S. Beyer, J. Shanmugasundaram, E.J. Shekita, C. Zhang, Storing and querying ordered xml using a relational database system, in: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 2002), 2002, pp. 204–215.
[49] A. Trotman, B. Sigurbjornsson, NEXI, now and next, in: Third International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX 2004), 2004, pp. 41–53.
[50] Y. Xu, Y. Papakonstantinou, Efficient keyword search for smallest LCAs in xml databases, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2005), 2005, pp. 527–538.
[51] Y. Xu, Y. Papakonstantinou, Efficient LCA based keyword search in XML data, in: Proceedings of the 11th International Conference on Extending Database Technology (EDBT 2008), 2008, pp. 535–546.
[52] S. Yi, B. Huang, W.T. Chan, XML application schema matching using similarity measure and relaxation labeling, Inf. Sci. 169 (1–2) (2005) 527–538.
[53] G. You, S. Hwang, Search structures and algorithms for personalized ranking, Inf. Sci. 178 (20) (2008) 3925–3942.