

Processing XML Keyword Search by Constructing Effective Structured Queries

Jianxin Li, Chengfei Liu, Rui Zhou and Bo Ning
Email: {jianxinli, cliu, rzhou, bning}@groupwise.swin.edu.au

Swinburne University of Technology
Melbourne, Australia

Abstract. Recently, keyword search has attracted a great deal of attention in XML database. It is hard to directly improve the relevancy of XML keyword search because lots of keyword-matched nodes may not contribute to the results. To address this challenge, in this paper we design an adaptive XML keyword search approach, called *XBridge*, that can derive the semantics of a keyword query and generate a set of effective structured queries by analyzing the given keyword query and the schemas of XML data sources. To efficiently answer keyword query, we only need to evaluate the generated structured queries over the XML data sources with any existing XQuery search engine. In addition, we extend our approach to process top- k keyword search based on the execution plan to be proposed. The quality of the returned answers can be measured using the context of the keyword-matched nodes and the contents of the nodes together. The effectiveness and efficiency of *XBridge* is demonstrated with an experimental performance study on real XML data.

1 Introduction

Keyword search is a proven user-friendly way of querying XML data in the World Wide Web [1–5]. It allows users to find the information they are interested in without learning a complex query language or knowing the structure of the underlying data. However, the number of results for a keyword query may become very large due to the lack of clear semantic relationships among keywords. There are two main shortcomings: (1) it may become impossible for users to manually choose the interesting information from the retrieved results, and (2) computing the huge number of results with less meaning may lead to time-consuming and inefficient query evaluation. As we know, users are able to issue a structured query, such as XPath and XQuery, if they already know a lot about the query languages and the structure of the XML data to be retrieved. The desired results can be effectively and efficiently retrieved because the structured query can convey complex and precise semantic meanings. Recently, the study of query relaxation [6, 7] can also support structured queries when users cannot specify their queries precisely. Nevertheless, there are many situations where structured queries may not be applicable, such as a user may not know the data schema,

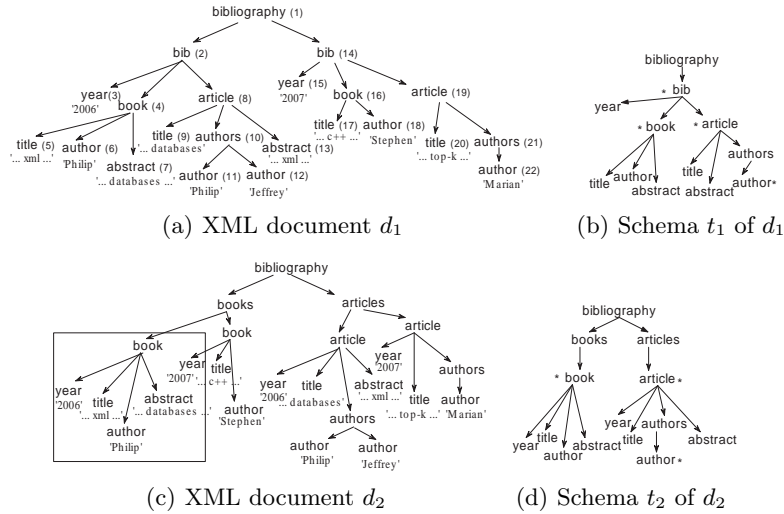


Fig. 1. XML Documents with Different Schemas

or the schema is very complex so that a query cannot be easily formulated, or a user prefers to search relevant information from different XML documents via one query.

Consider the example in Figure 1 showing the same bibliography data arranged in two different formats. In this case, a user can quickly issue a keyword query “*Philip, 2006, xml*” to obtain a list of answers. Node #4 *book* and node #8 *article* will be returned as relevant answers. From Figure 1(a), we can see that only node #4 *book* satisfies the searching requirement. For node #8 *article*, only the abstract contains the term *xml*, as such it does not meet the users’ original intention. It would be impossible for an IR-style keyword query to differentiate the semantics, e.g. one term can represent different meanings in different positions. In addition, when the size of XML documents becomes larger, it is difficult to choose the meaningful answers from the large number of returned results. As an alternative, users can construct an XQuery to represent this simple query and specify the precise context. But there are two challenges: first, they have to know that “publication” in the schema is actually presented as *book* and *article* in both schemas; second, they have to know that *title* and *author* are the child elements of “publication”, while *year* could be either a child or a sibling. Writing an accurate XQuery is non-trivial even for this simple example due to the complex structure of XML schemas. Therefore, it is highly desirable to design a new keyword search system that not only permits users specify more expressive queries, but also implement keyword search as efficiently as structured queries.

To address this problem, a formalized keyword query consisting of a set of label-term pairs is deployed in [4] and [8]. In [4], labels in the given keyword query are used to filter the node lists. In [8], labels are used to construct answer templates that includes all combinations together according to the schema of XML data stream. When the data stream is coming, all matched nodes will need to be maintained until the template-matched results are generated or the end of

<pre> For \$b in bibliography/bib For \$b2 in \$b/book Where \$b/year = '2006' and contains(\$b2/title, 'xml') and contains(\$b2/author, 'Philip') Return \$b </pre>	<pre> For \$b in bibliography/bib For \$a in \$b/article Where \$b/year = '2006' and contains(\$a/title, 'xml') and contains(\$a/authors/author, 'Philip') Return \$b </pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fig. 2. Structured Queries w.r.t. XML Schema t_1

<pre> For \$b in bibliography/books/book Where \$b/year = '2006' and contains(\$b/title, 'xml') and contains(\$b/author, 'Philip') Return \$b </pre>	<pre> For \$a in bibliography/articles/article Where \$a/year = '2006' and contains(\$a/title, 'xml') and contains(\$a/authors/author, 'Philip') Return \$a </pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fig. 3. Structured Queries w.r.t. XML Schema t_2

the stream is reached. Different from them, in this work we develop a keyword search system called *XBridge* that first infers the context of the set of labels and the required information to be returned according to XML data schema. And then it may generate a set of precise structured queries and evaluate them by using existing XML search engines. To evaluate the quality of the results, in *XBridge* we propose a scoring function that takes into account the structure and the content of the results together. In addition, we also design an execution plan to retrieve the more qualified results as soon as possible, which is suitable to process top- k keyword search.

Consider the same example again, the user may change to issue “*author:Philip, year:2006, title:xml*” as a keyword query to search the relevant publications. For this keyword query, *XBridge* is able to automatically construct different structured queries for XML documents conforming to different XML schemas. For example, for the source schemas of the two XML documents shown in Figure 1, we can construct two sets of structured queries as shown in Figure 2 and Figure 3, respectively. After that, we can evaluate the structured queries to answer the original keyword query. The book node #4 will be returned as answers. We do not need to identify whether or not the title node #5 and the author node #11 belong to the same publications. As such, the processing performance would be improved greatly due to the specific context in structured queries.

Contributions of this paper: (1) For different data sources, *XBridge* can infer different semantic contexts from a given keyword query with label-term, which can be used to construct adaptive structured queries. (2) A scoring function is proposed to evaluate the quality of the answers by considering the context of the keyword-matched nodes and the contents of the nodes in the answers. (3) An execution plan, adapting to the proposed scoring function, is designed to efficiently process top- k keyword search. (4) Experiments show that *XBridge* can obtain improved performance over previous keyword search approaches.

The rest of the paper is organized as follows: Section 2 provides the definition of XML schema and presents how to identify the context of terms and derive the returned nodes. Section 3 proposes a scoring function to evaluate the quality of returned answers. Section 4 describes our *XBridge* system and the algorithms for constructing and evaluating the generated structured queries. The experi-

mental results are reported in Section 5. Finally, we discuss the related work and conclude the study of this work in Section 6 and Section 7, respectively.

2 Identifying Context and Returned Nodes

In this section, we show how to identify the context and the types of return nodes for a keyword query w.r.t. XML schema. Here, we use XML schema tree to represent the structural summary of XML documents. Formally, a keyword query consists of the forms $l : k$, $l : \text{or} : k$ where l is a label and k is a term in [4]. The query model in our work *XBridge* permits users to distinguish the semantics of predicates from the returned nodes, such as we extend $l :$ to $l : *$ and $l : ?$. The former means that one node should exist in the returned answers where the node’s tagname is same to the label l and the node’s values may be anything. The latter shows that the information of the nodes will be returned as answers if the nodes’ tagname is same to the label l .

Definition 1. (*XML Schema Tree*) An XML Schema Tree is defined as $T = (V, E, r, Card)$ where V is a finite set of nodes, representing elements and attributes of the schema T ; E is set of directed edges where each edge $e(v_1, v_2)$ represents the parent-child (containment) relationship between the two nodes $v_1, v_2 \in V$, denoted by $P(v_2) = v_1$ or $v_2 \in Ch(v_1)$; r is the root node of the tree T ; $Card$ is a set of mappings that maps each $v \in V$ to $\{1, *\}$ where “1” means that v can occur once under its parent $P(v)$ in a document conforming to T while “*” means that v may appear many times.

2.1 Identifying Context of Keywords

From the set of labels given in a keyword query, we can infer the contexts of the terms for a data source based on its conformed XML schema. Each node in an XML document, along with its entire subtree, typically represents a real-world entity. Similarly, given a list of labels l_1, \dots, l_n and an input XML schema tree T , an entity of these labels can be represented with a subtree of T such that it contains at least one node labeled as l_1, \dots, l_n . We define the root node of the subtree as a master entity.

Definition 2. (*Master Entity*) Given a set of labels $\{l_i | 1 \leq i \leq n\}$ and an XML schema tree T , the master entity is defined as the root node of the subtree T_{sub} of T such that T_{sub} contains at least one schema node labeled as l_1, \dots, l_n .

Based on Definition 2, a master entity may contain one or more than one schema nodes taking a label as their tagnames. If one master entity node only contains one schema node for each label, we can directly generate FOR and WHERE clauses. For example, let $q(\text{year}:2006, \text{title}:xml, \text{author}:Philip)$ be a keyword query over the XML document d_2 in Figure 1(c). Based on the schema t_2 in Figure 1(d), we can obtain two master entities *book* and *article*. Since the master entity *book* only contains one node labeled as *year*, *title* and *author* respectively, we can directly construct “For \$b in bibliography/bib/book” and

“Where $\$/year='2006'$ and contains($\$/title, 'xml'$) and contains($\$/author, 'Philip'$)”. Similarly, we can process another master entity *article*. The constructed queries are shown in Figure 3.

If one master entity node contains more than one nodes taking the same label, to construct FOR and WHERE clauses precisely, we need to identify and cluster the nodes based on the semantic relevance of schema nodes within the master entity. To do this, we may deploy the ontology knowledge to precisely estimate the semantic relevance between schema nodes. However, the computation adding additional measurement may be expensive. Therefore, in this paper we would like to infer the semantic relevance of two schema nodes by comparing their descendant attributes or subelements. For example, given any two schema nodes v_1 and $v_2 \in$ a master entity T_{sub} , we can infer that the two schema nodes v_1 and v_2 are semantic-relevant nodes if they hold: $semi(v_1, v_2) \geq \sigma$. Here σ is the similarity threshold. If σ is set to 0.8, then it means that v_1 and v_2 contain 80% similar attributes or subelements.

Consider the same query q and the document d_1 in Figure 1(a). Based on the schema t_1 in Figure 1(b), we know there exists one master entity *bib* that contains one node with the label *year* and two nodes with the same labels *title* and *author* respectively. In this case, we first cluster the five nodes as C : $\{ year, \{title, author\}_{book}, \{title, author\}_{article} \}_{bib}$, and then identify whether or not the subclusters in C are semantic-relevant schema nodes. For instance, although $\}_{book}$ and $\}_{article}$ have different labels, both of them contain the same nodes *title* and *author*, i.e., the two nodes *book* and *article* contain 100% similar attributes. Therefore, the cluster C is partitioned into two clusters: C_1 : $\{ year, \{title, author\}_{book} \}_{bib}$ and C_2 : $\{ year, \{title, author\}_{article} \}_{bib}$. When all subclusters cannot be partitioned again, we can generate different sets of FOR and WHERE clauses. For C_1 , we have “For $\$/$ in bibliography/bib” for $\}_{bib}$ and “For $\$/2$ in $\$/book$ ” for $\}_{book}$ together. And its WHERE clause can be represented as “Where $\$/year='2006'$ and contains($\$/2/title, 'xml'$) and contains($\$/2/author, 'Philip'$)” according to the labels in the subclusters of C_1 . Similarly, we can process C_2 to generate its FOR and WHERE clauses. Figure 2 shows the expanded structured queries.

The contexts of the terms can be identified by computing all the possible master entities from the source schemas first, and then specifying the precise paths from each master entity to its labels by checking the semantic-relevant schema nodes. Once the contexts are obtained, we can generate the FOR and WHERE clauses of the structured queries for a keyword query. By specifying the detailed context in FOR clauses, we can limit the range of evaluating the structured queries over the XML data sources, which can improve the efficiency of processing the keyword queries.

2.2 Identifying Returned Nodes

Given a keyword query $q = \{l_i : k_i | 1 \leq i \leq n\}$ and an XML schema tree $T = (V, E, r, Card)$, we may retrieve a set of master entities $V_m \subseteq V$ for q w.r.t. T based on the above discussion. In this section, we will derive the returned

nodes only by identifying the types of the master entities V_m . For any master entity $v_m \in V_m$, if $\text{Card}(v_m) = "*"$, we can determine that the node v_m can be taken as return nodes in the corresponding RETURN clauses because the node represents the real entity at the conceptual level. However, if $\text{Card}(v_m) = "1"$, the node v_m may not represent an entity. In this case, we probe its ancestor nodes until we find its nearest ancestor v_a , such that $\text{Card}(v_a) = "*"$.

Consider another keyword query $q(\text{title:xml}, \text{author:Philip})$ over the XML document d_1 conforming to t_1 . We are able to obtain two master entities *book* and *article*. Since $\text{Card}(\text{book}) = "*"$ and $\text{Card}(\text{article}) = "*"$, we can take them as the return nodes in the corresponding RETURN clauses. However, if users issue a simple query $q(\text{title:xml})$ over d_1 , the master entity of this query is the *title* that is only an attribute of the book or article nodes. In this case, we can trust that users would like to see the information of the whole entity (book or article), rather than one single attribute. Therefore, to generate meaningful RETURN clauses, we have to extend the title node to its parent book or article nodes as the return nodes because book or article nodes belong to *-node type.

If there are some label-term pairs in the form of $\{l_{j_k} : ? | 1 \leq k \leq m \leq n \wedge 1 \leq j_k \leq n\}$ in $q = \{l_i : k_i | 1 \leq i \leq n\}$, instead of returning the master entity of q , we will compute the master entity of $\{l_{j_k}\}$ and use it to wrap all return values of l_{j_k} in the RETURN clause. This is because users prefer to see those nodes with the labels $\{l_{j_k}\}$ as the tagnames.

3 Scoring Function

Given a keyword query q and an XML schema tree T , a set of structured queries Q may be constructed and evaluated over the data source conforming to T for answering q . The answer to the XML keyword query q may be a big number of relevant XML fragments. In contrast, the answer to the top- k keyword query is an ordered set of fragments, where the ordering reflects how closely each fragment matches the given keyword query. Therefore, only the top k results with the highest relevance w.r.t. q need to be returned to users. In this section, our scoring function consists of the context of the given terms and the weight of each term.

Let a fragment A be an answer of keyword query q . It is true that we can determine a structured query q_i that matches the fragment A . This is because we first construct the structured query q_i from the keyword query q and then obtain the fragment A by evaluating q_i over XML data. Therefore, we can compute the context score of the fragment A by considering the structure of the query q_i .

Definition 3. (*Context Score*) Assume the structured query q_i matching the answer A consists of the labels $\{l_i | 1 \leq i \leq n\}$. Let its master entity be v_m and an XML schema be $T = (V, E, r, \text{Card})$, we can obtain a list of nodes $V' \subseteq V$ that match each label respectively.

$$\text{ContextScore}(A, q_i) = \frac{1}{n} \times \sum_{i=1}^n K_{\text{context}}(v_i, v_m) \quad (1)$$

where $K_{context}(v_i, v_m)$ is computed based on the distance from the node $v_i \in V'$ matching the label l_i to its master entity v_m , i.e., $LengthOfPath^{-1}(v_i, v_m)$.

In order to effectively capture the weight of each individual node, we are motivated by the $tf * idf$ weight model. Different from IR research, we extend the granularity of the model from document level to element level.

$$tf(v_i, t_i) = |\{v \in V_d | tag(v) = tag(v_i) \& t_i \in v\}| \quad (2)$$

$$idf(v_i, t_i) = \log\left(\frac{|\{v \in V_d : tag(v) = tag(v_i)\}|}{|\{v \in V_d : tag(v) = tag(v_i) \& t_i \in v\}|}\right) \quad (3)$$

Intuitively, the tf of a term t_i in a node v_i represents the number of distinct occurrences within the content of v_i while the idf quantifies the extent to which the nodes v with the same tagname as v_i in XML document node set V_d contain the term t_i . The fewer v_i nodes whose contents include the term t_i , the larger the idf of a term t_i and a node v_i . Without loss of generality, we will also assume that the weight value of each node is normalized to be real numbers between 0 and 1.

Definition 4. (*Weight of Individual Keyword*) The answer A contains a set of leaf nodes as tagnames with the given labels $\{l_i | 1 \leq i \leq n\}$ where each leaf node should contain the corresponding term at least once a time. For each node v_i with label l_i , we have:

$$\omega(v_i, t_i) = \frac{tf(v_i, t_i) \times idf(v_i, t_i)}{\max\{tf(v_i, t_i) \times idf(v_i, t_i) | 1 \leq i \leq n\}} \quad (4)$$

Definition 5. (*Overall Score of Answer*) Given a generated structured query q_i and its answer A , the overall score of the answer can be computed as:

$$Score(A, q_i) = \frac{1}{n} \times \sum_{i=1}^n \frac{\omega(v_i, t_i)}{LengthOfPath(v_i, v_m)} \quad (5)$$

Assume q consists of three pairs “*author:Philip, year:2006, title:xml*”. We evaluate the keyword query q over the XML document in Figure 1(c). After that, the fragment *book* in the box will be returned as an answer. Based on the pre-computation, we can get the weight of each keyword-matched node in the fragment where we assume each term only occurs once a time in the corresponding nodes.

According to Equation 1, Equation 4, Equation 5, we find that the overall score of an answer is equal to its context score when the weight of each keyword is set to 1 (the maximal value). Therefore, the context score can be taken as the upper bound of the answer.

4 Implementation of XML Keyword Query

In our *XBridge* system, for a given keyword query, we first construct a set of structured queries Q based on the labels in the keyword query and the XML

schemas, i.e., query structuring. Then Q will be sorted according to their context score based on Definition 3. After that, we get a query $q_i \in Q$ with the highest context score from the set of structured queries and send it to XQuery engine. We will evaluate the query q_i over XML data and retrieve all the results that match q_i . At last, we will process all the current results, i.e., computing the overall score for each result and caching the k results with the higher overall scores. After we complete the evaluation of the query q_i , we need start a new loop to process another structured query that comes from the current set $\{Q - q_i\}$ until the top k qualified results have been found. Now let us discuss the detailed procedures of query structuring and execution plan.

4.1 Query Structuring

Algorithm 1 Constructing structured queries

input: a query $q = \{l_i : k_i | 1 \leq i \leq n\}$ and an schema $T = (V, E, r, Card)$

output: a set of structured queries Q

- 1: Retrieve a list of nodes $V_i \subseteq V$ for each l_i in T ;
 - 2: Compute the master entities V_m by calling for **Pipeline**+ σ approach;
 - 3: **for all** each master entity $v_m \in V_m$ **do**
 - 4: Generate **FOR clause** with v_m , i.e. “For $\$x$ in $r/.../v_m$ ”;
 - 5: Cluster the nodes matching with labels l_i in the subtree of v_m by calling for $\text{Cluster_Domain}(\{l_i\}, v_m)$;
 - 6: **for all** each cluster **do**
 - 7: Construct a set of **FOR clauses** for the entity nodes representing different semantic ranges in the cluster;
 - 8: Generate **WHERE clause** with n paths from v_m to each node l_i ;
 - 9: Generate **RETURN clause** by identifying the types of v_m ($\text{Card}(v_m)=*$ or 1) and k_i (“?” symbol exists or not) and put the structured query into Q ;
 - 10: **return** the set of structured queries Q ;
-

Query structuring is the core part of *XBridge* where we compute the master entities, infer the contexts and derive the return nodes. To search the master entities quickly, Dewey number is used to encode the XML schema. Given a keyword query $q = \{l_i : k_i | 1 \leq i \leq n\}$ and an XML schema tree $T = (V, E, r, Card)$, we first retrieve a list of nodes $V_i \subseteq V$ for each l_i in T and sort them in a descending order. To compute master entities, we propose two approaches: (1) **Pipeline**: We take each node v_i from the list V_i where $|V_i| = \min\{|V_1|, |V_2|, \dots, |V_n|\}$ and compute the NCA (nearest common ancestor) of v_i and the nodes in the other lists V_j ($1 \leq j \leq n \wedge j \neq i$) in a sequence. At last, we preserve the NCA nodes as the master entity candidates in V_m ; and (2) **Pipeline**+ σ : When we get any node v_{ix} from V_i , we may take its next node v_{ix+1} as a threshold σ if the node v_{ix+1} exists in V_i . During the computation of NCA, we only probe the subset of nodes V in the other lists such that for $v \in V$ we have $v \prec v_{ix+1}$. As an optimization approach, the second one can reduce the negative computations of NCA while it can obtain the same structured queries as the first one does.

Since the subtree of a master entity v_m may cover one or more than one schema nodes labeled with the same label and the nodes may have different

semantic relevances, we develop a function `Cluster_Domain()` to identify and cluster the nodes matching with the labels l_i ($1 \leq i \leq n$) in the subtree according to the labels and the data types of their attributes or subelements. For each cluster c , we do not generate a structured query if the cluster c only contains a part of the labels in the given keyword query. Otherwise, we construct a structured query for the cluster c . In this case, we first generate a set of FOR clauses according to the classified clusters in the cluster c and then construct a WHERE clause for c . Finally, a RETURN clause is derived by identifying the type of the master entity node v_m . After all the clusters are processed, we may generate a set of structured queries Q . The detailed procedure has been shown in Algorithm 1.

4.2 Execution Plan for Processing Top- k Query

Algorithm 2 Dynamic Execution Plan

input: A set of ranked structured queries Q and XML data D

output: Top k qualified answers

```

1: Initialize the answer set  $SA = \text{null}$ ;
2: Initialize a boolean symbol  $flag = \text{false}$ ;
3: while  $Q \neq \text{null}$  and  $flag \neq \text{true}$  do
4:   Get a query  $q = \text{getAQuery}(Q)$  where  $q$  has the maximal context score;
5:   if  $|SA| = k$  and  $(\text{minScore}\{A \in SA\} \geq \text{ContextScore}(q))$  then
6:      $flag = \text{true}$ ;
7:   else
8:     Issue  $q$  to any XQuery search engine and search the matched fragments  $F_m$ ;
9:     for all  $A \in F_m$  do
10:      Compute overall score  $\text{Score}(A, q)$ ;
11:      if  $|SA| < k$  then
12:        Input the result  $A$  into  $SA$ ;
13:      else if  $\exists A' \in SA$  and  $\text{Score}(A', q') < \text{Score}(A, q)$  then
14:        Update the intermediate results  $\text{updateSA}(A, SA)$ ;
15: return Top  $k$  qualified answers  $SA$ ;
```

Given a keyword query q and XML documents D conforming to XML schemas, we may generate a set of structured queries Q . To obtain top- k results, a simple method is to evaluate all the structured queries in Q and compute the overall score for each answer. And then we select and return the top k answers with the k highest scores to users. However, the execution is expensive when the number of structured queries or retrieved results is large.

To improve the performance, we design an efficient and dynamic execution plan w.r.t. our proposed scoring function, which can stop query evaluation as early as possible by detecting the intermediate results. Our basic idea is to first sort the generated structured queries according to their context scores, and then evaluate the query with the highest score where we take the context score of the next query as the current threshold. This is because the weight of each keyword is assumed to be set as 1 (the maximal value). In this case, the overall score would be equal to the context score. Therefore, if there are k or more than k results, we will compute their overall scores based on Equation 5 and cache k

results with the k highest scores. At the same time, we will compare the scores of the k results and the threshold. If all the scores are larger than the threshold, we will stop query evaluation and return the current k results, which means no more relevant results exist. Algorithm 2 shows the detailed procedure of our execution plan.

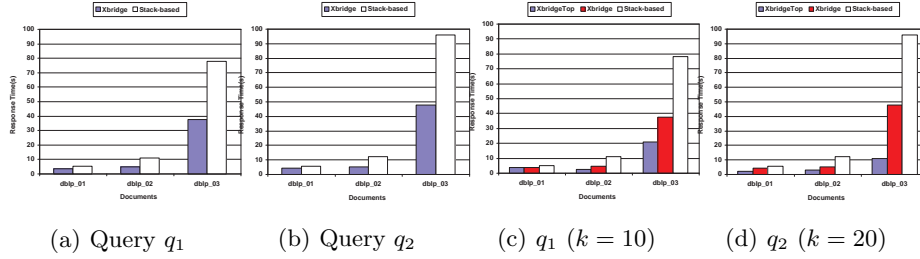
5 Experiments

We implemented *XBridge* in Java using the Apache Xerces XML parser and Berkeley DB. To illustrate the effectiveness and efficiency of *XBridge*, we also implemented the Stack-based algorithm[9] because the other related approaches in [10–12] are biased to the distribution of the terms in the data sources. All the experiments were carried out on a Pentium 4, with a CPU of 3GHz and 1GB of RAM, running the Windows XP operating system. We selected the Sigmod Record XML document (500k) and generated three DBLP XML documents as the dataset. In this paper, we evaluated the following keyword queries: q_1 (*author : David, title : XML*) and q_2 (*year : 2002, title : XML*).

As we know, the Stack-based algorithm may avoid some unnecessary computations by encoding the documents with the Dewey scheme. But it has to preserve all possible intermediate candidates during query evaluation and lots of them may not produce the results. However, *XBridge* can infer the precise contexts of the possible results based on the source schemas before query evaluation. Therefore, *XBridge* can outperform the Stack-based algorithm. For example, the Stack-based algorithm spent 188ms to evaluate q_1 on SigmodRecord.xml while *XBridge* only used 78ms to process the same keyword search. In addition, if the size of an XML document increases, most of the time it may contain more nodes that match a single keyword. But the number of return nodes that match all the keywords may not be increased significantly. In this case, the performance of query evaluation may be relatively decreased.

Figure 4(a) and Figure 4(b) illustrate the time cost of both methods when we evaluate q_1 and q_2 on the given three DBLP datasets respectively. From the experimental results, we find that in *XBridge* the change in time is not obvious when the size of dataset is less than 50M. But when the size of document is nearly 70M, the processing speed was decreased by 80%. This is because compared with dblp_01 or dblp_02, dblp_03.xml contains a huge number of nodes that match each single keyword but fail to contribute to return nodes. The figures also shows that *XBridge* would outperform over Stack-based algorithm in our experiments.

Figure 4(c) and Figure 4(d) compare the performance of the methods when k value is set as 10 or 20 respectively. Since Stack-based algorithm has to retrieve all the results and then select the k qualified answers, its response time for top- k query is nearly same to process general query. However, *XBridge* depends on the dynamic execution plan, which can stop query evaluation as early as possible and guarantee no more qualified results exist in the data source. Generally, *XBridge* is suitable to process top- k keyword search and most of the time, it only needs to evaluate parts of the generated structured queries. But when the number of



relevant results is less than k , *XBridge* also needs to evaluate all the generated structured queries. For example, as we know the document *dblp_01* only contains 2 qualified answers for the keyword query q_1 . When the specified value of k is 10, there is no change of the response time. Therefore, in this special case, its efficiency cannot be improved.

6 Related Work

Recently, keyword search has been investigated extensively in XML databases. Given a keyword query and an XML data source, most of related work [9, 5, 10–12] first retrieve the relevant nodes matching with every single keyword from the data source and then compute LCAs or SLCAs of the nodes as the results to be returned. XRANK [9] and Schema-Free XQuery [5] develop stack-based algorithms to compute LCAs as the results. [10] proposes the Indexed Lookup Eager algorithm when the keywords appear with significantly different frequencies and the Scan Eager algorithm when the keywords have similar frequencies. [13] takes the valuable LCA as results by avoiding the false positive and false negative of LCA and SLCA. [11, 14] takes the similar approaches as [10]. But it focuses on the discussions how to infer RETURN clauses for keyword queries w.r.t. XML data. [12] designs a MS approach to compute SLCAs for keyword queries in multiple ways. In addition, there are several other related work that process keyword search by integrating keywords into structured queries. [15] proposes a new query language XML-QL in which the structure of the query and keywords are separated. [5] embeds keywords into XQuery to process keyword search. Differently, *XBridge* first constructs the structured queries for the given keyword query based on the source schemas and then evaluates the generated structured queries in a sequence. For a top- k keyword query, *XBridge* can return the k qualified answers as early as possible without processing all the generated queries.

Ranking schemes have been studied for XML keyword search. XKeyword [16] ranks query results according to the distance between different keywords in the document. The ranking scheme in XSearch [4] takes the summary of the weights of all keywords within a result to evaluate its relevance. The ranking scheme in XRank [9] takes into account result specificity, keyword proximity and hyperlink awareness together. However, our scoring function considers the context of the matched keywords and the weight of each keyword-matched node.

7 Conclusions

In this paper, we have proposed *XBridge* - an adaptive XML keyword search approach to process keyword search by constructing effective structured queries, which can improve the performance of keyword search greatly by specifying the precise contexts of the constructed structured queries. In addition, we have also provided a scoring function that considers the context of the keywords and the weight of each keyword in the data source together. Especially, an execution plan for processing top- k keyword search has been designed to adapt to our proposed scoring function.

References

1. Albrecht Schmidt, Martin L. Kersten, and Menzo Windhouwer. Querying XML Documents Made Easy: Nearest Concept Queries. In *ICDE*, pages 321–329, 2001.
2. Anja Theobald and Gerhard Weikum. The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. In *EDBT*, pages 477–495, 2002.
3. Vagelis Hristidis, Nick Koudas, Yannis Papakonstantinou, and Divesh Srivastava. Keyword Proximity Search in XML Trees. *IEEE Trans. Knowl. Data Eng.*, 18(4):525–539, 2006.
4. Sara Cohen, Jonathan Mamou, Yaron Kanza, and Yehoshua Sagiv. XSearch: A Semantic Search Engine for XML. In *VLDB*, pages 45–56, 2003.
5. Yunyao Li, Cong Yu, and H. V. Jagadish. Schema-Free XQuery. In *VLDB*, pages 72–83, 2004.
6. Sihem Amer-Yahia, SungRan Cho, and Divesh Srivastava. Tree pattern relaxation. In *EDBT*, pages 496–513, 2002.
7. Jianxin Li, Chengfei Liu, Jeffrey Xu Yu, and Rui Zhou. Efficient Top-k Search across Heterogeneous XML Data Sources. In *DASFAA*, pages 314–329, 2008.
8. Weidong Yang and Baile Shi. Schema-aware keyword search over xml streams. In *CIT*, pages 29–34, 2007.
9. Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In *SIGMOD Conference*, pages 16–27, 2003.
10. Yu Xu and Yannis Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *SIGMOD Conference*, pages 537–538, 2005.
11. Ziyang Liu and Yi Chen. Identifying meaningful return information for XML keyword search. In *SIGMOD Conference*, pages 329–340, 2007.
12. Chong Sun, Chee Yong Chan, and Amit K. Goenka. Multiway slca-based keyword search in xml data. In *WWW*, pages 1043–1052, 2007.
13. Guoliang Li, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. Effective keyword search for valuable lcas over xml documents. In *CIKM*, pages 31–40, 2007.
14. Ziyang Liu and Yi Chen. Reasoning and identifying relevant matches for xml keyword search. *PVLDB*, 1(1):921–932, 2008.
15. Daniela Florescu, Donald Kossmann, and Ioana Manolescu. Integrating keyword search into XML query processing. *Computer Networks*, 33(1-6):119–135, 2000.
16. Vagelis Hristidis, Yannis Papakonstantinou, and Andrey Balmin. Keyword Proximity Search on XML Graphs. In *ICDE*, pages 367–378, 2003.