Beyond Bag of Words: A New Model for XML Keyword Query

Xiping Liu
School of Information Technology
Jiangxi University of Finance and Economics
Nanchang 330013, China
Email: liuxiping@jxufe.edu.cn

Changxuan Wan
School of Information Technology
Jiangxi University of Finance and Economics
Nanchang 330013, China
Email: wanchangxuan@263.net

***Abstract*—Keyword search is an effective paradigm for information discovery and has been introduced recently to query XML documents. Effective keyword search of XML documents needs full understanding of the keyword query. Traditional bag-of-words model cannot differentiate the roles of keywords as well as the relationship between keywords, thus is not proper for XML keyword queries. In this paper, we present a novel model specially designed for XML keyword query. The model takes a very different point of view on a keyword query: a keyword query is interpreted as a composition of several query units, each representing a query condition. We believe that this viewpoint captures the semantics of the query. To get an objective measure of the relevances of results with respect to the query, we devise a scoring method based on the proposed model that caters for query semantics as well as the structural properties of XML documents. Experimental results verify the effectiveness of our methods.**

***Index Terms*—XML keyword query, query model, scoring**

I. Introduction

Keyword search is an effective paradigm for information discovery that has been extensively studied for flat documents (text, HTML, etc.) As XML has been accepted as a standard for document mark-up and exchange, it is natural to extend keyword search techniques to support XML data [1], [2].

Scoring is at the core of keyword search. A scoring method assigns a score to a query result which reflects the relevance of the result with respect to the query at hand. Scoring methods have been extensively studied in traditional information retrieval (IR) field, and a number of scoring functions have been proposed, such as Okapi BM25 [3].

Ideally, a scoring method should reveal the semantic relevance of the query and the documents. Thus, designing a scoring method requires full understanding of the document as well as the query. Several scoring methods have been proposed concerning XML keyword search [1]. Some of the methods try to guess the intentions of the queries, and retrieve and score results based on some intuitions. For example, some work assumes that by a keyword query, the user is intended to find the SLCAs (smallest lowest common ancestor) [4]. Then they rank the SLCAs based on their structural importances. Other methods take more evidences of importance into account, such as the size of the resulting tree, the frequencies of the keywords, and so on.

Existing XML scoring methods are based on the "bag-of-words" model. In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. Though simple enough, the model is not well-suited for XML keyword search.

Consider a query Q_1: "journal database article transaction". The query intends to search for articles about "transaction" in a journal named "database". Given an XML document, an ideal result of the query may be a subtree rooted at an element labeled "article" containing "transaction" in its text content, which is nested in an element labeled "journal" with "database" in its content. Obviously, it is not proper to view the query as a bag of words. First, the keywords in the query are different in their roles. The keywords "article" and "journal" should be treated as labels of elements, while "database" and "transaction" are keywords appearing in text contents. Second, the relationship between keywords in the same query are different. The keyword "transaction" is more closely related to "article" than to "database", and "database" has closer relationship with "journal" than with "article".

As XML documents have complex structures, a keyword query searching XML documents are more complex in semantics. The traditional bag-of-words model cannot reveal the underlying semantics and intention of the query, thus cannot provide a solid basis for retrieval and scoring. In this paper, we propose a new model, called *keyword query with structure* (QWS), to model a keyword query against an XML document. The QWS model is an enhancement of the bag-of-words model. It is able to differentiate roles of keywords, and describe various connections between keywords. As the QWS model is a semantic modeling of the keyword query, it is not trivial to derive the QWS representation from the query, therefore, we discuss how to construct the QWS model of a keyword query. Then, based on the QWS model, we present a new scoring method, which capture the semantic relevance of the results.

To summarize, we make the following contributions in this paper:

1) We propose a novel model to analyze and interpret XML keyword query. The approach decomposes a keyword query into several fragments, each representing a

978-1-4799-6543-4/14 $31.00 © 2014 IEEE
DOI 10.1109/ICMeCG.2014.59

query unit. By this approach, we are able to capture the semantics of a keyword query and achieve a deep understanding of the query.

2) We present a novel scoring method that reflects the semantic relevance of results. The method is based on the proposed QWS model, and considers structural properties and matching patterns of the results.
3) We conduct a comprehensive set of experiments to verify the effectiveness of the proposed methods. Experimental results demonstrate that our method is effective.

The rest of the paper is organized as follows. In section 2, we give the motivation of the model for XML keyword query. In section 3, the proposed QWS model is presented. We develop a semantics-aware scoring method in section 4. Section 5 presents experimental studies. Section 6 reviews related work and we finally conclude the paper in section 7.

II. Motivation

Effective scoring requires full understanding of the query. When querying XML documents, which usually have complex structures, with simple keywords, the following issues should be addressed properly.

(1) Understanding the roles of keywords.

Keywords in a query against XML documents have different roles. Some keywords indicate the labels of constrained elements, others are simple keywords expected to appear in the texts.

Example 1. An XML document representing a bibliography database is shown in Fig. 1, which contains information about journals, articles and authors. Each node represents an element. In Fig. 1, some nodes that are not interesting to us are hidden, such as volume, num, etc. This sample will be used throughout this paper. Consider a query Q_1: "journal database article transaction". The query intends to search for articles about "transaction" on journals named "database". In this query, "journal" and "article" are expected to match the label of elements, and "database" and "transaction" are expected to appear in the text content of certain elements. Understanding the roles of the keywords is essential to obtaining desired results.

We make a distinction between query terms intending to match a label and terms expected to appear in text content of some elements.

Definition 1: Given a term t in a keyword query Q issued on an XML database, if t appears in the text content of some nodes, t is a *content query term*, or *C-term* in short; if t appears as a tag, t is a *tag query term*, denoted as *T-term*. The *type* of a query term refers to whether it is a C-term, T-term or both.

Consider the query Q_1: "journal database article transaction". According to the definition above, "database" and "transaction" are C-terms, and "journal" and "article" are T-terms.

The type of a term can be inferred from the contexts of the term where it appears in an XML document. We assume that an inverted list is built for each term, where each appearance of the term is recorded in the list. Using the inverted lists, T-terms and C-terms can be discovered.

(2) Understanding the relationships between keywords.

Keywords have not only different roles, but also different relationships. That is, some keywords are more closely related to some others. The relationship between query terms can be categorized as follows. Given a C-term and a T-term, the C-term may be constrained by the T-term, i.e. the C-term is expected to appear in the node labeled by the T-term. Given two C-terms ct_1 and ct_2, they may be constrained by the same T-term, i.e. they are expected to appear in the same node.

Consider the query Q_1: "journal database article transaction", the C-term "database" is constrained by "journal", while "transaction" is constrained by "article". Therefore, "database" is more closely related to "journal" than to "article", "transaction" is more closely related to "article" than to "journal". Consider another query Q_2: "journal database article transaction processing", the keywords "transaction" has closer relationship with "processing" than with "database", because they are expected to appear in the same node.

Capturing the relationships between keywords helps reveal the relevance of the results. Recall that in text retrieval, a semantic unit of a query is a keyword, and a basic principle is that the more query terms a document contains, the more relevant the document is. In the context of XML keyword search, however, the principle does not usually hold. In XML keyword search, since some query terms are semantically related, we take a different viewpoint against keyword query, where a query is viewed as a composition of several query conditions instead of individual keywords. Correspondingly, the relevance of a query result is assessed based on its match with query conditions, not its match with query terms. For example, for query Q_1: "journal database article transaction", terms in the query should be grouped into two query conditions: "journal database" constitutes the first query condition; "article" and "transaction" belong to the second query condition. The scoring of query results should be based on how many and how query conditions are matched.

The traditional bag-of-words model does not differentiate various roles of keywords in a query, it does not reveal the relationship between keywords either. Therefore, the bag-of-words model is not appropriate for XML keyword search. In this work, we propose a new model representing a keyword query against XML document, which explicitly capture the information about the queries.

III. The QWS Model

In this section, we present the QWS model for the XML keyword query.

A. The QWS Model

A keyword query can be treated as a simplification of a *natural language query* (NLQ). For example, for text retrieval, the intention of a keyword query "database transaction" can be expressed by a natural language query "find the documents that contain **database** and **transaction**". An NLQ can be

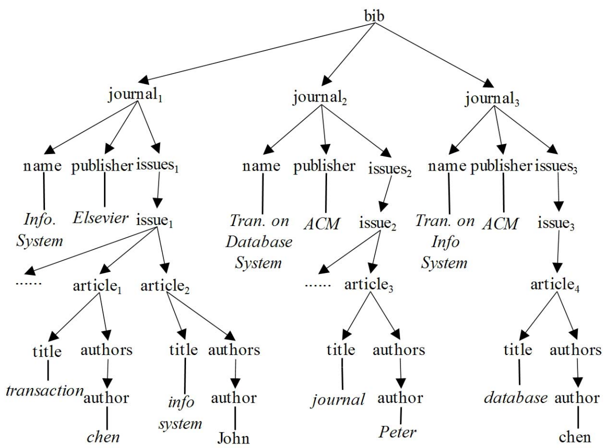


Fig. 1: A sample XML document.

decomposed into several fragments. Each of them represents a query condition. For example, the NLQ above is composed of two fragments, one specifies query condition "containing **database**", and the other one specifies "containing **transaction**". We can analyze XML keyword queries similarly. For example, the query intention implied by the query "article xml search" can be expressed using natural language query "find articles that are about **xml** and **search**". The NLQ indicates a query condition "about **xml search**".

Generally, an XML keyword query can be semantically decomposed into several fragments. Each fragment contains several terms, including T-terms and C-terms. Each fragment indicates a search constraint or a query condition.

Definition 2: Given a keyword query Q, a query unit q of Q is a pair $\langle context, cterms \rangle$, where $context$ is a sequence of T-terms indicating structural constraints, and $cterms$ is a sequence of C-terms indicating content constraints. Below we use $q^{context}_{cterms}$, $context(q)$ and $cterms(q)$ to denote the query unit q, the T-terms in q and the C-terms in q, respectively.

A query unit specifies that some keywords should appear under certain structural context. It indicates a query condition. Note that in some query units, there may be only structural constraints or content constraints. For example, if "info system" makes a query unit, it has only content constraints, i.e., no structural constraint needs to be satisfied. In another example "database transaction author", the query can be decomposed into two query units $q_{\text{database transaction}}$ and q^{author}. The former has no structural constraints while the latter has no content constraints.

Given a keyword query $Q = \langle t_1, \cdots, t_n \rangle$, we can obtain a set of query units $Q_s = \langle q_1, \cdots, q_m \rangle$ from Q such that: (1) $terms(q_i) \subseteq Q (1 \leq i \leq m)$, and (2) $\bigcup_{i=1}^{m} terms(q_i) = Q$, where $terms(q)$ denotes the set of terms in q. As Q_s is a reorganization of Q, and there is structure in Q_s, we say Q_s is a *keyword query with structure* (QWS) derived from Q.

For example, the QWS of Q_1: "journal database article transaction" can be represented as $\{q^{\text{journal}}_{\text{database}}, q^{\text{article}}_{\text{transaction}}\}$.

A QWS can be viewed as an intermediate form between a keyword query and a structured query. Obviously, a QWS is less ambiguous than a keyword query, so if we can derive QWSs from a keyword query, we are probably able to improve the search results and achieve better scoring. The problem here is how to decompose an XML keyword query into a set of query units. Towards this problem, we have several heuristics.

B. Heuristics

In this section, we provide several heuristics with the purpose of gaining some insights into query units and QWS.

Heuristics 1: Given a QWS, if a term t is a C-term, then t appears in exactly one query unit.

Heuristics 1 holds because a C-term is extracted from the content constraint of a query condition, and query conditions do not overlap.

Heuristics 2: Given a keyword query Q, let Q_s be a QWS derived from Q, and q be a query unit in Q_s, the C-terms in q (if any) are consecutive in Q.

The rationale behind Heuristics 2 is that, the content constraints in a query condition does not intersect with that in another one. As a result, the C-terms of a query unit, which are extracted from the content constraint of the corresponding query condition, cover some consecutive terms.

Example 1: If a user tries to find articles about "database transaction" on an "info system" journal, he may issue a query like "journal info system article database transaction" or "article database transaction journal info system" and the like, rather than "journal info transaction article database system". In the first two queries, "journal info system" and "article database transaction" make a query unit, respectively.

Although the C-terms in a query unit are consecutive in the query, a sequence of consecutive C-terms may not belong to the same query unit. Consider a query Q_3: "author Peter info system", the sequence of C-terms "Peter info system" should be split into two groups: "Peter" belongs to the same query unit as "author", while "info system" belongs to another unit. If all terms in a query are C-terms, it is more difficult to separate C-terms. Just consider the query "info system John database transaction". We need to find some hints from the occurrences of the terms.

Definition 3: Let v be a node in XML document D, we use $path(v)$ to denote the tag path from the document root to v. Given a term t in a leaf node v, $path(v)$ is termed as a *document context* of t.

Heuristics 3: If terms $t_1, \cdots, t_k$ are C-terms in the same query unit, it is probable that $t_1, \cdots, t_k$ have some common document contexts.

Heuristics 3 is reasonable because the C-terms in a query unit are extracted from the same query condition, so they probably show the same or similar structural characteristics. According to Heuristics 3, the document contexts of terms provide some clues for separating C-terms.

Example 2: In Q_4: "info system John database transaction", since "info" and "system" typically appear under title and name, but "John" occurs in author, so "info system" should be separated from "John". Similarly, "database transaction" should not be in the same query unit as "John".

The following heuristics are about the relationship between T-terms and C-terms in a query unit.

Heuristics 4: Given a keyword query Q, let q be a query unit in a QWS derived from Q. All the terms in $context(q)$ appear together either before terms in $cterms(q)$, or after terms in $cterms(q)$, in Q.

Heuristics 4 states that the relative positions of the T-terms and C-terms in a query unit are not arbitrary in the keyword query; instead, the T-terms appear either before or after the C-terms, but not both.

Example 3: In the query "journal info system article database transaction", we can decompose the query into two query units $q^{\text{journal}}_{\text{info system}}$ and $q^{\text{article}}_{\text{database transaction}}$; in each unit, the T-terms appear before the C-terms. Consider another query "database transaction article", from which we can get a query unit $q^{\text{article}}_{\text{database transaction}}$, here the T-term is after the C-terms. As another example, suppose a user intends to find articles which have an author "Peter", he will not likely issue a query like "article author Peter name".

Again, given a set of C-terms that belong to the same query unit, the T-terms preceding or following the C-terms in the query do not always belong to the same query unit as the C-terms. For this problem, we have the following heuristics.

Heuristics 5: Let q be a query unit. It is probable that the T-terms in $context(q)$ appear in the common document contexts of C-terms in $cterms(q)$.

Recall that the T-terms in a query unit specify structural constraints of the C-terms, so probably they appear in the paths leading to the C-terms. For example, in Q_3: "author Peter info system", since "author" is the label of an element containing "Peter" but not of "info system", it is probably in the same query unit as "Peter" but not "info system".

The heuristics above provide some insights for us to decompose a keyword query. According to Heuristics 2, each C-term belongs to exactly only one query unit, that is, the unit containing the previous C-term neighbor, or the unit containing the next C-term neighbor, or a new unit containing solely the C-term itself. To decide which query unit a C-term should be assigned, we can examine the candidate query units and choose the best one.

Now, we propose a method to decompose a keyword query. We first classify all C-terms into several groups, then for each group of terms, we pick out T-terms that are structural constraints of these C-terms. Each group of C-terms and the corresponding T-terms make a query unit. The problem here is: 1) how to group C-terms,and 2) given a group of C-terms, how to identify the T-terms that belong to the same query unit as the group of C-terms.

For the first problem, the document contexts of terms provide some clues for separating C-terms, that is, we can split a sequence of C-terms based on their document contexts.

Example 4: In Q_5: "author chen xml search", since "chen" has document context $\backslash\backslash$name, "xml" and "search" have document context $\backslash\backslash$title, "xml search" should be separated from "chen".

Next, we discuss the second problem, that is, given a set of C-terms that belong to the same group, how to identify the T-terms that are in the same fragment. Recall that the T-terms in a query unit specify structural constraints for the C-terms. Naturally, a feasible idea is to pick the T-terms that are the common document contexts of the given C-terms. According to heuristics before, we can go further to confine the candidate T-terms in the T-terms that appear before or after the given C-terms in the query. To sum up, given C-terms $t_1, \cdots, t_k$, the T-terms that (1) are common document contexts of $t_1, \cdots, t_k$, and (2) appear immediately before or after $t_1, \cdots, t_k$ in the query constitute the context part of the fragment.

At last, if some T-terms do not belong to any query unit, they can be grouped into a separate unit containing no C-terms. Note that, a keyword query may have more than query decompositions. Consider the query "journal name algorithm acm", since "acm" has multiple document contexts, e.g. $\backslash\backslash$publisher and $\backslash\backslash$name, the query can be understood as one query condition $q^{\text{journal name}}_{\text{algorithm acm}}$, or two query conditions $q^{\text{journal name}}_{\text{algorithm}}$ and $q^{\text{journal}}_{\text{acm}}$. In the former decomposition, user intends to find journal with "algorithm acm" in name; while in the latter case, a user probably finds journal with name "algorithm" and publisher "acm". In this case, we need to consider different query semantics and output a set of decompositions.

IV. QWS-based Scoring

In this section, we propose a new scoring method based on the QWS model of XML keyword query. Specifically, we first compute the scores of results with respect to each query unit, and then combine the scores on query units to arrive at a final score.

A. Scoring on Query Units

Given a result v, its score with respect to query unit q, denoted as $score(v, q)$, is computed as follows:

$$score(v, q) = sim_C(v, q) * (1 + \sqrt{sim_S(v, q)}) \qquad (1)$$

In formula (1), the score of a result with respect to query unit is composed of two parts: $sim_C(v,q)$ is the score of v on C-terms of q, while $sim_S(v,q)$ is the score of v on T-terms of q. The details of $sim_C(v,q)$ and $sim_S(v,q)$ will be discussed later. We now focus on the combination of the two parts. Various combining methods can be applied here. The key idea in formula (1) is that, C-terms and T-terms play different roles in q, so two partial scores should be differentiated and attached different importance in the formula. We hold that, user's information need is mainly conveyed through C-terms, while T-terms are used as hints to find the desired information or constraints to filter out undesired information. For example, consider a query "article title xml database" which looks for articles titled "xml database". Without a doubt, articles with "xml database" as keywords should also be considered as relevant, although they do not match the query on title. Inspired by the content-oriented feature of keyword search, formula (1) is designed as content-oriented, where the combined score mainly depends on the score on C-terms. Scores on T-terms, however, can also exert influence. Reconsider the query above, articles with "xml database" as keywords should be viewed as less relevant as papers titled "xml database". This influence is also reflected in formula (1). We use the square root instead of the original value of $sim_S(v,q)$ to decrease the impact of structural differences.

Now we will explain the computation of $sim_C(v,q)$ and $sim_S(v,q)$. Let $q.terms$ be the set of C-terms in q, n is a witness node matching $q.terms$, $n.content$ is the text content of n. Then $sim_C(v,q)$ can be taken as the similarity between $q.terms$ and $n.content$, which has been extensively studied in IR field. The technique we use is a rather standard tf*idf method [3]. Specifically, the weight of a term t in node $n.content$ and $q.terms$ is determined by $w_{t,n}$ and $w_{t,q}$ as follows, respectively:

$$w_{t,n} = 1 + \log tf_{t,n} \tag{2}$$

$$w_{t,n} = (1 + \log tf_{t,1}) \times \log \frac{N}{df_t} \tag{3}$$

where $tf_{t,n}$ is the term frequency of t in $n.content$, $tf_{t,q}$ is the term frequency of t in $q.terms$; N is the total number of text units in the document, and df_t is the number of text units containing t. With this weighting scheme, the score of $n.content$ for $q.terms$ can be computed using the cosine similarity between the vector representations of $n.content$ and $q.terms$, that is

$$score(q.terms, n.content) = \frac{\vec{V}_{q.terms} \cdot \vec{V}_{n.content}}{|\vec{V}_{q.terms}||\vec{V}_{n.content}|} \tag{4}$$

On the other hand, $sim_S(v,q)$ evaluates the similarity of structural part of v and T-terms of q. Again, let n be a witness node matching $q.terms$. We define $sim_S(v,q)$ as the similarity between the document context of n and T-terms in q. The method to compute the similarity is rather straightforward: let the document context of n be $DC(n)$, the set of T-terms in q are $T\text{-}terms(q)$, then

$$sim_S(v,q) = \begin{cases} \frac{|\{t|t\in T\text{-}terms(q), \exists l \in DC(n), l \approx t\}|}{|T\text{-}terms(q)|} & \text{if } T\text{-}terms(q) \neq \phi \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where $\approx$ stands for similar relationship, depending on different similarity functions, we can simply use "=" instead of $\approx$. In this formula, the percentage of the T-terms that are matched by context of n is computed and the value is used as $sim_S(v,q)$.

Given a result v and a query unit q, there may be multiple matches of q. In this case, we use the maximum score as the score of v on q.

B. Combining Scores

Now we move to the next topic, that is how to combine the scores on query units to get an overall score. An intuitive scheme is to sum up the scores on each query units so that the more query units are matched by an answer, the greater score it has. Based on this idea, we have:

$$score(v,Q) = \sum_{q_i \in Q} \rho_i \cdot score(v, q_i) \tag{6}$$

where Q is the keyword query, q_i is the i-th query unit of Q, ρ_i is the weight of the q_i, which represents the relative importance of a query unit in the query. To simplify the problem, we assume that all ρ_i are the same, e.g. 1.

While the number of matched query units is one factor to be considered, another point we should consider is how the query units are connected in the answer.

Example 5: Consider a query Q_6: "journal transaction database". Two answers of Q_6 are journal$_2$ and journal$_3$. The former has sub-node name with value "Transactions on Database Systems"; the latter has a child name with value "Transactions on Information Systems" and a descendant title under article with value "database". Obviously, journal$_2$ should be considered more relevant than journal$_3$ to the query. The difference between these two results lies in the matching patterns, which should be reflected in scoring.

Given a keyword query Q and a result v, let $n_1, \cdots, n_k$ be the best matches of C-terms $t_1, \cdots, t_k$, respectively. Here, n_i is the best match of t_i if we cannot find another match n_j of t_i such that the score of v on Q is greater. Then, we use best matching tree to measure the relative quality of the matching pattern of v on Q.

Definition 4: Given a keyword query Q and a result v, the best matching tree of v on Q is a subtree of v with root $r = lca(n_1, \cdots, n_k)$ and leafs $n_1, \cdots, n_k$ while preserving all paths from r to $n_1, \cdots, n_k$ in v, where $n_1, \cdots, n_k$ are best matches of C-terms in Q, and $lca()$ computes the LCA (Lowest Common Ancestor) [5] of nodes.

For best matching tree, we define a measure called the *compactness* of the tree as follows:

$$c(v,Q) = \frac{1}{k} \sum_{i \in [1,k]} dist(r, n_i) \tag{7}$$

TABLE I: Query set

No.	Query
DQ1	journal ieee tran article pattern recognition
DQ2	article title XML database
DQ3	article web search results clustering
DQ4	xml index query document
DQ5	journal vldb year 2004
DQ6	title search score rank
DQ7	journal ieee computer title software
DQ8	article author jim gray
DQ9	article image retrieval feedback
DQ10	article title database transaction
MQ1	country language french
MQ2	arab organization
MQ3	world trade organization
MQ4	canada city york
MQ5	new york
MQ6	religion muslim

where $dist(r, n_i)$ denotes the distance between r and n_i. That is, the compactness measure of the best matching tree is the average depth of the tree. For example, reconsider query Q_6, for answer $\mathsf{journal}_2$, its compactness is 0 while for $\mathsf{journal}_3$ the value is (1+4)/2=2.5.

The compactness measure reflects the compactness of the best matching tree. A compact best matching tree means that the result is more closely related to the query, and deserves higher score. To reflect the favor of compact best matching tree, we integrate another factor $\beta^{c(v,Q)}$ into the scoring formula, where β is a parameter decided by experiments. We set β to be 0.8 in this paper based on our experiments.

To sum up, the final scoring formula is defined as follows.

$$score(v, Q) = \sum_{q_i \in Q} \rho_i \cdot score(v, q_i) \times \beta^{c(v,Q)} \qquad (8)$$

Finally, if there is more than one decomposition for a keyword query, we need to consider each combination and score respectively, and then combine results.

V. Experimental Studies

A. Experimental Setup

We have implemented a prototype called XScore which supports keyword search on XML documents. The keywords inverted lists are stored in an index build by Lemur toolkit [6]. We run our experiments on a machine with Intel 2.30G CPU and 2G RAM running Windows. All algorithms are implemented in C++. We use two real datasets: DBLP 230M [7] and Mondial [8].

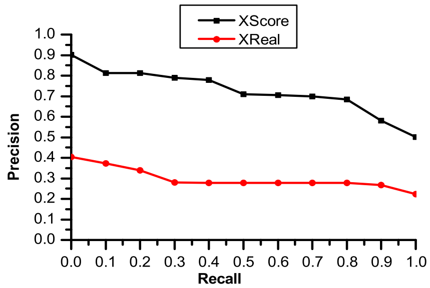
We test several keyword queries for each dataset. The queries are listed in Table I. Queries DQ1-DQ10 are designed for DBLP dataset, MQ1-MQ6 for Mondial dataset. These queries exhibit different features and selectivity.

The main competitor of our method is XReal [9], [10] which is closest to our work, it also represents the state of the art in XML keyword search. The main differences between our method and XReal is that our method is based on the identified query semantics, whereas XReal does not. To reveal the different impact of scoring methods on query results, we generate query results, score results using the two scoring methods, and rank the results according to their scores. Then we evaluate the ranking effectiveness of generated ranked list of results. Note that XML keyword search engine generates results at a granularity of elements or nodes, to make the two scoring methods comparable, we designate the result type for each query. For example, for DQ1, the result type is /dblp/article. Since both XReal and our method XScore are able to generate all results partially matching a query, given the same result type, the sets of results are in fact the same. As a result, the recall and precision of both methods are also the same, and the only differences lie in the ranking of individual results in the lists.

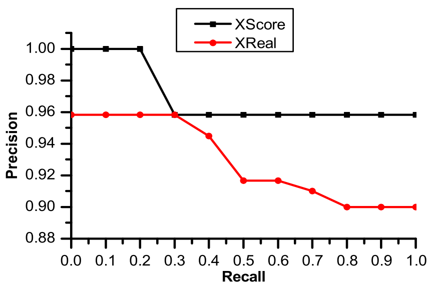
B. Experimental Results

We report the experimental results in this section. The measures we use in the experiments are 11-point interpolated average precisions, Mean Average Precision (MAP), Precision at k (P@k), R-precision and Mean Reciprocal Rank (MRR).

- **11-point interpolated average precisions.** The interpolated precision p_{interp} at a certain recall level r is defined as the highest precision found for any recall level $q \geq r$. For each information need, the 11-point interpolated precisions are the interpolated precision measured at the 11 recall levels of $0.0, 0.1, \cdots, 1.0$. For each recall level, the arithmetic mean of the interpolated precisions at that recall level for all information needs in the test collection are then calculated. A composite precision-recall curve showing 11 points can then be graphed.
- **Mean Average Precision (MAP).** For a single information need, average precision is the average of the precision value obtained for the set of top k documents existing after each relevant document is retrieved, and this value is then averaged over information needs to get Mean Average Precision (MAP). MAP provides a single-figure measure of quality across recall levels and becomes very common in recent years.
- **Precision at k(P@k) and R-precision.** The above measures factor in precision at all recall levels. For many prominent applications, what matters is rather how many good results there are on the first several pages. This leads to measuring precision at fixed low levels of retrieved results, such as 10 or 30 documents. This is referred to as "Precision at k" (P@k), for example "Precision at 10". R-precision is similar to P@k. Given a set of known relevant documents of size Rel, R-precision is the calculated precision of the top Rel documents returned. Similarly, this measure is usually averaged across queries.
- **Mean reciprocal rank (MRR).** The reciprocal rank is 1 divided by the rank at which the first correct answer is returned, or 0 if no correct answer is returned, it is decided by only the position of the first answer. The mean reciprocal rank is the average of the reciprocal ranks of results for a sample of queries. MRR is decided by only the positions of the first answers.



(a) curve on DBLP



(b) curve on Mondial

Fig. 2: 11-point interpolated average precisions curves.

We ask a group of volunteers to specify the search intentions for these queries, which will be used as the ground truth. According to users' information need, we specify the types of results (label paths), and compose schema-aware XML queries (in XQuery), which are then evaluated against DBLP and Mondial to obtain the set of relevant answers.

Figure 2 shows the 11-point interpolated average precisions on both DBLP and Mondial datasets. As we can see, XScore is much better than XReal in precision at each recall level. On DBLP dataset, for any recall level, XScore achieves a precision greater than 0.5, but the precision in case of XReal is below 0.5. On Mondial dataset, both methods achieve high precisions, and the gap between two curves is smaller, but still, XScore outperforms XReal at any recall level. Moreover, XScore is much more steady on this dataset. For each recall level from 0 to 0.2, the precision level is the same; when recall level grows from 0.3 to 1.0, the precisions remain the same. On the other hand, XReal is not so steady, for a high recall level the precision degrades.

The Mean Average Precision (MAP) values for XScore and XReal on DBLP and Mondial are shown in Table II. XScore shows a better MAP than XReal on both datasets. On DBLP dataset, the improvement brought by XScore is significant, which means that the semantic analysis of XML keyword queries is effective and useful.

Table III and Table IV show the precisions at k (P@k) for XScore and XReal. We vary the value of k to get several variants. Specifically, we consider P@1, P@5, P@10 and R-Precision. As we see, XScore exhibit considerable high precisions on DBLP. It also has a better performance than XReal on Mondial for all values of k. The performance shows that the top search results are more relevant under XScore than that under XReal.

TABLE II: MAP on datasets

Methods	DBLP	Mondial
XScore	0.72	0.96
XReal	0.23	0.92

TABLE III: Precision at k on DBLP

Methods	P@1	P@5	P@10	R-Precision
XScore	0.88	0.72	0.75	0.70
XReal	0	0.16	0.1713	0.18

TABLE IV: Precision at k on Mondial

Methods	P@1	P@5	P@10	R-Precision
XScore	1.0	0.5	0.33	0.95
XReal	0.83	0.43	0.25	0.92

TABLE V: MRR on datasets

Methods	DBLP	Mondial
XScore	0.90	1.0
XReal	0.19	0.92

The last table, Table V, presents the MRR metrics for XScore and XReal on both datasets. Again, XScore achieves much higher MRR values, especially on DBLP dataset. The result verifies that the first answers generated by XScore appear before that generated by XReal, which is also due to the proper scoring and ranking method in XScore.

VI. Related Work

Extensive research has been done on XML keyword search. We will review some of them in this section.

Query semantics interpretation. While a number of studies have discussed semantics in XML keyword search, most of them focus on how to effectively connect matches of keywords in a meaningful way. XRank [12] and ELCA [13] connects keyword matches by the LCA nodes that contain at least one occurrence of all keywords after excluding the occurrences of keywords in sub-elements that already contain all keywords. XSEarch [14] introduces the concept of interconnection. Two matches are interconnected if the path from these two nodes and to their LCA may not contain distinct nodes with the same labels except for themselves. Similar concept has also been proposed in [15]. Y. Xu et al [4] introduces the notion of the SLCA which extends the definition of LCA. An SLCA is a LCA that does not have further LCA nodes among their descendants. The concept of MLCA [16] is very similar to that of SLCA with the difference that MLCA impose constraints on the node labels. Recently, [19] propose a new semantics called MCN to capture the relationships of the given keywords from XML document graph by considering reference relationship. XBridge [**?**] proposes an estimation-based approach to compute the promising result types for a keyword query, it considers the value and structural distributions of the data.

Keyword query analysis. Only recently, analysis of query has attracted some attention. XSeek classifies keywords in the query into two categories: search predicates and return nodes, and propose some inference rules. Similarly, XReal identified the ambiguity that a keyword can appear both as an XML tag name and as a text value of some other nodes. However, these studies do not go further to interpret the semantics of query. Nalix [17] is a natural language query interface for a database, in which a large class of natural language queries can be translated into XQuery expressions. It also support iterative search in the form of followup queries. Our method differs with Nalix in that we interpret query semantics in a lightweight way: we do not translate the query but decompose the query; we do not rely on XQuery to express query semantics but use query fragments to represent query conditions. As a result, our method will be more efficient and suitable for keyword search.

XML search results scoring and ranking. XRank [12] extends PageRank to take specific characteristics of XML data into account. The query answers in XSEarch [14] are ranked using a tf*idf-based ranking which also reflects node relationships. EASE [18] combines tf*idf-based IR ranking and structural compactness-based DB ranking to fulfill keyword search on heterogeneous data. XReal [9] designs a novel XML TF*IDF similarity ranking scheme which takes the confidence of node types as search for/search via nodes and the co-occurrence of keywords into consideration. Our method differs with them in that our method is specially designed for the query semantics.

VII. Conclusion

In this paper, we propose a new query model (QWS) for XML keyword search. To interpret query semantics, we distinguish different types of terms in the query, and propose to decompose a keyword query into several query units. Each query unit corresponds to a query condition. To measure the relevance of a search result, we propose a semantics-oriented scoring method. The basic idea of the method is to measure the relevance of an answer based on how many and how well the query conditions are matched, thus caters for query semantics. We conduct comprehensive experiments to verify the effectiveness of our methods. For future work, we plan to investigate effective top-k query processing techniques for XML keyword search.

References

[1] Y. Chen, W. Wang, Z. Liu, et al. Keyword Search on Structured and Semi-structured Data. (tutorial) In SIGMOD 2009, pages 1005-1010, 2009.

[2] J. Coffman. An Empirical Performance Evaluation of Relational Keyword Search Techniques. IEEE Transactions on Knowledge and Data Engineering, 2014, 26(1): 1041-4347.

[3] C.D. Manning, P. Raghavan and H. Schtze. Introduction to Information Retrieval. Cambridge University Press, 2008.

[4] Y. Xu and Y. Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In SIGMOD 2005, pages 537-538, 2005.

[5] A.V. Aho, J.E. Hopcroft and J.D. Ullman. On finding lowest common ancestors in trees. In ACM STOC 1973, pages 115-132, 1973.

[6] The Lemur Toolkit for Language Modeling and Information Retrieval. www.lemurproject.org/.

[7] DBLP Bibliography, www.informatik.uni-trier.de/ ley/db/.

[8] The Mondial dataset, http://www.cs.washington.edu/research/xmldatasets/.

[9] Z. Bao, T.W. Ling, B. Chen, et al. Effective XML Keyword Search with Relevance Oriented Ranking. In ICDE 2009, pages 517-528, 2009.

[10] Z. Bao, J. Lu, T.W. Ling, et al. Towards an Effective Xml Keyword Search. IEEE Transactions on Knowledge and Data Engineering, 2010, 22(8): 1077 - 92.

[11] Z. Liu and Y. Chen. Identifying meaningful return information for XML keyword search. In SIGMOD 2007, pages 329-340, 2007.

[12] L. Guo, F. Shao, C. Botev, et al. XRANK: Ranked Keyword Search over XML Documents. In SIGMOD 2003, pages 16-27, 2003.

[13] Y. Xu and Y. Papakonstantinou. Efficient LCA based keyword search in XML data. In EDBT 2008, pages 535-546, 2008.

[14] S. Cohen, J. Mamou, Y. Kanza, et al. XSEarch: A Semantic Search Engine for XML. In VLDB 2003, pages 45-56, 2003.

[15] G. Li, J. Feng, J. Wang, et al. Effective keyword search for valuable lcas over XML documents. In CIKM 2007, pages 31-40, 2007.

[16] Y. Li, C. Yu and H.V. Jagadish. Schema-Free XQuery. In VLDB 2004, pages 72-83, 2004.

[17] Y. Li, H. Yang and H.V. Jagadish. NaLIX: A generic natural language search environment for XML data. ACM Trans. Database Syst., 32(4): 30, 2007.

[18] G. Li, B.C. Ooi, J. Feng, et al. EASE: an effective 3-in-1 key-word search method for unstructured, semi-structured and structured data. In SIGMOD 2008, pages 903-914, 2008.

[19] J. Zhou, Z. Bao, T.W. Ling, et al. MCN: A New Semantics Towards Effective XML Keyword Search. In DASFAA 2009, pages 511-526, 2009.