ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
DOI: 10.17559/TV-20150314113111

INTERPRETING XML KEYWORD QUERY USING HIDDEN MARKOV MODEL

Xiping Liu, Changxuan Wan, Dexi Liu

Original scientific paper
Keyword search on XML database has attracted a lot of research interests. As XML documents are very different from flat documents, effective search of XML documents needs special considerations. Traditional bag-of-words model does not take the roles of keywords and the relationship between keywords into consideration, and thus is not suited for XML keyword search. In this paper, we present a novel model, called semi-structured keyword query (SSQ), which understands a keyword query in a different way: a keyword query is composed of several query units, where each unit represents query condition. To interpret a keyword query under this model, we take two steps. First, we propose a probabilistic approach based on a Hidden Markov Model for computing the best mapping of the query keywords into the database terms, i.e., elements, attributes and values. Second, we generate SSQs based on the mapping. Experimental results verify the effectiveness of our methods.

Keywords: *hidden Markov model (HMM); semi-structured keyword query (SSQ); XML keyword query*

Prikaz pretrage XML ključne riječi primjenom skrivenog Markovljevog modela

Izvorni znanstveni članak
Pretraživanje ključne riječi na XML bazi podataka privuklo je prilično zanimanja. Kako se XML dokumenti vrlo razlikuju od plošnih (flat) dokumenata, učinkovita pretraga XML dokumenata zahtijeva posebno razmatranje. Tradicionalni model vreće riječi (bag-of-words) ne uzima u obzir uloge ključnih riječi i odnos između ključnih riječi pa prema tome nije pogodan za XML pretragu ključne riječi. U ovom radu predstavljamo novi model, nazvan polu-strukturno pretraživanje ključne riječi (SSQ), koji podrazumijeva pretraživanje ključne riječi na različit način; to se pretraživanje sastoji od nekoliko cjelina pretrage i svaka cjelina predstavlja stanje pretrage (query condition). Za interpretaciju pretrage po tom modelu, potrebna su dva koraka. Prvo, predlažemo probabilistički pristup zasnovan na skrivenom Markovljevom modelu za izračunavanje najboljeg uklapanja traženih ključnih riječi u termine baze podataka, tj. elemenata, atributa i vrijednosti. Drugo, generiramo konstrukcije ključnih riječi (SSQs) na osnovu uklapanja. Eksperimentalni rezultati potvrđuju učinkovitost naših metoda.

Ključne riječi: *polu-strukturno pretraživanje ključne riječi; skriveni Markovljev model (HMM); XML pretraživanje ključne riječi*

1 Introduction

Keyword search, due to its simplicity and friendness, has been widely used and extended to search a variety of sources of information, such as relational database and XML documents [1, 2]. An XML document is composed of nested elements. The nested structure of XML documents poses great challenges to keyword search techniques, as the users are able to search XML documents through structure and text contents.

The unique characteristics of XML documents calls for a fresh look at and deep understanding of the keyword query. Existing XML keyword search methods are based on the "bag-of-words" model. In this model, a text unit (such as a paragraph or a document) is taken as the bag (multiset) of words, which means that the grammar and order of words are not taken into consideration. However, this model is too simple for XML keyword search.

Consider a query Q_1: "journal info system article expert".The query intention is to search for articles about "expert" in a journal named "info system". In an XML database, the answer may be an element labelled "article" nested in an element labelled "journal", where the "article" element contains "expert" in its text content, and the "journal" element has "info system" in its content. Obviously, it is not natural to view the query as a bag of words. First, the keywords in the query have different roles. The keywords "article" and "journal" are labels of elements, while "expert" and "info system" are just keywords in texts. Second, there exist different relationships between keywords. The keyword "expert" is more closely related to "article" than to "info" and "system", and "info system" has closer relationship with "journal" than with "article".

From this example we can see that the traditional bag-of-words model is not proper for XML keyword search, because it does not provide information about the structure of the query hidden in the XML keyword query. In this paper, we present a new model, called *semi-structured keyword query* (SSQ), to model a keyword query against an XML document. An SSQ is different from a keyword query in that it has structural information, and it is less strict compared with a structured query. The SSQ model is special in that it makes explicit the structure of the query. However, it is not straightforward to transform a keyword query into a query in SSQ form. In this work, we propose two steps to make the transformation. In the first step, we map the words in the keyword query into database terms, where each term is either from the schema vocabulary or from the texts of the database. As each word can be mapped to many terms, we develop a Hidden Markov Model-based probabilistic approach for interpreting the query keywords in terms of database terms. In the second step, we design an algorithm which takes a sequence of database terms as input, and outputs a set of SSQs. Once the SSQs are generated, it is possible to improve XML search results based on the SSQs, but that is beyond the scope of this paper.

To summarize, the following contributions are made in this paper:

1) We propose a novel way to analyse and interpret an XML keyword query. The approach makes explicit the structural information hidden in the keyword query, and transforms the query into a semi-structured keyword query (SSQ). The SSQ helps to get the semantics and intention of a keyword query.

2) We present a novel method that interprets a keyword query as an SSQ. The method uses a Hidden Markov Model to compute the best mapping of the query keywords to the database terms, i.e., elements, attributes and values.

3) We conduct a comprehensive set of experiments. Experimental results show that the proposed method is effective.

The rest of the paper is organized as follows. In section 2, the motivation of our method is introduced. In section 3, the proposed SSQ model is presented. We interpret the keyword query using a HMM model in section 4. Section 5 presents an algorithm generating SSQs. Experimental studies are discussed in section 6. Section 7 reviews related work and we finally conclude the paper in section 8.

2 Motivation

Due to the complexity of the XML documents, keyword search over XML documents is faced with many challenges. Understanding XML keyword query is the key to resolve these challenges.

(1) The roles of keywords are important.

Query keywords have different roles in a keyword query. Some keywords are intended to find the labels of elements, while others are used to match words in the texts.

Example 1. Fig. 1 shows an XML document about bibliography data such as journals, articles and authors. Each node represents an element. In the figure, some irrelevant nodes, e.g. volume, number, are not shown. This example will be used throughout this paper. Consider a query Q_1: "journal info system article expert". The query is used to search for articles about "expert" on journals named "info system". In this query, "journal" and "article" are different from "info system" and "expert": the former are expected to be labels of elements, while the latter are expected to be words in texts. Obviously, differentiating roles of the keywords is very important to get the desired results.

According to the roles of keywords, we divide the query keywords into two categories.

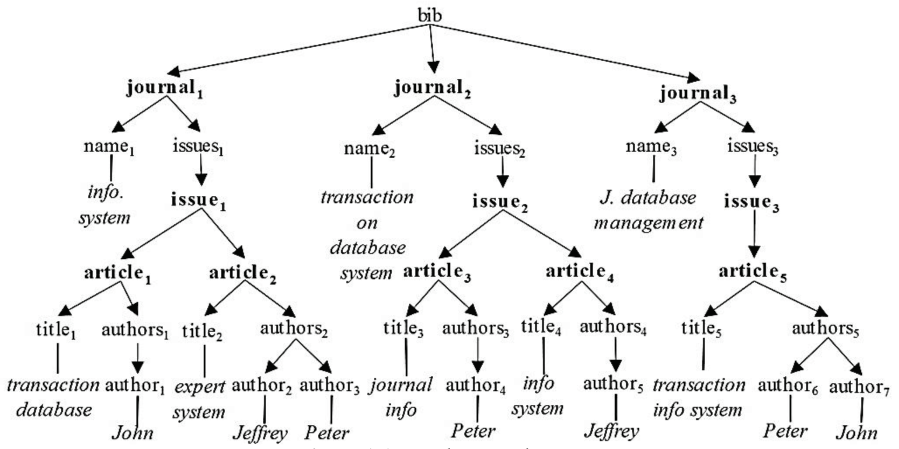


Figure 1 A sample XML document

Definition 1. Given a query keyword t in an XML keyword query Q, if t appears in the text, we say t is a content query term, or C-term in short; if t is the label of an element, t is a tag query term, denoted as T-term. The *role* of a query term is one of the two: C-term, T-term.

The role of a term can be easily inferred. We assume that an inverted list is built for each term, from which it is easy to know where the term appears.

(2) The relationships between keywords are important.

The connections between keywords are also different.

Given two different C-terms t_1 and t_2, there are two different possible relationships between t_1 and t_2: (a) t_1 and t_2 are expected to be from the same text node, and (b) t_1 and t_2 are not expected to be from the same text node. Consider the query Q_1: "journal info system article expert", "info" and "system" are expected to be from the same text node, while "info" and "expert" are not.

Given a T-term t_1 and a C-term t_2, there are also two different possible relationships between them: (a) t_2 is expected to appear under t_1, and (b) t_2 is not expected to appear under t_1. For example, in query Q_1: "journal info system article expert", "info" and "system" are expected to appear under "journal", and "expert" is expected to appear under "article", but "info" is not expected to appear under "article".

(3) The orders of query keywords are important.

When a user poses a keyword query, he will not order the keywords arbitrarily. In other words, the order of query keywords is important. For instance, in query Q_1: "journal info system article expert", "info system" and "expert" cannot be swapped, we cannot change the order of "journal" and "article" either.

We can see that a keyword query is not a casual mixture of keywords. There are some hidden rules beneath the surface. These rules form the structure of the query. If we can get the structure of a keyword query, we can understand the query better and thus can certainly get better query results. Consider the query Q_1: "journal info system article expert", the hidden structure of the query is that: (1) "info system" has a strong relationship with "journal", and "expert" is closely related to "article"; (2) the query is composed of two parts: "journal info system" and "article expert". These two parts cannot be mixed together.

In this paper, we propose a novel model to interpret a keyword query. The key point of the model is to make explicit the structure hidden in the query. Therefore, we call the model *semi-structured keyword query* (SSQ). The model is not a structured model, like SQL or XQuery, because the structure information is not so complete. For example, in the query Q_1: "journal info system article expert", the relationship between "journal" and "article" is unknown. We can see an SSQ as an immediate form between a keyword query and a structured query. It is less strict than a structured query, but has more structure than a keyword query. Obviously, a SSQ is less ambiguous than a keyword query, so if we can get SSQs from a keyword query, we are probably able to improve the search results.

In the following sections, we will present the SSQ model, and describe how to interpret a keyword query in terms of SSQ model.

3 The SSQ model

In this section, we present the SSQ model for the XMLkeyword query.

3.1 Definition

Definition 2. Given a keyword query Q, a query unit q of Q is a pair ⟨*context*, *cterms*⟩, where *context* is a sequence of T-terms, and *cterms* is a sequence of C-terms.

A query unit indicates a query condition, which states that certain query keywords (C-terms) should appear under certain elements (T-terms).

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. In certain query unit, there may not exist *context* or *cterms* part, in that case, we use NULL to denote null *context* or *cterms*. For example, the query unit *q*(NULL, info system) denotes a query condition that some text node should contain "info system". In another example "database transaction author", the query can be decomposed into two query units *q*(NULL, database transaction) and *q*(author, NULL). 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*. We say $Q_s$is a *semi-structured keyword query* (SSQ) derived from Q.

For example, a SSQ of Q_1: "journal info system article expert" can be represented as {*q*(journal, info system), *q*(article, expert)}.

A keyword query can be interpreted as different SSQs, each with different size and query intents. If an SSQ Q_s has *n* query units, we say the *size* of the SSQ is *n*, or it is *n*-sized. For example, Q_1: "journal info system article expert" can be interpreted as {*q*(journal, info system), *q*(article, expert)} or{*q*(NULL, journal info system), *q*(article, expert)}. Obviously, the two SSQs have different meaning. Consider the query "database transaction info system". If there is no T-term here, we can get three SSQs with size 2: {*q*(NULL, database), *q*(NULL, transaction info system)}, {*q*(NULL, database transaction), *q*(NULL, info system)}, and {*q*(NULL, database transaction info), *q*(NULL, system)}. Similarly, 3-sized SSQ also have three candidates, and one candidate exists for 4-sized SSQ and 1-sized SSQ, respectively. Thus we get a total number of eight SSQs from this query. Among these SSQs, some are more likely than others. It is not possible to handle all these SSQs. Instead, we try to find the SSQs that are most likely intended by the users.

3.2 Problem statement

Definition 3. An XML database D is a tuple <V, E, W, λ, *root*>, where V is the set of vertices in the database, and $E \in V*V$ is the set of edges in the database. $\lambda$is a mapping $V \rightarrow S$ that maps each vertice to a string, which is called the label of the vertice; *root* is a unique vertice in the database. W is the vocabulary consisting of the labels and terms in the database.

A *database term* refers to a label or a keyword in the texts of the database. We distinguish two subsets of the vocabulary W: the schema vocabulary W_S and the domain vocabulary W_D. The schema vocabulary consists of the labels of the vertices, and the domain vocabulary consists of the terms appearing in the texts of the database. Each element of the set is referred to as a database term.

Definition 4. A configuration *cfg*(Q) of a keyword query Q on a database D is a function from the keywords in Q to database terms in W. For each keyword t_i in Q, $cfg(t_i)$ is a term in the database vocabulary. Moreover, given two keywords t_i and t_j, if $t_i \neq t_j$, then $cfg(t_i) \neq cfg(t_j)$. In other words, a configuration maps each keyword in the original query to a distinct term in the database vocabulary.

We can see that a configuration is in fact an injective function.The reason is as follows.

First, each keyword in the original query addresses part of the query intention. That is, no keyword is redundant. We assume that there are no stop words or unjustified keywords in the query. Therefore, each keyword is intended to find an element of interest, i.e. keywords always have a correspondent database term.

Second, we assume that the query intention is very clear, so each keyword has a specific meaning in a configuration, i.e., it is mapped to only one database term.

Third, keyword queries are compact and users will not use two keywords to refer to one database term, i.e. it is not likely that multiple keywords are mapped to the same database term in a configuration.

Given a keyword query, we use SSQ to describe its possible semantics. Each such SSQ is referred to as an interpretation of the keyword query in terms of database terms.

Definition 5. An interpretation of a keyword query Q on an XML database D using a configuration *cfg*(Q) is a SSQ $QS=\{q_1, \ldots, q_n\}$ such that the following holds:

- for each T-term A in QS, $\exists k \in Q$ such that $cfg(k)=A$;
- for each C-term v in QS, $\exists k \in Q$ such that $cfg(k) \in W_D$, and $k=v$;
- for each k in Q, $cfg(k) \in terms(QS)$.

We can estimate the number of configurations of a keyword query.

Since each query keyword can be mapped into a T-term or a C-term, there are $|W|$ different mappings for each keyword. Since at most only one keyword can be mapped to a database term, for a query containing k keywords, there are $\frac{|W|!}{(|W|-k)!}$ possible configurations. Of course, not all interpretations generated by the configurations are equally meaningful. Some are more likely than others to represent the query intention. In the following sections, we will show how to effectively identify these meaningful interpretations using information about the occurrences and inter-dependencies of elements in the database.

4 Computing configurations using a HMM

A naive method to get the configuration is as follows. For each keyword in the original query, we find its best match in the database terms. The best match of a keyword is a database term where the mapping is more likely to happen. Then the configuration can be obtained by combining the best matches of keywords. This method considers the keywords in the query independently, and finds the mappings of keywords in isolation. However, in real cases, keywords are not independent. The mapping of a keyword is influenced by the mappings of previous keywords, and will influence the mappings of following keywords. In fact, the inter-dependencies among keywords are of fundamental importance to interpret the keyword query. The method thus is not applicable to real cases.

In light of the importance of the inter-dependencies between keywords, we need to consider the inter-dependencies when analysing the keyword queries. We model the matching process of keywords as a sequential process in which the order is determined by the keyword ordering in the query. In each step of the process, a single keyword is matched against a database term, where the result of previous steps is taken into account. The process has a finite number of steps, i.e. the length of the query. The whole process is stochastic as the matches of the same keyword are different in different queries. This is because a keyword can have different meanings in different queries, thus can be mapped to different database terms. The process can be modelled naturally by the Hidden Markov Model (HMM) which is a stochastic finite state machine.

4.1 HMM

The Hidden Markov Model (HMM) is a powerful statistical tool for modelling generative sequences where the system being modelled is assumed to be a Markov process with unobserved (hidden) states. HMMs have been widely applied in many areas such as signal processing, NLP (natural language processing), and so on.

The Hidden Markov Model is a finite set of states, each of which is associated with a (generally multidimensional) probability distribution. The transition probabilities refer to the probabilities of changing from one state to another state. In a particular state an outcome or observation can be generated, according to the associated probability distribution.

The definition of an HMM needs the following element

(1) N, the number of states in the model.

(2) M, the number of observation symbols in the alphabet.

(3) A set of state *transition probabilities*$A = \{a_{ij}\}$.

$$a_{ij} = p\{q_{t+1} = j | q_t = i\}, 1 \le i, j \le N$$

where q_t and q_{t+1} denotes the current and next state.

Transition probabilities should satisfy the following constraints.

$$a_{ij} \ge 0, 1 \le i, j \le N$$

and

$$\sum_{j=1}^{N} a_{ij} = 1, 1 \le i \le N$$

(4) $\{b_j(k)\}$,a probability distribution in each state.

$$b_j(k) = p\{o_t = v_k \mid q_t = j\}, 1 \le j \le N, 1 \le k \le \mathrm{M}$$

where v_k denotes the kth observation symbol in the alphabet, and o_t is the current parameter vector. $b_j(k)$ is usually called the *emission probability*. $b_j(k)$ should satisfy the following stochastic constraints.

$$b_j(k) \ge 0, 1 \le j \le N, 1 \le k \le \mathrm{M}$$

and

$$\sum_{k=1}^{M} b_j(k) = 1, 1 \le j \le N$$

If the observations are continuous, then we employ a continuous probability density function, instead of a set of discrete probabilities. In this case we approximate the probability density by a weighted sum of M Gaussian distributions,

$$b_j(o_t) = \sum_{m=1}^{M} c_{jm} \mathcal{N}(\mu_{jm}, \textstyle\sum_{jm}, o_t)$$

where, c_{jm} denotes the weighting coefficients, μ_{jm} is the mean vectors, and $\sum_{jm}$ is the Covariance matrices.

c_{jm} should satisfy the stochastic constrains,

$$c_{jm} \ge 0, 1 \le j \le N, 1 \le m \le M$$

and

$$\sum_{m=1}^{M} c_{jm} = 1, 1 \le j \le N$$

(5) The *initial state distribution*, $\pi = \{\pi_i\}$, where

$$\pi_i = p\{q_1 = i\}, 1 \le i \le N$$

Therefore we can just use the notation $\lambda = (A, B, \pi)$to denote an HMM with discrete probability distributions, while $\lambda = (A, c_{jm}, \mu_{jm}, \sum_{jm}, \pi)$ denotes one with continuous densities.

HMM makes the following assumptions.

(1)The Markov assumption

As described in the definition of HMMs, transition probabilities are defined as,

$$a_{ij} = p\{q_{t+1} = j | q_t = i\}, 1 \leq i,j \leq N$$

Therefore, we make the assumption that the next state is dependent only upon the current state. This is known as the Markov assumption.

(2) The stationarity assumption

This assumption assumes that state transition probabilities are independent of the actual time when the transitions take place. i.e.,

$$p\{q_{t1+1} = j | q_{t1} = i\} = p\{q_{t2+1} = j | q_{t2} = i\}$$

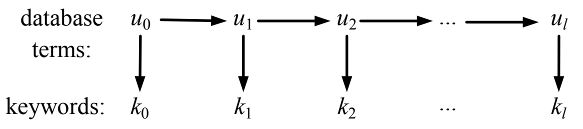
for any t_1 and t_2.

(3) The output independence assumption

This assumption states that current output (observation) is statistically independent of the previous outputs (observations).

4.2 Using HMM

In our context, we model the keywords as observations and each term in the database vocabulary as a state. The HMM used is illustrated in Fig. 2.



$$\begin{tikzcd} u_0 \arrow[r] \arrow[d] & u_1 \arrow[r] \arrow[d] & u_2 \arrow[r] \arrow[d] & \cdots \arrow[r] & u_l \arrow[d] \\ k_0 & k_1 & k_2 & \cdots & k_l \end{tikzcd}$$

Figure 2 The HMM model

Then, our problem is: given a sequence of observations $k_0, k_1, \cdots, k_l$, find the most likely state sequence, i.e. the sequence of database terms $u_0, u_1, \cdots, u_l$.

To infer using HMM, we need to know the HMM parameters, i.e. the transition probability distribution *A*, the emission probability distribution *B* and the initial state distribution *π*. These parameters are usually inferred from training data. In this work, we initialize the model parameters using the database structure so that query interpretation can be performed even without any training data.

Computing transition probabilities

The transition probabilities are computed based on the semantic relationships that exist between the database terms. We assume the existence of a database summary, where it summarizes the structure of the database, and for each path, it summarizes the keywords appearing under the path. An example of the summary is DataGuide+.

Then, the transition probabilities can be computed from the summary. We divide the transitions between database terms into three cases:

(1) Transition between T-terms. Given two T-terms u_i and u_j, the transition probability $p\{q_{t+1} = u_j | q_t = u_i\}$ or simply denoted $A(u_i, u_j)$ can be thought of as *navigation probability*, i.e. the probability of navigating to u_j from u_i. The navigation consists of several steps, and we assume that 1) each step is independent of previous steps, and 2) each step is a stochastic process. More specifically, suppose the navigation path from u_i to u_j is $u_i \to u_i+1 \to \ldots \to u_j$, then the navigation probability from u_i to u_j is computed as

$$p(u_i \dashrightarrow u_j) = p(u_i \to u_{i+1}) p(u_{i+1} \to u_{i+2}) \cdots p(u_{j-1} \to u_j)$$

$$p(u_i \to u_{i+1}) = \frac{freq(u_i \to u_{i+1})}{freq(u_i)}$$

where $freq(u_i \to u_{i+1})$ is the frequency of the edge from u_i pointing to u_j, and $freq(u_i)$ is the frequency of the node u_i in the summary.

The goal of the rules is to foster the transition between database terms close to each other. If the path-based distances between two T-terms are short, it means they are closely related and thus are very likely to be queried together.

(2) Transition between T-terms and C-terms. Given a T-term u_i and a C-term u_j, the transition probability $p\{q_{t+1} = u_j | q_t = u_i\}$ or $A(u_i, u_j)$ is computed as the probability of u_j occurring under the context of u_i. In formal,

$$A(u_i, u_j) = \frac{freq_{ui}(u_j)}{\sum_c freq_{ui}(c)}$$

where $freq_{ui}(u_j)$ is the frequency of u_j occurring under the context of u_i, and $\sum_c freq_{ui}(c)$ is the sum of all C-terms under the context of u_i. According to this formula, the more frequently a C-term appears in a certain T-term, the higher the transition probability between the two states.

(3)Transition between C-terms. Given two C-terms u_i and u_j, the transition probability $p\{q_{t+1} = u_j | q_t = u_i\}$ or $A(u_i, u_j)$ is taken as the similarity of the C-terms. The commonly used similarity measures such as edit distance measure can be employed here.

Computing emission probabilities

The emission probability is the conditional distribution of the observed variables from a specific state. Recall that $b_j(k) = p\{o_t = v_k \mid q_t = j\}$ denotes the emission probability at the state of q_t. We classify the emission probability into two categories as follows.

(1) The state q_t is a T-term. In this case, the emission probability is the probability of a keyword occurring under the element of q_t. Thus can be computed as

$$p\{o_t = v_k \mid q_t = j\} = \frac{freq_j(v_k)}{|KU(j)|}$$

where $KU(j)$ is the union of keywords contained in the texts in the element of j, and $freq_j(v_k)$ is the frequency of v_k occurring in the element of j.

(2) The state q_t is a C-term. In this case, the emission probability is the probability of seeing the keyword v_k following the C-term j. Therefore, we define the probability as the co-occurrence significance of v_k and j.

In order to compute the co-occurrence significance of two words k_i and k_j, we first observe the co-occurrence frequencies $freq(k_i, k_j)$, that is the number of times k_i and k_j occurring under the same text unit. They are then interpreted by a co-occurrence significance measure, given individual frequencies $freq(k_i)$ of k_i and $freq(k_j)$ of k_j. A number of measures have been developed in related work, and we use the following measure in this work.

$$sig(k_i, k_j) = \frac{freq(k_i, k_j) - \frac{freq(k_i)\cdot freq(k_j)}{n^2}}{\sqrt{\frac{freq(k_i)\cdot freq(k_j)}{n^2}}}$$

This measure is known as the z-score, where n is the number of text units in the document.

Then, when the state q_t is a C-term, the emission probability is

$$p\{o_t = v_k \mid q_t = j\} = sig(j, v_k)$$

Setting the initial state probabilities

At this step, we need to estimate the probability of each initial state. Recall that each state in our setting is a database term, a T-term or a C-term. We first discuss the initial state probability of a T-term state.

We understand the state probability of a T-term state as the importance of the T-term in the document, and then, we model the importance of a T-term in a way analogue to the PageRank algorithm. The basic idea is similar to the XRank system.

The objective importance of a T-term is computed based on the hyperlinked structure of XML documents. It is similar to PageRank, but is computed at the granularity of an element and takes nested structure of XML elements into account. It retains the original ranking semantics for HTML but is refined to considering the differences between computing ranks for HTML and XML documents.

An intuitive definition of PageRank algorithm shows how to measure the importance or influence of a web page on the Internet. The basic intuition is that, if a webpage is linked to by many other webpages, then it is important; if a webpage is linked by other important pages, then it is also important; if a webpage and many other pages are linked by an important page, then it shares the importance of the webpage with others.

The method of computing PageRanks [9] of HTML documents is to apply the following formula repeatedly.

$$PR(u) = (1-d) + d \sum_{v\in B(u)} \frac{PR(v)}{N(v)}$$

where

- $PR(u)$ is the PageRank of webpage u,
- $B(u)$ is the set of webpages that backlink to u,
- $PR(v)$ is the PageRank of page v which links to page u,
- $N(v)$ is the number of outbound links on page v and
- d is a damping factor which can be set between 0 and 1.

Now we extend the PageRank algorithm to XML documents. An XML database can be simply modelled as a directed graph $G=(N, CE, HE)$ where N refers to nodes representing XML elements and values, CE is the set of containment edges (parent-child edges) and HE is the set of hyperlink edges. The rank $e(v)$ of an element v is computed as follows.

$$e(v) = \frac{1-d_1-d_2-d_3}{N_d \times N_{de}(v)} + d_1 \sum_{(u,v)\in HE} \frac{e(u)}{N_h(u)} + d_2 \sum_{(u,v)\in CE} \frac{e(u)}{N_c(u)} + d_3 \sum_{(u,v)\in CE^{-1}} e(u)$$

where N_d is the total number of XML documents, $N_{de}(v)$ is the number of XML elements containing v, $N_h(v)$ is the number of out-going hyperlinks from document v, $N_c(u)$ is the number of sub-elements of u, CE^{-1} is the set of reverse containment edges, $d_1$is the probabilities of navigating through hyperlinks, d_2 and d_3 are the probabilities of navigating through forward and reverse containment edges, respectively.

Like the random walk interpretation of the PageRank algorithm [9], the formula also has a general interpretation in the context of random walks over XML graphs. Consider a random surfer over a hyperlinked XML graph. At each step, the surfer visits an element e, and chooses an action from the following choices: (1) he jumps to a random element within a random document with probability 1-d_1-d_2-d_3, (2) he follows a hyper-link from e with probability d_1, (3) he goes down a containment edge to one of e's sub-elements with probability d_2, and (4) he ascends to e's parent element with probability d_3. In this model, $e(v)$ is exactly the probability of finding the random surfer in element v.

While the ranking formula computes the rank of a specific element, what we need is a measure of the objective importance of a T-term. Note that a T-term is actually an element tag. Therefore, we aggregate the ranks of all elements labelled with a T-term t to get the importance of t. That is

$$p\{q_1 = i\} = \sum_{label(v)=i} e(v)$$

where $label(v)$ is the label of element v.

Now we discuss the initial state probability of a C-term state. Note that a C-term is actually a keyword in the database, therefore, the initial state probability of a C-term t is actually the importance or weight of a keyword t in the database.

Initially, we may still interpret the importance of a keyword t in the context of random walk through XML graph. To get a keyword t, a random surfer will first visit a leaf element e, and then pick the keyword t in the content of e. Therefore, the probability $prob(t \in e)$ of reaching t under e is

$$prob(t \in e) = prob(e) \times prob(t|e)$$

where the probability $prob(e)$ is the probability of reaching element e, and $prob(t|e)$ the probability of picking up t under element e. The $prob(e)$ can be taken as the importance of the element e in the graph, and $prob(t|e)$ can be computed using tf*idf measures, as follows.

$$prof(t|e) = \left(1 + log\ tf_{t,e}\right) \times log\frac{N}{df_t}$$

where $tf_{t,e}$ is the frequency of t in element e, N is the total number of content units in the database, df_t is the number of content units containing the keyword t.

Given a keyword t, it may appear under a lot of leaf elements. Therefore, a natural way to assess the importance of a keyword t is to aggregate the importance of t under all leaf elements containing the keyword. That is to say,

$$prob(t) = \sum_{t\in e} prof(t|e)$$

An alternative way to compute $prob(t)$ is to perform the steps above on the summary of the database, where contents with the same path are collapsed together. This version of $prob(t)$ can be computed much faster because the summary is very small in size.

4.3 Decoding algorithm

The aim of decoding is to discover the hidden state sequence that was most likely to have produced a given observation sequence. We use the List Viterbi algorithm [10], which is a generalization of the well-known Viterbi algorithm, to find the single best state sequence for an observation sequence.

5 Generating SSQs

In this section, we present an algorithm of generating SSQs based on the sequence of terms generated by HMM model.

The algorithm first decodes the keyword query into a sequence of database terms, then it generates a set of interpretations (SSQs) based on the database terms. Given a sequence of database terms, we can infer the SSQs based on some observations about query units and SSQ.

Observation 1. Given a keyword query Q, let Q_S be a SSQ derived from Q, and q be a query unit in Q_S, the C-terms in q (if any) are consecutive in Q.

Example 2. If a user wants to find articles about "expert" on an "info system" journal, he probably poses a query like "journal info system article expert" or "article expert journal info system", or "info system journal article expert" etc. Instead, he is unlikely to issue the query "journal info expert article system". That is to say, "info" and "system" will be kept together. The reason is that "info system" are from the same query unit, so they cannot be split up.

Observation 2. Given a keyword query Q, let Q_S be an SSQ derived from Q. For any query unit q in Q_S, the terms in q appear together in Q.

Observation 1 states that the C-terms in any query unit appear together in Q, now Observation 2 goes further to state that all terms in a query unit are together. In other words, C-terms in a query unit should neighbour the T-terms from the same query unit in Q.

Example 3. In the query "journal transaction database article xml search", we can decompose the query into two query units q(journal, transaction database) and q(article, xml search). In each unit, the T-terms appear together with the C-terms. For that query intention, one may also use the query "article xml search journal transaction database" to express the query intention; on the other hand, one would not pose the query "journal article xml search transaction database", where terms from the same query units are separated.

```
Algorithm 1: GenerateQWS
Input: a keyword query $Q=\langle t_1, \dots, t_n \rangle$, an XML document $D$
Output: a set $QS$ of QWSs
1  $QS = \varnothing$
2  $decode(t_1, \dots, t_n; tp_1, \dots, tp_n)$;
   //determine the sequence of terms using HMM, where $tp_i$ is the type
   (T-term or C-term) of the mapped term of $t_i$
3  segment $\langle t_1, \dots, t_n \rangle$ into groups $C_1, \dots, C_m$ according to $\langle tp_1, \dots, tp_n \rangle$
4  $Q_S$ = null //$Q_S$ is a QWS
5  foreach group $C$
6      make a new query unit $q$ containing terms in $C$, add $q$ into $Q_S$
7  $Queue = \{Q_S\}$ //$Queue$ is a queue containing QWSs
8  while $Queue$ is not empty
9      $Q_S = Queue.pop()$, $optimized = false$
10     foreach pair of query units $q_i$, $q_{i+1}$ in $Q_S$
11         if $Q'_S = merge(Q_S, q_i, q_{i+1})$ is not $null$
12             push $Q'_S$ into $Queue$, and set $optimized = true$
13     if $optimized = false$
14         $refine(Q_S)$, then add $Q_S$ into $QS$
Function $refine(Q_S)$
1  foreach query unit $q_i$ in $Q_S$
2      if $q_i$ is not valid
3          split $q_i$ into $q_{i_1}, \cdots, q_{i_k}$, such that each $q_{i_j}$ is valid
4          replace $q_i$ with $q_{i_1}, \cdots, q_{i_k}$ in $Q_S$
```

Figure 2 Algorithm of generating SSQs

The process of generating SSQs is sketched in Fig. 2. It first decodes the keyword query and gets the roles of the terms. The keyword query is first segmented into several groups according to the roles of the terms. Each group is composed solely of C-terms or T-terms. Then, a SSQ is constructed, where each query unit is made from a group. The SSQ is an initial raw SSQ, the algorithm then tries to derive more SSQs from the raw SSQ. It first puts the SSQ into a queue. For each SSQ in the queue, it tries to merge the neighbouring query units. A pair of query units can be merged if one has empty context and the other has empty C-terms. Every time a pair of neighbouring query units is successfully merged, a new SSQ is generated and pushed to the queue for further merge. If no query units in an SSQ can be merged, the resulting SSQ will be refined by the function *refine*, which refines a SSQ to make sure it is valid with respect to the observations. For each query unit q_i, the function checks if it is valid, if not, the function splits it up to multiple ones substituting for q_i. We use a running example to explain the process.

Example 4. Take query "journal transaction database article xml search" as an example. Suppose the "journal" and "article" are T-terms. The terms are first segmented into four groups: C_1 ("journal"), C_2 ("transaction database"), C_3("article") and C_4 ("xml search"). Based on these groups, a raw SSQ is constructed, i.e. {q(journal, NULL), q(NULL, transaction database), q(article, NULL), q(NULL, xml search)}. Then, the following SSQs will be generated in order: {q(journal, transaction database), q(article, xml search)}, {*q*(journal, NULL), *q*(article, transaction database), q(NULL, xml search)}. These queries are valid, so we finally translate the original query to three SSQs.

6 Experimental studies

In this section, we conduct a set of experiments to verify the SSQ model and the proposed methods.

6.1 Experimental setup

We have implemented the proposed method. We use the keywords inverted lists to store the occurrences of keywords. The inverted lists are built by Lemur toolkit [6]. All algorithms are implemented in C++. The experiments are conducted on a machine with Intel 2.30GCPU and 2G RAM running Windows.

We use two real datasets: DBLP 230M [7] and Mondial [8]. We test several keyword queries for each dataset. The queries are listed in Tab. 1. Queries DQ1-DQ10 are designed for DBLP dataset, MQ1-MQ10 for Mondial dataset. These queries exhibit different features and selectivity. Each query is actually transformed from a set of structured queries, thus the corresponding database terms can be inferred manually, and we can use them as the ground truth. For example, in DQ1: "journal ieeetran article pattern recognition", the desired sequence of database terms should be "journal(T) ieee(C) tran(C) article(T) pattern(C) recognition(C)", where "T" refers to a term from schema vocabulary, and "C" refers to a term from domain vocabulary.

Table 1 Query Set

No.	Query
DQ1	journal ieeetran article pattern recognition
DQ2	article title XML database
DQ3	article cluster web search results
DQ4	xml document index query
DQ5	journal vldb year 2004
DQ6	title search score rank
DQ7	title software journal ieee computer
DQ8	article author jimgray
DQ9	article image retrieval feedback
DQ10	article title database transaction
MQ1	country language french
MQ2	arab organization
MQ3	world trade organization
MQ4	city yorkcanada
MQ5	new york
MQ6	religion muslim
MQ7	republic government
MQ8	city Mexico city
MQ9	continent America country
MQ10	located_atsea

6.2 Experimental results

We report the experimental results in this section.

1) Effectiveness of HMM

First, we check if the sequences of database terms generated by the HMM model are correct. Note that the HMM model generates a set of sequences, each with a probability. We examine the top 3 sequences, ordered by their probabilities, and then examine if they are or include correct sequence. The result is shown in Fig. 3. We can see that most of the time, the sequences with best probabilities are exactly what we want. If we examine the top 2 sequences, we find that they all contain the desired sequence on DBLP dataset. For Mondial dataset, the top 3 sequences contain the desired sequence. From this set of experiments, we can see that the HMM model used to decode the keyword query is effective.

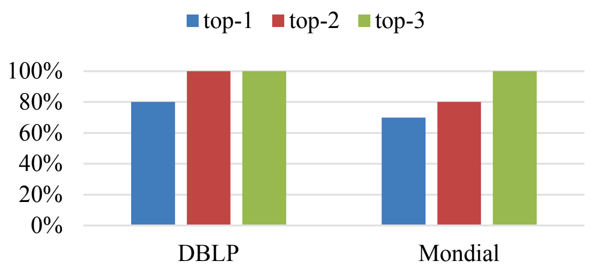


Figure 3 The effectiveness of the HMM model

2) Effectiveness of SSQ generation algorithm

Then, we check if the algorithm used to generate SSQ is effective, i.e., if it generates the desired SSQ. We use the correct sequence of database terms as input, and then check if the generated SSQs include desired SSQ. The measures we use in the experiments are precision and recall. The formula is as follows.

$$precision = \frac{number\ of\ desired\ QWS\ generated}{number\ of\ generated\ QWS}$$

$$recall = \frac{number\ of\ desired\ QWS\ generated}{number\ of\ desired\ QWS}$$

The result is shown in Fig. 4. The precision and recall are averaged over all queries on the datasets. For most of the query, there is only one desired SSQ, and they have been successfully collected, that's why the recall is 1 in both datasets. The precision is not high on DBLP dataset, because each configuration can generate several possible SSQs. This problem can be approached by using a ranking function to rank the SSQs, so that the most probable SSQs will be discovered. The ranking of the SSQs is not the topic of this paper.

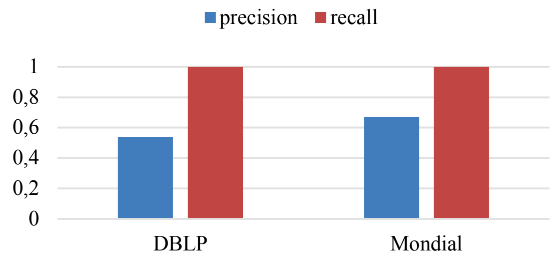


Figure 4 The effectiveness of the algorithm

7 Related work

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

As the semantics of an XML keyword query is vague, a lot of researches are devoted to the query semantics interpretation. Up to now, a number of query semantics has been proposed, most of them are based on LCA semantics. According to the LCA semantics, an answer is the lowest common ancestor of matches of query keywords. The basic LCA semantics is deficient, so many variants are proposed. XRank [12] and ELCA [13] connect keyword matches by the LCA nodes that contain at least one occurrence of all keywords, but it excludes the occurrences of keywords in sub-elements that already contain all keywords. XSEarch [14] is based on the concept of interconnection. Two keyword 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 can be found in [15]. In [4], Y. Xu et al propose the notion of the SLCA which is an extension of LCA. An SLCA is a smallest LCA. Here "smallest" means that it does not have further LCA nodes among their descendants. MLCA, proposed in [16], is very similar to SLCA but it has additional constraints on the node labels. Recently, [19] propose MCN to capture the relationships of the query keywords from XML document graph by considering reference relationship. XBridge [20] proposes an estimation-based approach to compute the promising result types for a keyword query.

In order to search effectively, many researchers try all efforts to analyse the XML documents, e.g. building various indexes, pre-computing a lot of statistics, etc. Recently, analysis of query has attracted some attention. XSeek [11] classifies query keywords into two categories: search predicates and return nodes, and inference rules are also proposed. Similarly, XReal also notices that a keyword can have different roles: it can appear both as an XMLtag name and as a text value of some other node. However, these studies do not go further to find out the structure hidden in the query. Nalix [17] was proposed to build a natural language query interface for a database. It supports a large class of natural language queries which can be translated into structured, e.g. XQuery, expressions. Our method differs from Nalix in that we interpret a keyword query in a lightweight way: we model a keyword query as a set of semi-structured query; we use HMM instead of NLP techniques to parse the keyword query. As a result, our method will be more efficient and suitable for keyword search.

Another research field is the ranking of XML search results. XRank [12] mimics PageRank in that it considers the XML document as a graph and computes the rank of elements based on the edges linking the element. XSEarch [14] employed the classical tf*idf formula in IR field. EASE [18] considers not only tf*idf-based IR ranking, but also structural compactness-based DB ranking. XReal [22] computes the confidence of node types as search for/search via nodes, and designs a novel XMLTF*IDF similarity ranking scheme. It also takes the co-occurrence of keywords into consideration. Our work is orthogonal to these works.

8 Conclusion

In this paper, we propose a new query model (SSQ) for XML keyword search, which interprets a keyword query as a set of semi-structured queries. We take two steps to infer the SSQs from a keyword query. First, we map the keyword query into configurations, i.e. sequences of database terms, where a database term is from schema vocabulary or domain vocabulary. We propose a probabilistic approach based on a Hidden Markov Model to compute the best mapping of the query keywords into the database terms. Second, we generate SSQs based on the configurations. Experimental results verified the effectiveness of our method.

Acknowledgements

This work was partially supported by Natural Science Foundation of China (NSFC) under Grant No. 61363010, 61173146, 61363039, 61662027 and 61262033, and the Science and Technology Project of Education Department of Jiangxi Province of China under Grant No. GJJ12732.

9 References

[1] Chen, Y.; Wang W.; Liu, Z.; Lin, X. Keyword Search on Structured and Semi-structured Data. // In Proceedings of SIGMOD 2009, pp. 1005-1010. DOI: 10.1145/1559845.1559966

[2] Coffman, J. An Empirical Performance Evaluation of Relational Keyword Search Techniques. // IEEE Transactions on Knowledge and Data Engineering. 26, 1(2014), pp. 1041-4347. DOI: 10.1109/TKDE.2012.228

[3] Manning, C. D.; Raghavan, P.; Schtze, H. Introduction to Information Retrieval. Cambridge University Press, 2008. DOI: 10.1017/CBO9780511809071

[4] Xu, Y.; Papakonstantinou, Y. Efficient Keyword Search for SmallestLCAs in XML Databases. // In Proceedings of SIGMOD 2005, pp. 537-538.

[5] Aho, A. V.; Hopcroft, J. E.; Ullman, J. D. On finding lowest common ancestors in trees. // In Proceedings of ACM STOC 1973, pp. 115-132.

[6] The Lemur Toolkit for Language Modelling and Information Retrieval. 2013. URL:www.lemurproject.org/.

[7] Ley, M. DBLP Bibliography. 2013. URL: www.informatik.uni-trier.de/ley/db/.

[8] Miklau, G. The Mondial dataset. 2002. URL: http://www.cs.washington.edu/research/xmldatasets/.

[9] Brin, S.; Page, L. The anatomy of a large-scale hypertextual Web search engine. // Computer Networks and ISDN Systems. 30, (1998), pp. 107-117. DOI: 10.1016/S0169-7552(98)00110-X

[10] Seshadri, N.; Sundberg, C.-E. W. List Viterbi decoding algorithms with applications. // IEEE Transactions on Communications. 42, 234(1994), pp. 313-323. DOI: 10.1109/TCOMM.1994.577040

[11] Liu, Z.; Chen, Y. Identifying meaningful return information for XML keyword search. // In Proceedings of SIGMOD 2007, pp. 329-340. DOI: 10.1145/1247480.1247518

[12] Guo, L.; Shao, F.; Botev, C. XRANK: Ranked Keyword Searchover XML Documents. // In Proceedings of SIGMOD 2003, pp. 16-27. DOI: 10.1145/872757.872762

[13] Xu, Y.; Papakonstantinou, Y. Efficient LCA based keyword search in XML data. // In Proceedings of EDBT 2008, pp. 535-546. DOI: 10.1145/1353343.1353408

[14] Cohen, S.; Mamou, J.; Kanza, Y. XSEarch: A Semantic Search Engine for XML. // In Proceedings of VLDB 2003, pp. 45-56. DOI: 10.1016/b978-012722442-8/50013-6

[15] Li, G.; Feng, J.; Wang, J. Effective keyword search for valuablelcas over XML documents. // In Proceedings of CIKM 2007, pp. 31-40.

[16] Li, Y.; Yu, C.; Jagadish, H. V. Schema-Free XQuery. // In Proceedings of VLDB 2004, pp. 72-83. DOI: 10.1016/b978-012088469-8.50010-3

[17] Li, Y.; Yang, H.; Jagadish, H. V. NaLIX: A generic natural language search environment for XML data. // ACM Trans. Database Syst. 32, 4(2007). pp. 30. DOI: 10.1145/1292609.1292620

[18] Li, G.; Ooi, B. C.; Feng, J. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. // In Proceedings of SIGMOD 2008, pp. 903-914. DOI: 10.1145/1376616.1376706

[19] Zhou, J.; Bao, Z.; Ling, T. W. MCN: A New Semantics Towards Effective XML Keyword Search. // In Proceedings of DASFAA 2009, pp. 511-526. DOI: 10.1007/978-3-642-00887-0_45

[20] Li, J.; Liu, C.; Zhou, R.; Wang, W. Suggestion of promising result types for xml keyword search. // In Proceedings of EDBT 2010, pp. 561-572. DOI: 10.1145/1739041.1739108

[21] Evert, S. The Statistics of Word Concurrences: Word Pairs and Collocations. // PhD thesis, University of Stuttgart, Stuttgart, Germany (2004).

[22] Bao, Z.; Ling, T. W.; Chen, B. et al. Effective XML keyword search with relevance oriented ranking. // In Proceedings of ICDE2009, pp. 517-528 DOI: 10.1109/icde.2009.16

Authors' addresses

Xiping Liu, Dr. in Computer Science
School of Information Technology,
Jiangxi University of Finance and Economics,
No. 169, East Shuanggang Road,
Nanchang 330013, Jiangxi, P. R. China
E-mail**:** lewislxp@gmail.com

Changxuan Wan, Prof. in Computer Science
School of Information Technology,
Jiangxi University of Finance and Economics,
No. 169, East Shuanggang Road,
Nanchang 330013, Jiangxi, P. R. China
E-mail: wanchangxuan@263.net

Dexi Liu, Dr. in Computer Science
School of Information Technology,
Jiangxi University of Finance and Economics,
No. 169, East Shuanggang Road,
Nanchang 330013, Jiangxi, P. R. China
E-mail: dexi.liu@163.com