http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (1): 51-60, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.51.60

Semantics Oriented Inference of Keyword Search Intention over XML Documents

Xiping Liu
School of Information Technology, Jiangxi University of Finance and Economics,
Nanchang 330013, China

Abstract: Keyword search is an effective paradigm for information discovery and has been introduced recently to query extensible markup language (XML) documents. Due to the existence of ambiguities in query semantics, effective keyword search of XML documents needs to infer search intention. In this study, a semantics oriented approach for keyword search intention inference over XML documents is proposed. The method infers search intention in two steps: in the first step, XML nodes are mapped into the concepts in ER model and the content features, occurrence features and reference features are utilized to classify the nodes into different categories. In the second step, query-dependent features are incorporated to refine search intentions. An efficient algorithm is also proposed to process keyword queries. Experimental results demonstrate the effectiveness of the proposed methods.

Key words: Extensible markup language, keyword search, search intention, semantics, keyword queries

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 (Chen *et al*., 2011).

XML keyword search returns query answers at the granularity of elements or XML nodes. Naturally, given an XML keyword query, a question arises as what kind of nodes are qualified as answers.

For example, an XML document representing a bibliography database is shown in Fig. 1, which contains information about journals, articles and authors. The structure of the document is shown in Fig. 2. In Fig. 1, some nodes that are not related to the analysis are hidden, such as volume, num, etc. This sample will be used throughout this study. Consider a query Q1: "article database transaction". The query intends to search for articles about “database transaction”. Due to the ambiguities in the query, it is not straightforward to capture the intention. According to the query intention, relevant article nodes should be returned to the user, e.g., node 0.0.2.0.3. Several methods have been proposed to infer answers for XML search, these methods fall into several categories. In the first category, answer types are predefined in queries (Li *et al*., 2004) or set by users (Dar *et al*., 1998; Bhalotia *et al*., 2002). These methods are not flexible enough. The second category is to use the Smallest Lowest Common Ancestor (SLCA) of query terms and its variants (Cohen *et al*., 2003; Guo *et al*., 2003; Li *et al*., 2004; Xu and Papakonstantinou, 2005; Li *et al*., 2007; Liu and Chen, 2007) as answers. According to SLCA semantics, the answers for Q1 are journal (0.1) and journal (0.2), which are not appropriate. The third category (Bao *et al*., 2009) infers answer types based on the overall statistics about terms. By the approach by Bao *et al*. (2009), the answer type of Q1 may be article, issue, journal or even issues, depending on statistics of the document. Thus, the method is not stable and does not always guarantee desired answers.

It is believed that desired answers have certain essential properties, e.g., being relevant, informative or meaningful and precise and these properties should be observed independent of users’ queries. Besides, there may be some explicit or implicit clues about search intentions in users’ queries, so information expressed in XML keyword queries should also be utilized to infer search intentions. The main motivation of this work is to consider the semantics of nodes and users’ queries and then infer search intention based on these semantics.

Specifically, the following contributions are made in this study:

- A query-independent method is proposed to infer search intentions. The key of this method is to analyze the semantics of different types of nodes. It analyzes the features of nodes in XML documents and proposes several rules based on these features to classify the nodes into different categories

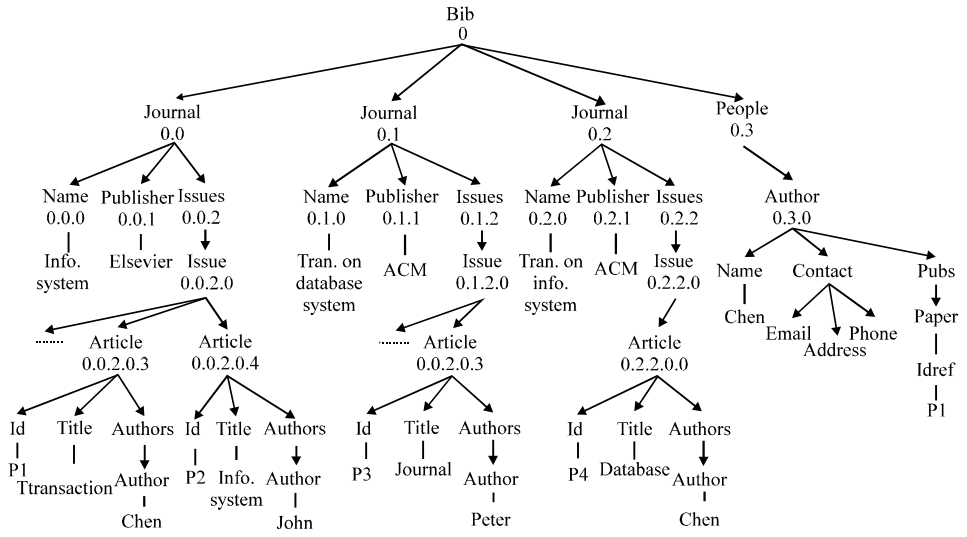

Fig. 1: A sample XML document

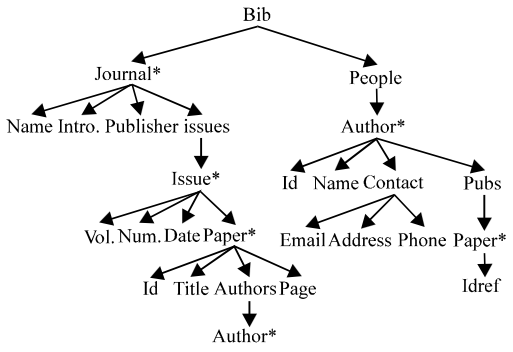

Fig. 2: The structure of the bib document

- Query-dependent features are incorporated to refine search intentions. The method considers the interests of node types as well as the relevance of nodes to the keyword query. It leads to meaningful, informative and relevant answers
- An efficient algorithm is proposed to implement the proposed methods and a comprehensive set of experiments is conducted to verify the effectiveness of the proposed methods. Experimental results demonstrate that the proposed method is effective, efficient and scalable

KEYWORD SEARCH INTENTION INFERENCE

Here, a new semantics-oriented method is proposed for keyword search intention inference. Two types of semantics are employed. The first one is the categories of XML nodes, which are independent of users' keyword query. The second type of semantics is query semantics, i.e., the information embodied in the keyword query. In the following sections, it will be discussed in detail how to utilize the semantics to infer search intentions.

QUERY-INDEPENDENT INFERENCE

It is observed that many nodes in an XML document are not qualified as answers regardless of users' queries. The main task of this section is to infer what types of nodes are fit for answers independent of queries.

At first, it is necessary to differentiate node types from nodes in XML documents. The type of a node is the label path from root to that node. For example, the type of node 0.0.2.0.3 in Fig. 1 is /bib/journal/issues/issue/ article. Without confusion, sometimes the simplified representations are used, e.g., the previous node type can be represented as /bib//article or //article. The type of an answer is an answer type. The objective of query-independent inference is to infer the set of node types that are proper to be answer types.

Information in XML documents is encoded in nested XML nodes. Nodes at different positions of XML tree have different importance, which reflects the semantic implication of the nodes. This study follows the idea of XSeek (Liu and Chen, 2007) to capture the importance of a node, i.e., it differentiates node types representing entities from node types representing attributes and generate answers based on the entities related to the matches of the query. The reasonableness of this idea has been justified (Liu and Chen, 2007).

In XSeek (Liu and Chen, 2007), some Naïve heuristics were proposed to infer node category (entity, attribute or connection). While these heuristics are effective in certain simply structured documents, they do not hold in many real XML documents, e.g., DBLP document. To get a profound understanding about the connection between XML nodes and ER concepts, various features of XML documents are explored in this study, including schema features, content features, occurrence patterns, etc., to infer the categories of XML nodes.

The basic building blocks of ER model are concepts such as entities, attributes and relationships. XML nodes can be classified into three categories based on their semantics:

- Attribute nodes, which correspond to attributes of entities. To avoid confusion with attributes of elements in XML documents, this type of nodes is denoted as A-nodes
- Entity nodes, which represent entities in real world. This type of nodes is denoted as E-nodes
- Wrapper nodes: nodes in this category (denoted as W-nodes) have different meanings:
    - It may describe a compound attribute or a weak entity, which wraps a set of closely related nodes. For example, contact node with sub-nodes email, address and phone can be viewed as a wrapper
    - It may describe an entity set or a multi-value attribute, which wraps a set of similar nodes to make sure that the tree hierarchy of an XML document is reasonably shaped and more understandable. For example, the issues nodes with multiple issue as children fall into this category. Another example is authors node. This kind of wrapper nodes may correspond to relationship between entities or may not

To classify XML nodes into the categories defined above, three kinds of information are utilized: structural properties, statistical information and referencing information. In detail, the following features are used:

- **Node property:** This property is used to distinguish nodes representing attributes of elements from nodes representing elements. The two kinds of nodes are denoted as attributes and elements, respectively
- **Content features:** These features first aim to distinguish nodes with text content and nodes having sub-elements. The former are called leaf nodes and the latter is called internal nodes. The features are also used to differentiate nodes with only one child from nodes having multiple children. For brevity, these two kinds of nodes are called simple nodes and compound nodes, respectively
- **Occurrence feature:** The feature identifies two different occurring characteristics:
    - **Single occurrence:** A node appears at most once in a context. For example, title appears only once in a paper
    - **Multiple occurrences:** A node appears multiple times in a context. For example, issue may appear multiple times under an issues context

Then, a classifier is built using decision tree method. Specifically, eight XML documents with different schemas are collected, which contain more than 100 node types. Three experts are asked to label the node types as entities or wrappers, other types not labeled are assumed to be attributes. Then a decision tree is trained using the labeled data. The generated rules are summarized as follows:

Rule 1: An attribute is an A-node
Rule 2: A leaf node is an A-node
Rule 3: A simple node is in the same category as its child
Rule 4: An internal node with multiple occurrences is an E-node
Rule 5: An internal node with single occurrence is an E-node or W-node
Rule 6: An internal node with single occurrence and repeated children is a W-node
Rule 7: If node n is an internal node with single occurrences and n has two parts of children: in one part there are some repeated children, in the other part there are some additional A-nodes, then n is an E-node

For example, in bib database, journal and issue are E-nodes according to rule 4, while issues and people are W-nodes according to rule 6.

References between nodes in XML documents are very popular. In Fig. 1, each paper under //people// pubs references an article under journal. Since some nodes can be viewed as entities, the referencing relationships can be viewed as the relationships between entities. For example, the reference of paper to article is analogous to the referencing relationship between author entity and article entity.

Now, the rules above can be augmented with a new heuristic rule:

Rule 8: If node a references another node b and b is an attribute of node c, then c is an E-node

In ER model, references always exist between entities, so it can be inferred that:

Rule 9: If node a references another node b, then the nearest possible E-node that is an ancestor of a is an E-node. For example, the author node in Fig. 1 is an E-node

To apply these rules, various features of XML documents needs to be collected, which can be obtained from document schemas. Note that classification is done at the schema level. So the inference just needs to be done once for each set of documents with the same schema and then this information can be stored in the database, which can be directly used when processing documents with the same schema.

After the categories of node types have been inferred, the entity types, i.e., the types of E-nodes, are taken as candidates of answer types.

QUERY-DEPENDENT INFERENCE

On the basis of answer type candidates, this section infers search intention by incorporating keyword queries. First, answer types are refined according to clues in the query. Not all entity types are relevant to user's query intention. For example, query Q2 "journal article peter chen" intends to find articles from "peter chen". An author with type //people/author should be deemed marginally relevant, if not irrelevant, to the query, although it is an E-node. To measure whether an entity type is relevant to the given query, a measure called interest is introduced.

Given a term t in a keyword query Q, if t appears in text content of 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.

Given a node type N and a keyword query Q, the interest of N to Q is defined as:

$$\text{Interest}(N,Q)=\begin{cases}\dfrac{|\text{DSLSet}(Q)\cap\text{DSL}(N)|}{|\text{DSLSet}(Q)|} & \text{if } T-\text{terms}(Q)\neq\varnothing\\ 1 & \text{if } T-\text{terms}(Q)=\varnothing\end{cases} \quad (1)$$

where, T-terms(Q) is the set of T-terms in Q, DSLSET(Q) = $U_{t\in T\text{-Terms}(Q)}$ DSL(T), DSL(N) is the labels of descendant-or-self instances of N. For example, let Q be the query Q2, N is /bib/people/author, then T-terms(Q) = {journal, article}, DSLSet(Q) = {journal, name, issues, issue, article,…}, DSL(*N*)) = {author, contact, address, name, pubs, …}, according to the definition above, interest(N, Q) = 2/15 which indicates the interest of node type /bib/people/author to the query is low.

However, how to incorporate the interests of node types to a query depends. In general, it is acceptable to choose node types of non-zero interests as answer types. In other cases, it may be more desirable to set some thresholds for interests. The experiments in this study simply choose node type of non-zero interests.

Second, answers are inferred from the instances of the identified answer types. Given a keyword query, if a node is of answer type and matches the query, then the node is an answer candidate. There are different definitions on the match of a keyword query. In this study, the match of a keyword query is defined as follows. Given a term t, node v is said to be a match of t if t appears in the text content of v (t is a C-term) or t appears as the tag of v (t is a T-term). Given a keyword query Q, node v is a match of Q if the following condition holds: if Q contains C-terms, v is an ancestor of matches of certain C-terms in Q; or else, v contains matches of certain T-terms in Q. Note that this definition implies an OR-semantics between terms in the query, that is, it does not require all terms in a query appear under an answer candidate. This semantics is consistent with traditional information retrieval.

An answer candidate is a possible but not necessary answer to the query. In the last step, the method chooses from the set of answer candidates the ones that are not redundant. An answer candidate may be redundant because it may overlap with another one. Given two nodes u and v, if u is an ancestor of v, then u overlaps with v. If two answer candidates overlap, both candidates may be meaningful or may not.

Consider query "info system". By this query, users want to get article or journal. Consider journal (0.0), note that there is an article (0.0.2.0.4) under node 0.0 about "info system". In this case, both nodes (0.0 and 0.0.2.0.4) are qualified to be answers, although they overlap. The reason lies in that the two nodes represent different matching semantics, i.e., node 0.0 appears as an answer candidate because it matches the query in name, whereas node 0.0.2.0.4 because of title. On the other hand, the issue (0.0.2.0) is also an answer candidate but not both 0.0.2.0 and 0.0.2.0.4 are answers because of their overlap. The rationale is that both nodes match the query due to the same fact: the title of article (0.0.2.0.4) matches the query.

From the example above, it can be seen that overlap is allowed between answers provided that overlapping

answers offer different matching semantics. To get a full understanding about this problem, the notion of witness node is defined.

Definition 1: Given a keyword query $Q = <t_1, \ldots, t_n>$, S_i is the list of nodes matching t_i $(1 \le i \le n)$, v is an answer (candidate), for each $n_i \in S_i$ contained by v, n_i is said a witness of v about t_i.

Definition 2: Given a query Q, $v_1, \ldots, v_n$ are answer candidates of Q and $W_1, \ldots, W_n$ are witness set of v_i $(1 \le i \le n)$, respectively. If $W_n \subset W_2 \cup \ldots \cup W_{n-1}$, v_n is said to be subsumed by $v_1, \ldots, v_{n-1}$.

The overlapping problem can be solved as follows: if an answer candidate v is subsumed by its descendant answer candidates, v is redundant and should be eliminated.

In summary, the semantics of answers inferred by the method presented in this study is as follows. Given a node v, v is an answer to a query Q if the following conditions hold: (1) v is of entity type, (2) the type of v is of some interest to Q, (3) v is a match of Q and (4) v is not redundant.

ALGORITHMS

This section presents algorithms that implement the proposed techniques. To speed up query processing, keyword inverted lists are built which retrieve a list of nodes in document order whose textual content contain the input keyword. Each node is assigned a Dewey label when the documents are parsed.

The two algorithms in Fig. 3 generate answers to a given query Q. Algorithm 1 is a naive algorithm which

Algorithms for keyword search intention inference

Input: keyword query Q; Output: ranked list of answers

Algorithm 1: Naïve algorithm

```
1  : T = getAnswerNodeTypes()
2  : Ans = getInstances(T)//let Ans be in decreasing order
3  : For each answer a in Ans
4  : If match (a, Q)
5  : a.witness = getWitness (a, Q)
6  : for each descendant d of a in Ans, push d into a.desAns
7  : DW = Merge (a.desAns)
8  : If DW ⊃ a.witness
9  : Remove a from Ans
10 : Else
11 : Remove a from Ans
12 : Return Ans
```

Algorithm 2: Group match

```
1  : T = getAnswerNodeTypes()
2  : For each C-terms ti in Q, Si = getMatches(ti)
3  : Ans = getAnswerNode(T, S1, S2, …, Sn)
4  : Return Ans
```

Function getAnswerNode (T, S_1, S_2,..., S_n)

```
1  : For each Si, let ci be the cursor associated with Si, ci=0
2  : Ans = ∅
3  : While (c1<S1.end()&c2<S2.end()…&cn<Sn.end())
4  : m = min (S1[c1], S2[c2],…, Sn[cn]), a = LAS (m, T)
5  : If (α! = null)
6  : a.witness = fetchWitness (a, c1,…, cn, S1, S2,…, Sn), Ans.push(a)
7  : For each E-node e, type (e)∈T, which is an ancestor of a
8  : Push a into e.desAns
9  : For each e∈Ans with non-empty desAns
10 : DW = merge (e.desAns)
11 : If DW ⊃ e.witness
12 : Remove e from Ans
13 : Return Ans
```

Function fetchWitness(a, c1,…,cn, S1, S2, …, Sn)

```
1  : WSet = ∅
2  : For each ci
3  : While (a is ancestor-or-self of Si[ci])
4  : WSet.push(Si[ci]), ci++;
5  : Return WSet
```

Fig. 3: Algorithms

processes a keyword query in a rather straightforward way. It first identifies answer types as discussed in section 2. Then for each potential answer a that is an instance of an answer type, if a matches Q, it obtains the witnesses of a. If a is an ancestor of some other answers which should be accessed before a, it attaches the descendant answers of a into a.desAns and merges their witnesses. If the descendant answers of a subsume a, it removes a from the answers list.

Algorithm 1 is inefficient in that it iterates through all instances of identified answer types, even though many of them are not relevant at all. To remedy this shortcoming, another algorithm is proposed in Algorithm 2. It differs with Algorithm 1 in that only relevant answers are accessed. Thus, it avoids a lot of useless computation. Algorithm 2 also begins with answer type identification. Then, the getMatches procedure retrieves nodes matching the C-terms in the query. After that, the algorithm calls the getAnswerNode function to obtain answers.

Function getAnswerNode is the key to Algorithm 2. It infers answers based on the given answer type T and matches of keywords. A cursor is attached to each set S_i and is initialized to the beginning of S_i. Then, it infer answers based on $S_i[c_i]$ $(1 \le i \le n)$. In each loop, it first finds the minimal match m of all $S_i[c_i]$ $(1 \le i \le n)$, then computes the lowest ancestor-or-self of m whose type is in T by calling LAS(m, T). If it gets a non-null LAS(m, T), say a, then a is an answer, witnesses of a are fetched from S_i $(1 \le i \le n)$ by calling function fetchWitness and a is pushed into answer set. If a is a descendant of node e which is also of answer type, it push a into the descendant answer set of e, i.e., e.desAns. After all raw answers are obtained, the algorithm prunes subsumed answers. The pruning process is similar to that in Algorithm 1. Note that the fetchWitness function in Algorithm 2 is different from getWitness in Algorithm 1: getWitness needs to retrieve inverted lists each time it is called, while fetchWitness only needs to scan several nodes from current positions of inverted lists. So fetchWitness is much more efficient than getWitness. Besides, cursors will be forwarded monotonously in fetchWitness, so all matches are scanned only once.

EXPERIMENTAL EVALUATION

Experimental setup: A prototype called XScore have been implemented which supports keyword search on XML documents. The keyword inverted lists are stored in an index build by Lemur toolkit. All experiments are run on a machine with Intel 2.30 G CPU and 2 G RAM running Windows XP. All algorithms are implemented in C++.

Table 1: Query set

No.	Query	No.	Query
DQ1	Author chen	MQ1	Hong Kong
DQ2	Title XML	MQ2	Hong Kong population
DQ3	Title joke	MQ3	York
DQ4	Keyword search	MQ4	Religion Muslim
DQ5	Keyword search rank	XQ1	Item trees
DQ6	Journal article iphone	XQ2	Namerica item shoe
DQ7	ps3 iphone	XQ3	Asia item tree
		XQ4	Item location united states
		XQ5	Person Murthy
		XQ6	Region Namerica item weapon

Both real and synthetic datasets are used. The real dataset is DBLP 230M (Ley, 2009) and synthetic dataset is XMark (XMark, 2009).

Several keyword queries are tested for each dataset. The queries are listed in Table 1. Queries DQ1~DQ7 are designed for DBLP dataset, MQ1~MQ4 for Mondial dataset and XQ1~XQ6 for XMark. These queries exhibit different features and selectivity. Note that queries DQ3, DQ6 and DQ7 are designed to test the extreme case when query has very few answers. Some additional queries are used in some experimental sections and they will be pointed out later.

The following aspects are evaluated in experiments: the quality of the answers, retrieval effectiveness measured by precision and recall and efficiency as well as scalability.

The main competitor of the proposed method is XReal (Bao *et al*., 2009) and XSeek (Liu and Chen, 2007) which are closest to this work. XSeek is also typical of SLCA-based method in evaluating effectiveness in search intention inference.

EXPERIMENTAL RESULTS

Answer semantics: The first set of experiments is to test the effectiveness of the proposed method of inferring search intentions by examining semantics of answers. The algorithms in XReal, XSeek and Section 4 are run on the datasets to collect all answers to the listed queries and then answer types are summarized. The inferred answer types are listed in Table 2. Only the results that some differences exist among the methods are shown.

As shown in Table 2, compared with XReal and XSeek, the answer types inferred by XScore are closest to user's search intention. The problem with XReal and XSeek is that, they may get answers of low quality. While XSeek usually selects tiny nodes which are not informative enough as answers, XReal is more inclined to pick big nodes which are usually overwhelming (DQ6, XQ2, XQ5 and XQ6 for XReal and DQ1, MQ1~MQ3 for XSeek). Besides, XReal tends to miss many answers since it cannot obtain all answer types. Also note that for DQ7

Table 2: Inferred answer types

Query	Intention	XScore	XReal	XSeek
DQ1	//article, //inproc//proc, //book	//article, //inproc//proc, //book	//inproc	//author
DQ2	all	all	//inproc	all
DQ4	//article, //inproc//proc, //phdthesis	//article, //inproc//proc, //phdthesis	//inproc	//article//inproc//proc
DQ6	//article	//inproc	/dblp	/dblp
DQ7	//inproc	//inproc	none	none
MQ1	//city	//city	//country	name
MQ2	//city	//city	//city	population
MQ3	//city, //province	//city, //province	//province	name
MQ4	//country	//country	//country	religion
XQ1	//asia/item, //eu/item//na/item, //au/item	//asia/item, //eu/item, //na/item//au/item/mail, //eu/item/mail//annotation	//eu/item//na/item	//asia/item, //eu/item//na/item, //au/item
XQ2	//na/item	//na/item, //aunotation	/site	//na
XQ3	//asia/item	//asia/item, //eu/item, //na/item//au/item/mail, //eu/item/mail	//na/item	//asia, //eu/item//na
XQ4	//item	//item, //person/address	//eu/item, //na//item	//item
XQ5	person	person	/site	person
XQ6	//na/item	//eu/item, //na/item//annotation, //annotation	/site	//regions

all: //article, //inproc, //proc, //phdthesis, etc.
inproc: In proceedings, proc: Proceedings, au: Australia, na: Namerica, eu: Europe

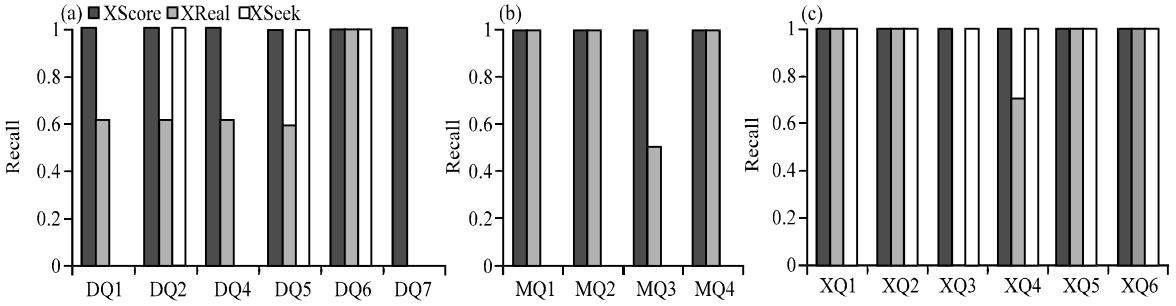

Fig. 4(a-c): Recall on datasets, (a) DBLP, (b) Mondial and (c) XMark

both XReal and XSeek cannot figure out any answers. The reasons are that XSeek adopts an AND-semantics while XReal also asks that at least all keywords appear in the collection which does not hold in this case. Another point not reflected in Table 2 is that, XReal is not stable, the inferred answer types are easily influenced by the size of the document, because it heavily depends on the statistics in the document.

The problem with the proposed method is that, XScore tends to return more answer types than desired which degrades the precision. This problem can be solved by effective top-k query processing techniques which will be probed in later work.

Retrieval effectiveness: The retrieval effectiveness is measured by precision and recall borrowed from IR field. Precision measures the percentage of the output nodes that are desired, recall measures the percentage of the desired nodes that are output. The set of relevant nodes is obtained by running the schema-aware XML query (in XQuery).

As can be seen in Fig. 4 and 5, XScore keeps almost perfect recall and reasonably good precision, while performances of XReal and XSeek are not stable. The dissatisfactory performances of two competitors are due to their improperly inferred answer types. For example, for XReal, since it misses some answer types for DQ1, DQ2, DQ4 and DQ5, the recall degrades; while for XQ2, XQ5 and XQ6, since it gets some huge answers, the precision goes down. The case for XSeek is even worse, since it often collects some tiny nodes as answers. XScore gets relatively low precisions for XQ3, XQ4 and XQ6, the reason is that it collects some irrelevant answers. Nevertheless, this problem can be approached by effective top-k techniques.

Efficiency: The set of experiments in this section evaluates the efficiencies of proposed algorithms. In these experiments, the Naïve algorithm is used as baseline and the improvement in Algorithm 2 GroupMatch (section 5) is measured. The same set of queries is run under each algorithm and the total processing time includes time consumed in processing queries and inferring answers.

Figure 6 shows the processing time for various queries using different algorithms, where GM stands for GroupMatch in Algorithm 2 and Naïve denotes the Naïve

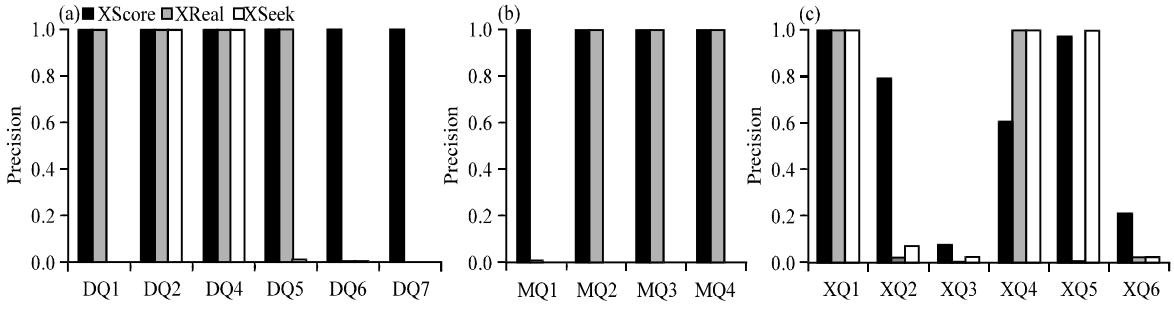

Fig. 5(a-c): Precision on datasets, (a) DBLP, (b) Mondial and (c) XMark

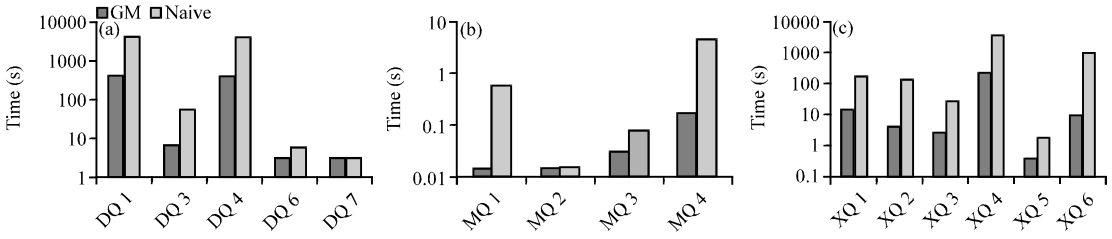

Fig. 6(a-c): Processing time, (a) DBLP, (b) Mondial and (c) XMark

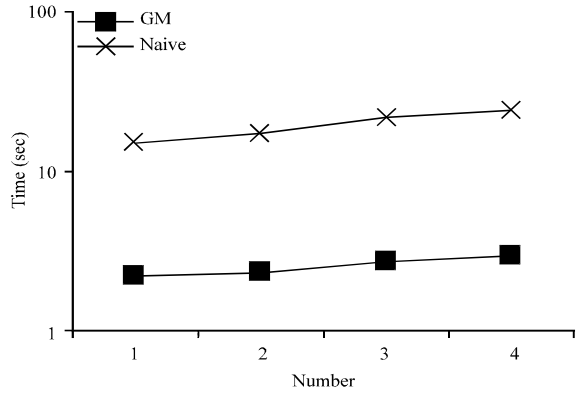

Fig. 7: Processing time w.r.t No. of keywords

algorithm in Algorithm 1. The datasets are DBLP 230M, Mondial 1.3M and XMark 113M. As expected, the GroupMatch algorithm outperforms naive algorithm for all queries. In fact, an improvement up to 40 times faster on performance is observed for some queries. Such advantage is extended in XMark, that is because nodes are nested deeply in that document which results in more time consumption according to Naïve algorithm.

Scalability: The scalability of various algorithms is tested over two parameters: number of keywords in the query and document size.

As discussed earlier, the number of answers mainly depends on C-terms in the query. This set of experiments focuses on C-terms and varies the number of C-terms in the queries. Specifically, four keyword queries with 1 to 4 C-terms are constructed and the queries are posed against DBLP 12M document. The queries are constructed as follows: randomly select a title from DBLP document, remove the stop words from the title and then get some candidate keywords. Then four keywords are randomly picked from candidates and four queries are constructed which are composed of 1 to 4 keywords, respectively. The four queries are executed three times and the average processing time is recorded. The result is shown in Fig. 7. It can be seen that both algorithms scale well w.r.t number of keywords but GroupMatch algorithm again outperforms the Naïve algorithm.

The efficiencies of algorithms are also tested on DBLP and XMark with different document sizes. As for DBLP, different portions of original document are extracted to get several documents with size from 1.3 M to 94 M. Variants of XMark are obtained by running the XML generator provided by (XMark, 2009) with different factors (.01, 0.1, 0.2, 0.5, 1). The queries used are DQ4 for DBLP and XQ2 for XMark. Results are shown in Fig. 8a, b. Once again, significant improvement gained by GM over Naive is observed on Fig. 8a. It is shown that GM

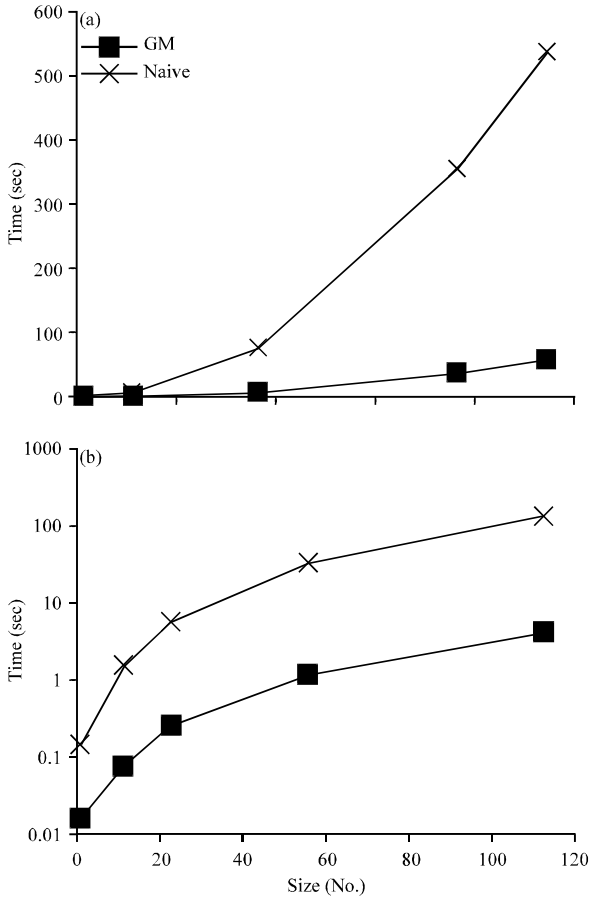

Fig. 8(a-b): Processing time w.r.t document, (a) DBLP and (b) XMark size

grows at a rather slow rate. In contrast, the performance of Naive degrades greatly as the size of document increase.

RELATED WORK

Extensive works have been done on keyword search intention inference and several methods have been proposed which can be classified into one of the following categories:

Manual methods: The methods in this category (Dar *et al*., 1998; Bhalotia *et al*., 2002) ask the domain experts to label some logically integral nodes as answer types based on domain knowledge and schema information. These methods guarantee that the recall is perfect but the precision is usually very low. What is more, it is labor intensive to label manually.

LCA-base methods: LCA (Lowest Common Ancestor) is a concept borrowed from the tree theory. Given a keyword query $t_1,\ldots, t_k$, v is an LCA of $t_1,\ldots, t_k$ if v is a common ancestor of $n_1,\ldots, n_k$ (n_i is a match of t_i, $1 \le i \le k$) and there is not another common ancestor of $n_1,\ldots, n_k$ under v. LCA is a good answer candidate. As LCAs of a keyword query may overlap with each other, many variants of LCA have been proposed, such as SLCA (Xu and Papakonstantinou, 2005; Liu and Chen, 2007), ELCA (Guo *et al*., 2003) and VLCA (Li *et al*., 2007). These variants are based on LCA but enhance the semantics of LCA, thus have the potential to improve the precisions. Even with the variants of LCA as answers, it is still possible to get some answers that are not so meaningful. Inspired by the entity-relationship model, XSeek (Liu and Chen, 2007) imposes more restrictions on the answers, requiring that the answers should represent entities (or master entities) and presents some naive heuristics to identify the categories of nodes in XML documents. However, these rules are so simple that they are not applicable to many cases, even on the simple document such as DBLP.

Statistics-based methods: XReal (Bao *et al*., 2009) infers the types of target nodes based on the query and statistical information and then infers the answers. The type of a node is the prefix path from the root to the node. Given a keyword query, the method computes the confidence of a node type T as query target using a function, then the node types with the greatest confidence score are chosen as the answer types. The statistics-based method is inclined to the node types with more instances which leads to missing answers. What is more, the method is not mature enough.

The method proposed in this paper takes the node categories and relationships, i.e., node semantics, into consideration; what is more, it also analyzes the query and incorporates the query semantics. As the method incorporates more semantics than others, it leads to meaningful, relevant and specific answers.

CONCLUSION

In this study, a semantic oriented approach for search intention inference in XML keyword search is proposed. The approach makes use of semantics of nodes as well as queries to infer answers. As for the semantics of nodes, it analyzes various explicit and implicit features to classify the nodes into several categories, then chooses answer types among these categories. On the other hand, the approach computes the interest of a node type as the confidence of the type as an answer type. It also utilizes the witness of an answer candidate to remove redundancy. An efficient algorithm is proposed to implement the proposed techniques and comprehensive experiments are conducted to verify the effectiveness of

the proposed methods. Part of the future work is to investigate effective top-k query processing techniques for XML keyword search.

REFERENCES

Bao, Z., T.W. Ling, B. Chen and J. Lu, 2009. Effective XML keyword search with relevance oriented ranking. Proceedings of the 25th International Conference on Data Engineering, March 29-April 2, 2009, Shanghai, China, pp: 517-528.

Bhalotia, G., A. Hulgeri, C. Nakhe, S. Chakrabarti and S. Sudarshan, 2002. Keyword searching and browsing in databases using BANKS. Proceedings of the 18th International Conference on Data Engineering, February 26-March 1, 2002, San Jose, CA., USA., pp: 431-440.

Chen, Y., W. Wang and Z. Liu, 2011. Keyword-based search and exploration on databases. Proceedings of the 27th International Conference on Data Engineering, April 11-16, 2011, Hanover, Germany, pp: 1380-1383.

Cohen, S., J. Mamou, Y. Kanza and Y. Sagiv, 2003. XSEarch: A semantic search engine for XML. Proceedings of the 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany, pp: 45-56.

Dar, S., G. Entin, S. Geva and E. Palmon, 1998. DTL's dataspot: Database exploration using plain language. Proceedings of the 24th International Conference on Very Large Data Bases, August 24-27, 1998, New York, USA., pp: 645-649.

Guo, L., F. Shao, C. Botev and J. Shanmugasundaram, 2003. XRANK: Ranked keyword search over XML documents. Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, June 9-12, 2003, San Diego, CA., USA., pp: 16-27.

Ley, M., 2009. DBLP-some lessons learned. Proceedings of the VLDB Endowment, August 24-28, 2009, Lyon, France, pp: 1493-1500.

Li, G., J. Feng, J. Wang and L. Zhou, 2007. Effective keyword search for valuable LCAs over XML documents. Proceedings of the 16th ACM Conference on Information and Knowledge Management, November 6-10, 2007, Lisbon, Portugal, pp: 31-40.

Li, Y., C. Yu and H.V. Jagadish, 2004. Schema-free XQuery. Proceedings of the 13th International Conference on Very Large Data Bases, August 31-September 3, 2004, Toronto, Canada, pp: 72-83.

Liu, Z. and Y. Chen, 2007. Identifying meaningful return information for XML keyword search. Proceedings of the ACM SIGMOD International Conference on Management of Data, June 12-14, 2007, Beijing, China, pp: 329-340.

XMark, 2009. XMark-an XML benchmark project. http://www.xml-benchmark.org/

Xu, Y. and Y. Papakonstantinou, 2005. Efficient keyword search for smallest LCAs in XML databases. Proceedings of the ACM SIGMOD International Conference on Management of Data, June 14-16, 2005, Baltimore, MD., USA., pp: 537-538.