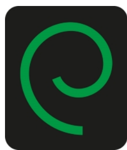
Emerald Insight

International Journal of Web Information Systems

# Effective keyword query structuring using NER for XML retrieval

Abubakar Roko Shyamala Doraisamy Azrul Hazri Jantan Azreen Azman

## Article information:

To cite this document:
Abubakar Roko Shyamala Doraisamy Azrul Hazri Jantan Azreen Azman , (2015),"Effective keyword query structuring using NER for XML retrieval", International Journal of Web Information Systems, Vol. 11 Iss 1 pp. 33 - 53
Permanent link to this document:
http://dx.doi.org/10.1108/IJWIS-06-2014-0022

Downloaded on: 09 November 2016, At: 02:04 (PT)
References: this document contains references to 19 other documents.
To copy this document: permissions@emeraldinsight.com
The fulltext of this document has been downloaded 182 times since 2015*

## Users who downloaded this article also downloaded:

(2015),"Design of interactive conjoint analysis Web-based system", International Journal of Web Information Systems, Vol. 11 Iss 1 pp. 17-32 http://dx.doi.org/10.1108/IJWIS-04-2014-0011

(2015),"Relevance status value model of Index Islamicus on Islamic History and Civilizations", International Journal of Web Information Systems, Vol. 11 Iss 1 pp. 54-86 http://dx.doi.org/10.1108/IJWIS-06-2014-0024

Access to this document was granted through an Emerald subscription provided by emerald-srm:563821 []

## For Authors

If you would like to write for this, or any other Emerald publication, then please use our Emerald for Authors service information about how to choose which publication to write for and submission guidelines are available for all. Please visit www.emeraldinsight.com/authors for more information.

## About Emerald www.emeraldinsight.com

Emerald is a global publisher linking research and practice to the benefit of society. The company manages a portfolio of more than 290 journals and over 2,350 books and book series volumes, as well as providing an extensive range of online products and additional customer resources and services.

Emerald is both COUNTER 4 and TRANSFER compliant. The organization is a partner of the Committee on Publication Ethics (COPE) and also works with Portico and the LOCKSS initiative for digital archive preservation.

*Related content and download information correct at time of download.

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

The current issue and full text archive of this journal is available on Emerald Insight at:
www.emeraldinsight.com/1744-0084.htm



# Effective keyword query structuring using NER for XML retrieval

Abubakar Roko, Shyamala Doraisamy, Azrul Hazri Jantan and Azreen Azman
*Department of Multimedia, University Putra Malaysia, Serdang, Malaysia*

Received 5 June 2014
Revised 11 September 2014
9 October 2014
Accepted 28 October 2014

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

## Abstract

**Purpose** – The purpose of this paper is to propose and evaluate XKQSS, a query structuring method that relegates the task of generating structured queries from a user to a search engine while retaining the simple keyword search query interface. A more effective way for searching XML database is to use structured queries. However, using query languages to express queries prove to be difficult for most users since this requires learning a query language and knowledge of the underlying data schema. On the other hand, the success of Web search engines has made many users to be familiar with keyword search and, therefore, they prefer to use a keyword search query interface to search XML data.

**Design/methodology/approach** – Existing query structuring approaches require users to provide structural hints in their input keyword queries even though their interface is keyword base. Other problems with existing systems include their inability to put keyword query ambiguities into consideration during query structuring and how to select the best generated structure query that best represents a given keyword query. To address these problems, this study allows users to submit a schema independent keyword query, use named entity recognition (NER) to categorize query keywords to resolve query ambiguities and compute semantic information for a node from its data content. Algorithms were proposed that find user search intentions and convert the intentions into a set of ranked structured queries.

**Findings** – Experiments with Sigmod and IMDB datasets were conducted to evaluate the effectiveness of the method. The experimental result shows that the XKQSS is about 20 per cent more effective than XReal in terms of return nodes identification, a state-of-art systems for XML retrieval.

**Originality/value** – Existing systems do not take keyword query ambiguities into account. XKSS consists of two guidelines based on NER that help to resolve these ambiguities before converting the submitted query. It also include a ranking function computes a score for each generated query by using both semantic information and data statistic, as opposed to data statistic only approach used by the existing approaches.

**Keywords** Managing and storing XML data, Indexing and retrieval of XML data, Metadata and ontologies

**Paper type** Research paper

## 1. Introduction

A more effective way to extract information from XML data is to use structured queries. These queries are formed using a query language such as XPath or XQuery. They retrieve precise answers because the queries contain specifications of what to return and where in the document to find it. However, using query languages to express queries proves to be difficult for most users because this requires learning a query language and knowledge of the underlying data schema. On the other hand, the success of Web search

International Journal of Web Information Systems
Vol. 11 No. 1, 2015
pp. 33-53
© Emerald Group Publishing Limited
1744-0084
DOI 10.1108/IJWIS-06-2014-0022

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

engine has made many users to be familiar with keyword search and, therefore, they prefer to use a keyword search query interface to search XML data. Using keyword query a user does not need to learn a query language and/or know the schema of the underlying documents. At this point, a question arises: Can we relegate the task of generating structured queries to an IR system while retaining the simple keyword search query interface?

To answer this question, many systems have been proposed (Li *et al.*, 2009; Petkova *et al.*, 2009; Li *et al.*, 2010; Hummel *et al.*, 2011). These systems relegate the task of generating structured queries from a user to themselves. This task is called query structuring (Li *et al.*, 2009) and such systems are called query structuring systems. Using knowledge of the XML schema, data statistics and heuristic, these systems convert a given keyword query to a set of structured queries. The challenges faced by these systems are how to find the return nodes and the search via nodes (i.e. predicates) from a keyword query due to keyword query ambiguities. These challenges are exacerbated by two ambiguities:

1. a keyword can refer to an element or data value of an element;
2. a keyword can appear as data value of different elements carrying different meanings (Bao *et al.*, 2010).

The example in Figure 1 contains the keyword *Access*, which occurs both in *title* and *author* node. A valid question that shoots is, if a query contains this keyword, is the user referring to that of the *title* node or the *author* node? These ambiguities cause keyword queries to be less effective and, hence, the need to develop systems that resolve these ambiguities. Existing systems do not take keyword query ambiguities into account. Apart from this problem, the other problems are they require users to provide structural hints in their input keyword queries even though their interface is keyword base and

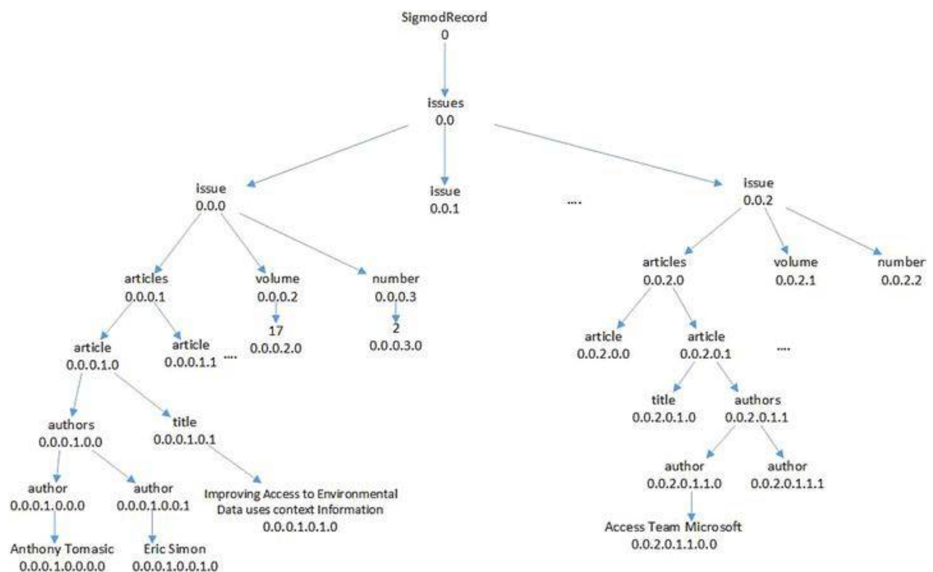


**Figure 1.**
A portion of Sigmod XML document

how to select the best generated structure query that best represents a given keyword query.

For example, Li *et al.* (2009) developed a system called XBridge that can identify the contexts (i.e. predicates) and types of return nodes from a given keyword query. The contexts and returns nodes are then use to compute a set of effective structured queries. Also, the study (Petkova *et al.*, 2009) developed a method for generating structured queries from a given user keyword query like a search problem. The top k-structured queries are selected as the best interpretation of the user keyword query. However, the problem with these systems is that each requires users to include structural hint in the input keyword query. That is, the systems restrict the way users would pose their queries and hence not user friendly. Also, the ranking formulae used by Petkova *et al.* (2009) to give score to each generated query does not take into consideration the semantic relationship between the contexts and return nodes that form a structured query.

In a study, Hummel *et al.* (2011) proposed a system called StruX, which does not dictate the way a user would pose his/her queries. Given a keyword query, the system first breaks the query into a sequence of segments. A segment consists of one or more keywords. Thereafter, pairs of segments are combined to form what is called "segment combinations". The segment combinations are then labelled with elements from the target database forming a list of a predicates. Predicates are used to restrict the type of return nodes. Also, based on an inference and document type definition (DTD) of an underlying XML document (also used in Liu *et al.*, 2007), a list of real-world entities call entity nodes in the data is generated. These entity nodes serve as return nodes. The list of return nodes and that of predicates are combined to compute a set of ranked structured queries. However, StruX requires the DTD of the underlying XML database to compute the entities in the data. The query processing then requires the presence of DTD, which is sometimes not available. Another drawback with StruX is that the formation of segment combination and labelling a combination with *any* elements in XML data could lead to unnecessary computation of element/segment statistics. This is because some segments' keywords can never be found in data value of some nodes (Zeng *et al.*, 2013).

To solve aforementioned problems, we propose XKQSS, a query structuring technique that allows users to submit a schema independent keyword queries and put keyword query ambiguities into account during query structuring. Specifically, given a keyword query, XKQSS first use the named entity recognition (NER) system to categorize the query keywords into different categories of named entities. Because, according to Guo *et al.* (2009), about 71 per cent of keyword queries contains named entities and finding them could help identify user search intention. It then extracts any consecutive sequence of keywords that describes a person or an organization named entity and handle that sequence as a single query keyword. Next, XKQSS splits the tagged query into a set of segments, as done in StruX. Based on this, an algorithm that uses the segments and our index as input is proposed to compute a list of semantically relevant element/segment pairs, which we called predicates. This algorithm is derived by a set of propose guidelines. Another algorithm that exploits our index (and not DTD) to entity nodes that represent real-world entities in the XML document in question is also proposed. The lists of predicates and entity nodes computed are combined to generate a set of structured queries. To find which query best interprets user search

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

intention, we also proposed a ranking function that scores each individual query. Finally, user or a system can select the top 1 structure query and submit it to XML database system for retrieval. Our experimental result shows that XKQSS can compete with some state-of-the-art systems in terms of determining user search intention.

The contributions of our work are:

- We propose and implement two algorithms that exploit our indexes to compute predicates and entity nodes.
- As a query keyword can refer to many sort things an XML data, two guidelines were proposed to resolve these ambiguities.
- Developed a ranking function that is used to rank individual generated structured queries.
- Conduct an experiment to compare the effectiveness of the proposed system with a state-of-the-art system.

The rest of the paper is organized as follows: Section 2 describes related works, while Section 3 describes the XML data model and some notations used in the paper. Section 4 describes our propose keyword structuring approach, while Section 5 describes the index for effective query structuring. Experiment is discussed in Section 6, and Section 7 contains the conclusion.

## 2. Related works

Several systems have been developed to improve the retrieval effectiveness of keyword queries. One category of such systems is the query structuring systems, which, given a keyword query, automatically generates a set structured queries. For example, Li *et al.* (2009) developed a system called *XBridge*, which improved the effectiveness of keyword search. Given a keyword query in which each keyword in the query is expressed as *label:term* pair and the XML schema, *XBridge* first computes the contexts (i.e. predicates) of the set of labels and types of return nodes from the query. The contexts and returns nodes are then used to generate a set of effective structured queries, which are evaluated using a XML search engine. As *XBridge* requires each keyword in the user query to be labelled with a tag name, this means user must have knowledge of the schema of the underlying XML document. This is not possible for most users.

Similar to XBridge, given a keyword query in which a user has provided structural clues to indicate the type of XML element he/she is interested in and a thesaurus of XML tags, Petkova *et al.* (2009) developed a method for transforming a user keyword query into a set of structured queries. Their method first splits the keyword query into content and structural terms based on whether a query keyword appears in the thesaurus or not. As a structured query is made up of different parts, the authors introduced a notation called "target" that designate a part of a structured query. A structural term is an unbounded target and content term was used for bound target. For example, consider the query "book xml retrieval" where "book" is in the tag thesaurus and the remaining terms are not. Thus, "book" is a structural term, while "xml" and "retrieval" are content terms. The query is then transformed into a set of targets: *//book[~'xml']* and *//book[~'retrieval']* and scored based on the probability that "xml" and "retrieval" occurs in the text of an element of type "book". Operators are then used to combine two targets into a single structured query and their probabilities are multiplied together. For

example, the above two targets are merged into *//book[~'xml retrieval']*. Similar to XBridge, users have to provide structural hint in the input keyword query. Also, the XML schema is required in advance to create the thesaurus of XML tags.

However, the system proposed by Li *et al.* (2010) does not dictate the way a user would pose his/her queries. They proposed a framework called XML Information Object Finder (XIOF) that generates structured queries from free formatted keyword query by analyzing the given keyword query and schema of the XML data sources. A user does not need to know the XML structure.

In a recent study, Hummel *et al.* (2011) developed a system which also does not dictate the way a user poses his/her queries called StruX. Given a keyword query, StruX generates a sequence of segments based on the keywords in the query, where a segment consists of one or more query keywords. Thereafter, pairs of relevant segments are combined to form what is called "segment combinations". Each segment combination is labelled with elements from the XML data forming a set of predicates. Appropriate entities are then computed from the data based on the heuristic by Liu *et al.* (2007) and used to label the element–segment combinations. This step generates a set of structured queries and one of them is selected for execution. The splitting of query keywords into segments in StruX does not consider the fact that a group of consecutive sequence of keywords represents a single named entity and, therefore, need to be treated as such. Splitting that group of keywords could distort the semantic of the query, thus generating many unnecessary segments. Also, combination of any two pair of segments to form predicates can lead to the formation of predicates that violate the structure of the XML data. Their ranking functions assign score to individual structure query based on statistic of the keyword in the nodes that form the query only.

The second category of systems that improve the effectiveness keyword search is called lowest common ancestor (LCA)-based systems (Guo *et al.*, 2003, Xu *et al.*, 2005, Li *et al.*, 2007, Sun *et al.*, 2007). These systems compute the LCA or their variants of XML elements that contain the input keywords. A LCA is a lowest node in the XML tree that is the common ancestor of the nodes containing these keywords. The systems then identify the sub-trees rooted at the LCAs as the answer. Specifically, LCA-based systems make restrictions on the choice of the root node. This leads to answers that are either irrelevant to user search intention, or answers that may not be meaningful or informative enough.

## 3. Preliminary

### *3.1 XML data model and notations*

We represent XML data as a rooted label and ordered tree. Every internal node in the tree has a name and each leaf node has a data value. In this paper, we do not consider XML elements' attributes as part of our data. However, we use the term attribute to describe those nodes with data value as their only child node. Also, each node is given a Dewy label as its unique ID.

*3.1.1 Definition 1.* A node is a sequence of element names that appear in the path from the root to a node. A node type is the tag name of the node. For example, "SigmodRecord/issues/issue/articles/article/authors/author" is a node and it type is "author".

*3.1.2 Definition 2.* A deweyId of a node is a unique sequence of digits separated by (.) that is given to each node to uniquely identify it. For example, in Figure 1, the string 0.0.2.1 represents the deweyId of node "SigmodRecord/issues/issue/volume". The last

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

digit of deweyId indicates that the node is the second child of node 0.0.2, i.e. node "SigmodRecord/issues/issue".

*3.1.3 Definition 3.* A predicate in this paper is a node-value pair. A node represents a node in the XML data and value is sequence of one or more keywords. For example, */authors/author* = "Victor John" and */article/title* = "xml retrieval" are examples of predicates, where */authors/author* and */article/title* are called *predicate nodes* and "Victor John" and "xml retrieval" are called *predicate values*. We say, the node */authors/author* contains or labels "Victor John". A *simple predicate* is one with just a node and a value, while *complex predicate* is a combination of simple predicates connected with an "or" or "and" operator.

### *3.2 Exploiting semantic from XML document and user query using NER*

This section describes how NER can be used to identify the semantic relationship between elements in Sigmod XML data (www.cs.washington.edu/research/xmldatasets) and the user queries. It describes how NER is used to determine the proportion of named entities in the data value of a leaf node and a query. If the proportion of a named entity in a node matches that of the query, we say the node/element and query are semantically related.

XML document nodes contain rich semantics which can be conveniently inferred from their text or numerical expression. Given a piece of text, a NER (Finkel *et al.*, 2005) is used to detect a word or sequence of words as a person, an organization or a location, and so on. Using NER, we can exploit the content of all leaf nodes to determine proportion of named entities the node usually contains. A named entity is a word or sequence of words that is used to refer to something of interest in a particular application, e.g."something of interest" can be a person's name, an organization's name, a location's name, and so on (Goh *et al.*, 2013). Consider Table I obtained by applying NER on the leaf nodes content of Sigmod XML document; the rows show the leaf nodes in the document and the columns show the three different categories of named entities considered in this paper. For example, the *author* node has above 92 per cent of its text content to be composed of *person* named entities, while *title* node has above 70 per cent other named entity keywords. Note that the *other* named entity column refers to other named entities that are neither *person* nor *organisation*. For example, a keyword or a sequence of keywords could describe other entity like a *location* or a *date* or a *quantity*. These named entities fall under the *other* named entity category.

The table indicates that *author* node is used to label text, which mostly contains person or organization named entities and we say that the node is semantically related to a keyword or a sequence of keywords that describes a person/organization. Consider, for example, the segment "John Victor". Because each word in the segment is a person

**Table I.**
Leaf nodes and proportion of their named entity categories

| Node | Person (%) | Organization (%) | Other (%) |
|---|---|---|---|
| Title | 2 | 6 | 70 |
| Author | 92 | 8 | |
| Number | 0 | 0 | |
| Volume | 0 | 0 | |

named entity, we say the segment is semantically related to "author" node. While the segment "xml retrieval" is more related to "title" node than the "author" node.

Similar to a document, a user keyword query can also be analysed using NER system. Consider for example, the query Q = *client server Access Team Microsoft*. Using NER, we found the sequence "Access Team Microsoft" to be an organization named entity. Therefore, based on Table I, the sequence is semantically related to the *author* node. While the remaining sequence *client server* is categorized as belonging to *other* named entity category and so semantically related to *title* node. Based on these, we can infer from Q that user is searching for *articles* whose *title* contains "client server" written by "Access Team Microsoft".

*3.2.1 Definition 4. (Confidence value)* The confidence value of a node w.r.t, a named entity category is a numeric value that measures the degree to which a node would contain a query keyword. This also means the degree to which a node can be used to label the segment. It is the total number of a certain named entity that appears in a text value of a node over the total number of all named entities in the text of the node in question.

Table I shows the confidence values of some leaf nodes of the XML document in Figure 1 with respect to three categories of named entities: *person*, *organisation* and *other*. For example, given a segment whose keywords are *person* named entities, the Table I shows that we are 92 per cent confident that the segment is related to *author* node.

## 4. Query structuring – XKQSS

This section presents our proposed method for generating structured queries from keyword query. The method is implemented as a four-step process. First, we compute the predicates, .i.e. search condition (see Section 4.2); second, we compute the return nodes by computing the real-world entities in the XML document (see. Section 4.3). Next, we have query formulation, where the predicates and return nodes are combined, a collection of ranked structured queries are generated and the highest ranked structured query is selected (Section 4.4) by the system. The following section gives an overview of the propose method and then detail each of its steps.

```
Algorithm 1:
Input: document collection, keyword query qry
Output: top 1 structured query
1.    segments = genSegments(qry);
2.    predicates = generatePredicates (segments, index);
3.    entities = generateReturnNode (predicates, index);
4.    strucQueries = formQuery (entities, predicates);
5.    query = strucQueries[0];
```

### *4.1 Overview*

Algorithm 1 describes the general steps for proposed technique. We describe each step using the example shown in Figure 2. Given a keyword-based query as input, our system first pre-processes the query using NER and extracted consecutive sequences of query keywords that represent a person or an organization named entity. This consecutive sequence of keywords is treated as a single keyword. The

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

client server Access Team Microsoft

**(a)**

client/O server/O Access/ORGANIZATION Team/ORGANIZATION
Microsoft/ORGANIZATION

**(b)**

client
client server
client server Access_Team_Microsoft
server
server Access_Team_Microsoft
Access_Team_Microsoft

**(c)**

**Figure 2.**
A query and its segments

tagged query is then split into segments as done in StruX. For example, Figure(2a) shows an example keyword query, and Figure (2b) shows the same query with each query keyword annotated with its named entity category by the NER system. The consecutive sequence of tagged keywords "Access Team Microsoft" shown in Figure (2b) is then identified as an organization and so treated as a single keyword during segments formation. Figure (2c) shows segments generated by our system. Note that the underline sequence of keywords is treated as a single keyword and no segment shall contain more than one named entity category. Only six segments are obtained from a query of five keywords as against 15 segments obtained by StruX. We also define a segment as a sequence of one or more keywords in the order in which the keywords appears in the query.

Secondly, for each node in the underlying XML database, the system computes relevant segments and labels each relevant segment with the node. Also, for each related node and segment, a score would be computed which indicates the degree of relevance between the segment and the node. As a node can have more than one relevant segment, the most relevant segment would have to be selected for that node based on our proposed guidelines. This step is discussed in detail in the following section. Its purpose is to compute predicates. Thirdly, entities are computed using our proposed algorithm. Finally, given the list of predicates and that of the entities obtained from the above two steps, these lists are combined meaningfully to form a set of ranked structured queries and a rank list of structured queries is returned to the user. The user or system then selects the top 1 structure query. The selected query is then submitted to an XML database for execution.

### *4.2 Computing the predicates*

This section presents how to generate predicates, which are used to restrict the type of query result to be returned by an information retrieval system.

Algorithm 2 computes all the predicates given a list of segments and inverted index. It first retrieves all nodes from our index Line 1 by calling *getAllNodeType*() function. Next, for each node, the algorithm finds (in Lines 2-10) relevant segments for that node out of the given segments and labels each relevant segment with the node. Also for each node and a relevant segment, a score is computed in Line 7 using equation (1) adopted from Bao *et al.* (2010). The score indicates the degree of relevance between the segment and the node.

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

```
Algorithm 2: generatePredicates
Input: list of segments, i.e. segmentlist
Output: List of predicates, i.e. predList
 1. allnodesType = getAllNodeTYPE ();
 2. for each NodeType n in allnodesType do
 3.    segments = null; // list of relevant segments for node n
 4.    for each segment s in segmentsList do
 5.       s = label(n, s); // applying Guideline 1 & 2
 6.       if (s != null)
 7.          score = log (1 + Σ_{k∈s} f_n, k)
 8.          if (score > 0) then segments.add(s, score)
 9.    segment = selectBestSegment (segments)
10.    if   (segments ! = null) then
11.             predList.add(n, segment);
12. predList = genbestPredivates (predList);
13. return predList;
```

$$Score(n, s) = \log_e\left(1 + \sum_{k \in s} f_{n,k}\right) \quad (1)$$

where $n$ is a node, $s$ is a segment, $k$ is a keyword from s, and $f_{n,k}$ is the number of nodes of type $n$ containing the keyword $k$.

Although equation (1) provides a way to compute the confidence that a node can be used to label a segment, it alone is not adequate to infer the likelihood of an individual node to be used to label a given segment, as demonstrated by the following example.

*4.2.1 Example 4.1.* Consider a query "17 2 Tandem Performance Research Group" issued on Sigmod data in Figure 1, and most likely it intends to search for an *issue* whose volume "*17*" and number "2" and one of the authors is "Tandem Performance Research Group". Using equation (1) alone would return the following four predicates together with their scores:

(1) */issue/volume* = "17" *0.6931471805599453*.

(2) */issue/number* = "2" *2.995732273553991*.

(3) */article/title* = "Performance Research Group" *4.709530201312334*.

(4) */authors/author* = "Tandem Performance Research Group" *1.791759469228055*.

The third predicate got the highest score. This is because there are more */article/title* nodes that contain the segment keywords than the */authors/author* node. But the user search intention does not include *title*. Consequently the third predicate is not required even though it has the highest. Actually, the keywords in the */article/title* node, i.e. "Performance Research Group", semantically refers to an organization that wrote an article and not an article's title. Therefore, it is safe to assume that the user is referring to those keywords in the "/authors/author" node and delete those in the "/article/title' node. This action left the *title* node with an empty segment so that predicate is ignored. The limitation of equation (1) is also observed by XReal and proposed two formulae to compliment it.

Motivated from the above example, we pre-process the segments using two guidelines proposed before applying equation (1). The first guideline removes a

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

segment's keyword that matches the tag name of a node from the set of segment's keywords, while the second guideline uses the NER system to categorize segment's keywords and removes those keywords that are not semantically related with the node in question. These guidelines would be explained in some detail in the following sections.

4.2.1.1 Guideline 1. Given a node $v$ and a segment $s$. We label the keywords in $s$ with $v$. If a keyword $k \in s$ matches an element name that appears along path from $v$ to the root node, it means user is referring to a node in the document, not a node value. Such a keyword would be removed from the segment. This pre-processing guideline helps to resolve the popular keyword query Ambiguity 1. The following example demonstrates how to apply Guideline 1 works in Algorithm 2.

*4.2.2 Example 4.2.* Supposing that on Line 2 of Algorithm 2, we have node $n$ = "SigmodRecord/issues/issue/articles/article/title" and on Line 4 segment $s$ = "title xml retrieval", and as the keyword "title" in $s$ matches the "title" tag found along the path from "title" node to root, the algorithm removed the "title" keyword from segment $s$. This generates a new segment, $s$ = "xml retrieval", on Line 5. A score for the node and new segment is then computed on Line 7. If it is not zero, segment $s$ would be added to the list of relevant segments for the "title" node.

4.2.2.1 Guideline 2. As XML tags are used to label data in the XML documents while labelling a segment with a node, we try to find out if all the keywords in the segment match the node semantically. If a keyword is not semantically related to the node in question, it is removed from the segment. This guideline addresses another popular keyword query Ambiguity 2. The following example demonstrates how to apply the guideline.

*4.2.3 Example 4.3.* It is supposed that there is a list of three segments: *s1, s2 and s3*, supplied as input to Algorithm 2, and on Line 2 of the algorithm, we have node $n$ = "SigmodRecord/issues/issue/articles/article/authors/author", as shown in Figure 3. We want to find which of these segments are relevant to the given node $n$. Notice that $n$ is a node of type *author* and based on Table I, *author* node is used to label text composed of either a person or an organization entity.

Next, given $n$ and the three segments, Line 4 first considers segment *s1*, Line 5 of the algorithm applies Guideline 2 to find if $n$ and *s1* are semantically related. Because none of the keywords in segment *s1* represents a person or organization entity, all keywords in *s1* are removed and Line 5 returns $s = null$. This means $n$ and *s1* are not semantically related. Now because $s1 = null$, the algorithm skips that segment and iterates back to Line 4 and takes the next segment *s2*. As all the keywords in *s2* represent a person entity, this implies that $n$ and *s2* are semantically related. Consequently, a score for $n$ and *s2* is computed on Line 7 using the equation (1) and if the score is not zero, *s2* is added to the list of relevant segments for the *author* node. The algorithm iterates back again and considers *s3*. The first keyword in *s3* is not a person entity, so it is removed from the segment on Line 5, given *s3* = "John Michael". Next is to compute a score for $n$ and the new segment. However, because such a predicate was considered in the second iteration,

SigmodRecord/issues/issue/articles/article/authors/author

s1: xml retrieval
s2: John Michael
s3: retrieval John Michael

**Figure 3.**
A node and three segments

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

no score would be computed and the algorithm iterates back again to Line 4. This time, no segment was found. The *author* node (and its relevant segments) is then added to the overall list of nodes in Line 11. If a node has more than one relevant segment, Line 9 of Algorithm 2 selects the most relevant segment (see Section 4.2.1). After Line 11, Algorithm 2 iterates back to Line 2 and considers the next node. Here also, for this node the algorithm tries to find relevant set of segments from the input list of segments and label the segments with the node as it does in the first iteration. This is done for all the nodes in the database. Lines 2-8 of the algorithm return a list of node and with each node having one or more segments.

For example, for the query $Q$ = "xml retrieval model Victor", Lines 2-8 of the algorithm return list of nodes and their respective segments, as shown in Figures (4a) and (4b).

From Figures (4a) and (4b), we can see that there are two nodes, *author* and *title*. The *author* has just one segment [Figure (4a)], while the *title* node has several segments [Figure (4b)]. As the *title* node has more than one segment, the most relevant segment must be selected (Line 9). The selection process is based on a guideline discussed in the section that follows.

4.2.3.1 Selecting best relevant segment. As a node can have more than one relevant segment with each segment having a score, it is necessary, in such a case, to determine which segment is more relevant to represent the original user's query intention. In view of this, we propose a guideline for selecting the most relevant segment as follows.

4.2.3.2 Guideline 3. Given a node and a list of its relevant segments, the most relevant segment for that node is the segment with highest score among all other segments. If the node has more than one segment having same highest score, the segment having the smallest length is the most relevant and is selected.

*4.2.4 Example 4.4.* The article"/article/title" node in Figure (4b) has several segments. One of them must be selected. For that node, segments "xml retrieval model" and "xml retrieval model Victor" *have* the same score, which is also the highest score. As the later segment has the least number of keywords, it is selected on Line 11 as the most relevant segment with w.r.t. the *article* node.

The list of predicates for query Q is shown in Figure (4c). The following describes how to select the best predicates among the list of predicates.

SigmodRecord/issues/issue/articles/article/authors/author

| Victor | 1.6094379124341003 |
|---|---|

**(a)**

SigmodRecord/issues/issue/articles/article/title

| xml | 3.8066624897703196 |
|---|---|
| xml retrieval | 3.8066624897703196 |
| xml retrieval model | 4.736198448394496 |
| xml retrieval model victor | 4.736198448394496 |
| ... etc | .... etc |

**(b)**

SigmodRecord/issues/issue/articles/article/title = xml retrieval model
SigmodRecord/issues/issue/articles/article/authors/author = Victor

**(c)**

**Figure 4.**
Two nodes and their segments with two predicates

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

4.2.4.1 Selecting most relevant predicates. Algorithm 2 returns list of predicates. However, it is possible for two or more distinct predicates to have different nodes that label same data value. In such a case, we say the nodes label same type of named entity category. One of these predicates must also be selected. Below is a guideline which helps to determine the best predicates.

4.2.4.2 Guideline 4. When two or more distinct predicates contain nodes that label same type of named-entity category as their text value, a predicate whose segment consists of more numbers of named-entity keywords of the same type would be selected or if the number of named-entities keywords is the same, we select the predicate whose node has higher confidence value to label such named-entity. Confidence value can be obtained from our index. To understand this guideline, let us consider the following examples.

*4.2.5 Example 4.5.* Supposing that after Line 10 of Algorithm 2, we have a list of two predicates, as shown in Figure (3c) and brought here for clarity:

SigmodRecord/issues/issue/articles/article/title = xml retrieval model.
SigmodRecord/issues/issue/articles/article/authors/author = Victor.

Because the two predicates contain nodes that label different types of name-entities, we do not apply Guideline 4 on these predicates. Meaning that, no predicate would be removed and so we have a list of two predicates.

However, supposing, for example, that, instead of what we have in Figure (4c), we have:

SigmodRecord/issues/issue/articles/article/title = Victor.
SigmodRecord/issues/issue/articles/article/authors/author = Victor.

Both nodes, in this example, contained/labelled the same keyword "Victor", which is categorized as a *person* entity. Because both nodes labelled the same *Person* entity and each contain same number of keywords, we check our index and retrieve the confidence value of the two nodes (i.e. *title* and *author* nodes) with respect to the *Person* entity and remove the predicate with the smallest confidence value. In the above case, we remove the predicate having the *title* node because the *autho*r node has high confidence value to label/tag *Person* name-entity. This gives us only one predicate:

SigmodRecord/issues/issue/articles/article/authors/author = Victor.

Also, in supposition as another example, instead of what we have in Figure (4c), we have:

SigmodRecord/issues/issue/articles/article/title = Victor.
SigmodRecord/issues/issue/articles/article/authors/author = Victor Samuel.

Here, both nodes labelled same type of named-entity, i.e. *person* entity. We select the *author* node since it has two *person* entity keywords as against one keyword in the *title* node:

SigmodRecord/issues/issue/articles/article/authors/author = Victor Samuel.

This guideline helps to reduce the number of predicates and it also helps to resolve Ambiguity 2 by selecting predicates whose nodes are not only statistically relevant but also semantically related with the query.

*4.2.6 Merging*. The next step called the merge operation used the generated predicates from above step and generates more complex predicates in addition to the simple one. The merge operation combines two or more predicates to form a complex predicate using the "and" operator if their lowest common prefix (LCP) is not null. This "not null" condition would prevent us from violating the structure of the XML documents. For example, consider the two predicates in Figure (4c). Using the two predicate nodes, merging operation would first compute their LCP. Their LCP is *SigmodRecord /issues /issue /articles /article*. As LCP is not null, merge combines the two predicates using the "and" operator and updates the list of predicates to a list of three predicates, as shown in Figure 5.

Figure 5 shows the three predicates generated by the merging operation. We then find appropriate set entity nodes that would be used to label the predicates. The following section describes our proposed algorithm that computes entity nodes from our indexes. Entity nodes are the result nodes while the predicates obtained in the above step restrict the type of entity that would be returned as answer to a query.

### *4.3 Entity generation*

When a user issued a query, he/she is looking for an entity or entities in real-world along with their relationships in the document (Termehchy and Winslett, 2011; Bao *et al.*, 2010; Liu *et al.* 2007). In relational database, an entity denotes a "thing" or an "object" in the real world. While in the concept of XML databases, as these databases have tree structures, there are many sub-trees in the databases that denote "things" or "objects" (Lou *et al.*, 2013). The root nodes of these sub-trees are called entity nodes. Also called return nodes because they are the target document elements the user is interested to return. This section describes an algorithm (Algorithm 3) that is used to generate entity nodes in an XML document using our index described in Section 5.

To generate entity nodes, similar to StruX, we use the following inference in the study by Liu *et al.* (2007) for entity node identification:

- If a node has siblings of the same name, then this indicates a many-to-one relationship with its parent node, and is considered to represent an entity.
- If a node does not have siblings of the same name, and it has one child, which is a value, then it is considered to represent an attribute.
- A node is a connection node if it represents neither an entity nor an attribute.

```
Algorithm 3: generateEntity
Input: inverted index
output: List of entity nodes
 1. enodes = null // list of entity nodes
 2. Lnodes = getNodes(index.FrequncyTb);
 3. for (int i = 0; i &lt;Lnodes.size(); i++)
 4.     if ( Lnodes (i) and Lnodes(i+1) are from same document) and
                         (Lnodes(i) and Lnodes(i+1) are of the same type) and
                                (Lnodes(i) and Lnodes(i+1) have the same parents))
```

/title = 'semantic database'
/authors/author = victor
/title = 'semantic database' **and** /authors/author = 'Victor'

**Figure 5.** Three generated predicates

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

```
 5. diff = Lnodes(i+1). getDewSuffix() - Lnodes(i).getDewSuffix()
 6. if (diff == 1) // we got an entity node
 7.       entNode = nodes(i).path();
 8.       if [! enodes.contains (entNode)]
 9.          enodes.insert (entNode)
10. return enodes
```

However, in StruX, entities are computed from DTD based on the above inference. The problem with using DTD is that query processing relies on the presence of XML document DTD. However, DTD is not always available. Also, the system can infer multiple value attributes as entities if DTD is used to compute entities.

To avoid these challenges, we inferred our entities based on the above inference but using our index, not DTD. Thus, the process of generating entities is guided entirely by the underlying XML database. Algorithm 3 is a flowchart describing how to model the above inference using our index to compute the entities. The algorithm works as follows:

On Line 2, Algorithm 3 first retrieves all the rows in the *frequencytb* table ordered by document order. As described in Section 5, each row in the table represents the details of a node with respect to a query keyword. For ease of description, we call each row a *node*. The function *getNodes()* returns a list of nodes, *Lnodes(i)* denotes node i in the list of nodes. Each node support the *getDewSuffix()* method. This method returns the last digit of the dewey Id of a node. Therefore for a node with dewey Id 0.0.2.3, *getDewSuffix()* returns 3.

Line 3 scans the list of nodes. For each iteration, the algorithm takes a pair of nodes *Lnodes(i)* and *Lnodes(i+1)* from the nodes list, and then for each pair, Lines 4-9 find the entity nodes if all the following conditions are true:

Determines if the two nodes:

(1) have same parent;

(2) are of same node type; and

(3) have $Lnode(i+1).getDewSuffix() - Lnode(i).getDewSuffix() = 1$.

On Line 10, the algorithm returns a list of nodes (*enodes*) that satisfy the above three conditions. These nodes represent the real-word entities in XML document which users might be interested in. However, only one of them is valid for a user query (Lou *et al.*, 2012). Therefore, it is necessary to find a valid entity node from the returned list of entity nodes (*enodes*). In this paper, the selection is done during query formulation discussed in the following section where a proposed ranking formula is used to fish out the valid entity node.

*4.4 Query formulation*

This section describes how to generate a set of structured queries given a list of predicates and that of entity nodes, as shown in Figure 6. The Figure shows a list of three predicates and a list of four candidate entity nodes computed by our system. We label each predicate with each entity node from the entity nodes list. We call each pair entity – predicate. Each entity – predicate pair is a candidate structured query. Equation (2) is proposed as a ranking formula to compute score for each entity – predicate pair. The scores are used to rank the queries. Our ranking formula is better than the existing systems ranking formulae because existing systems did not take into account the semantic relationship between an entity node and that of predicate node(s):

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

$$Score(q) = \left(\alpha\frac{1}{distance} + \beta \sum_{i=1}^{k} score(n_i, s)\right) \quad (2)$$

Where $q$ is a structure query, $k$ is the number of nodes in a predicate, $score(n_i, s)$ is as explained in equation (1), *distance* is the difference between a predicate's node path length and an entity node path length. If the predicate contains more than one node we take the minimum *distance*.

The factor is 1/*distance* in the equation to measure the semantic relationship between the entity node and the predicate node. The closer they are, the more semantically related they appear. The second factor $\Sigma score(n_i, s)$ is based on intuition 1 of XReal (Bao *et al.*, 2010). The constants $\alpha$ and $\beta$ indicate the importance of each factor. We use $\alpha = \beta = 0.5$ to indicate that both semantic relationship and statistic of keyword have same importance.

Algorithm 4 represents a flowchart for our query formulation. It first takes a predicate $p$ from the list of predicate $Pr$. On Line 2, it takes an entity node $e$ from the list entity nodes $E$. As a predicate $p$ can have more than one predicate nodes, it then call its getAllPredNodes()method to return all its predicates nodes on Line 3. This means on this line, we have an entity node $e$ and set of predicate nodes. The algorithm then compute the most related predicates node w.r.t node $e$ as follows (see Line 4-11): Each node supports *getPathLength()* method that compute its path length. So path lengths for pn and e are computed and the absolute difference between their path lengths on Lines 7-11 if $e$ is an ancestor of $pn$ is also computed. We called this "distance" which is used in equation (2). Lines 4-12 generates a list of nodes with their corresponding "distance" w.r.t an entity node $e$. After this, function *getNodeWithSmallDistance()* returns the node with smallest distance on Line 13. This predicate node with smallest distance is then store in list of predicate nodes. The algorithm iterates back to Line 2 and consider another entity node, also Lines 4-12 compute another predicate w.r.t the previous entity node. The process is repeated until all entity nodes are considers. The entity and set of predicate nodes are used to form a structured query using xquery syntax. After this, the algorithm iterates back to Line 1 and considers another predicate. Same process as describe before is repeated until no more predicate. Equation (2) is used to compute the score for each generate structured query. This score is used to find the query that best describe the user query intention. Finally a list of structured queries is returned.

```
Algorithm 4: formQuery
Input: list of predicates Pr, List of entity nodes E
Output: List of structured queries
  1. for each predicate p in Pr
```

```
predicates    /title = 'semantic database'
              /authors/author = 'Victor'
              /title = 'semantic database' and  /authors/author = 'Victor'

Entities      SigmodRecord/issues/issue
              SigmodRecord/issues/issue/articles/article
              SigmodRecord/issues/issue/articles/article/authors
              SigmodRecord/issues/issue/articles/article/authors/author
```

**Figure 6.**
Predicates and entity nodes example

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

```
2.  for each entity e in E
3.     Let nodes = getAllPredNodes(p)
4.     for each node pn in nodes
5.        if (is_ancestor(e, pn))
7.           plength1 = pn.getPathLength()
8.           plength2 = e.getPathLength()
9.           distance = plength1 - plength2
10.       else
11.          distance = 0
12.       nodeList.add(pn, distance)
13.    smallNode = getNodeWithSmallDistance(nodeList)
14.    distance = smallNode.getDistance();
15.    if (distance > 0)
16.       query = XqueryBuilder()
17.       pscore = 0
18.       for each node n in nodes
19.          pscore = pscore + n.getSegScore();
20.       score = 0.5(1/distance + pscore)
21.       QueryList.add(query, score)
22. Return QueryList
```

Figure 7 shows the overall query structuring steps by XKQSS. It first shows the keyword query, next the query tagged with their respective named entities, the next line contains the best generated XQuery query, fourth line contains the score of the newly generated and finally the result obtained when the generated query is executed against Sigmod XML database.

## 5. Index construction

This section describes our index structure used for effective keyword query conversion. To construct our index, we created a four database tables to store the indexes using MySQL database, as shown in Figure 8.

```
Keyword query: semantic  Victor Viannu
Tagged query: semantic/O   Victor/Person  Viannu/Person
XQuery query:  for $entity in collection('SigmodData.dbxml')//article
               where contains ($entity/title, "semantic") and
               contains ($entity/authors/author, "Victor Viannu")  return $entity
Query Score: 7.2930176
Result:  <article>
           <title articleCode="162050">Mapping a semantic database model to the relational
model</title>
           <authors>
             <author AuthorPosition="01">Peter Lyngbaek</author>
             <author AuthorPosition="02">Victor Viannu</author>
           </authors>
         </article>
```

**Figure 7.**
Overall query structuring steps

docTb{ *docId*, dname}

elmTypeTb {*NId*, epath, noOfchilds}

frequencyTb {*NId, dewed, docId, trm,* frqc}

semanticTb{*NId*, namEntity, confValue}

**Figure 8.**
Structure of the index

We traverse the XML document tree in pre-order and collect the following information for each document D and for each node *n* in D visited:

- Store the document ID and name in docTb in Figure 8.
- Assign a Dewey label dewId to *n*.
- Store the prefix path *prefixPath* of *n* as its node type in a database table in *elmTypeTb* in Figure 8. Each row in the table consists of the a node's name (*epath*), nodes' number of children (*noOfchilds*) and node Id (*NId*). *NId* is used to identify the node.
- In case *n* is a leaf node, we compute the number of occurrences of each term *x* in *n* and store a record for each term in *frequencyTb* in Figure 8. The table is similar to posting list. Each row in the table stores the detail of a node w.r.t. a particular keyword in the node. It consists of node Id (*NId*), the keyword *trm* and the number of times *trm* appears in the node (*frqc*). Also, in the row, we have the document Id (docId) and the nodes' dewey Id (dewId). This table is used to compute both the predicates and as well as the return nodes.
- Also for each leaf node (*NId*) and for each of the three named entity categories *namedEntity* considered, we compute the proportion of each named entity category in the text of NId (*confValue*).

## 6. Experiment

This section discusses the experimental setup to evaluate our proposed system. The effectiveness of the system was evaluated by measuring search quality using precision and recall evaluation metrics.

### *6.1 Experimental setup*

We used Intel (R) Core (TM) 3.20-GHZ with 8 GB memory running Windows 7 Professional. The propose system was implemented in Java and we created our index for effective query structuring and stored it in a MySQL database. Berkeley DB for XML was used to store and query the XML documents.

*6.1.1 Dataset.* This study would be conducted using two datasets. The first dataset, called "Sigmod" is obtained from Washington XML data repository (www.cs.washington.edu/research/xmldatasets). Sigmod XML dataset is a computer science bibliography dataset widely used for XML IR evaluation. It has a simple structure and the content information is riched (Guo *et al.*, 2003). It is a single file XML document with ten distinct elements, 4 of them were leaf elements. The second dataset called IMDB is obtained from INEX. The IMDB dataset contains information about various entities like movies, actors, directors and soon. The dataset consists of over 800 XML files and only 215 files were considered for the experiment. It consists of 50 distinct elements, of which 30 are leaf elements. For both datasets, stopwords were removed and no stemming.

Recall that our system returns an entity node from data being searched. In the IMDB datasets, every document root, such as *<movie>* is intuitively an entity node. Similar to StruX, our system considers a root element as not suitable to be an entity node. Consequently, to return a *<movie>* as an entity node, we automatically create a dummy root node *<movies>* to contain all the *<movie>* elements. Also, entity node consists of a set of related leaf nodes, called predicate nodes. With respect to IMDB dataset, ten predicate nodes were selected out of the 30 leaf elements from a survey involving 20 PhD

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

candidates. Each candidate was given a list of the 30 leaf elements and was asked to select the ones he often uses to describe a movie. The need to select the most frequent use leaf element out 30 is born out of the following observations:

- The smaller the number of predicates in an entity node the closer they are and the more meaningful (Cohen *et al.*, 2003).
- The basic requirement for an XML search system is to return an entity node that is not overwhelming to contain too much irrelevant information (Bao *et al.*, 2010, Keyaki *et al.*, 2011).

These leaf elements were used for predicate computation in Section 4.2.

*6.1.2 Query set.* To demonstrate the performance of our method, similar to XIOF and XReal, nine queries were randomly selected for each dataset. Queries *QS1-QS9* are evaluated on Sigmod data, while queries *MS1-MS2* for IMDB data. Table II shows the query evaluation result on Sigmod dataset, and Table III shows the query evaluation result on IMDB dataset. Notice that for IMDB the search intention is always a *movie* as it the only meaningful XML fragment because all other fragments are not informative enough. These queries contain explicit and implicit structural hints. As in Bao *et al.* (2010), Guo *et al.* (2003) Liu *et al.*, 2007, Xu *et al.*, 2005), this paper also assumes that a query keyword has at least one occurrence in the XML data being searched.

*6.1.3 Judgement.* Each keyword query is manually changed to its corresponding XQuery expression and the new query is used to retrieve relevant XML fragments. These relevant XML fragments are the accurate answers.

| Query Id | Query | Search intention | XKQSS | XReal |
|---|---|---|---|---|
| QS1 | Semantic database Victor Vianu | Article | Article | Article |
| QS2 | Georges Gardarin | Article | Authors | Articles |
| QS3 | Multimedia object manager | Article | Article | Title/articles |
| QS4 | Two client server | Issue | Issue | Article |
| QS5 | Fox Development Team Microsoft | Article | Authors | Article |
| QS6 | 17 2 Tandem Performance Group | Issue | Issue | Issue |
| QS7 | 24 2 Georges Gardarin | Issue | Issue | Issue |
| QS8 | Performance evaluation | Article | Article | Issue |
| QS9 | Server | Article | Article | Title |

**Table II.** Query evaluation result on Sigmod dataset

| Query Id | Query | Search intention | XKQSS | XReal |
|---|---|---|---|---|
| MS1 | Knots Reckell Peter | Movie | Movie | Movie |
| MS2 | Murder McGill Bruce | Movie | Movie | Movie |
| MS3 | Midnight Special 1973 | Movie | Movie | Title |
| MS4 | Countdown Lewinsky Monica | Movie | Movie | Movie |
| MS5 | Garafano Michelle director | Movie | Movie | Movie |
| MS6 | Organised crime Jo Hang | Movie | Movie | Movie |
| MS7 | Morris Haviland actor | Movie | Movie | Movie |
| MS8 | Zion Brother accident | Movie | Movie | Movie |
| MS9 | Proud family | Movie | Movie | Title |

**Table III.** Query evaluation result on IMDB dataset

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

*6.1.4 Search quality.* The output of our system is a ranked list of structured queries. To evaluate the search quality of our system, we compare the performance of the highest-scoring structured query with that of manually created XQuery queries and XReal. We evaluate effectiveness of the system by the metrics of precision and recall. Precision refers to the percentage of relevant results among all the returned results while recall refers to the percentage of returned relevant results among all the relevant results existing in an XML dataset. Formally, these two metric are defined as follows:

$$Precision = \frac{\|R_r \cap R_s\|}{R_r} \quad \text{and} \quad Recall = \frac{\|R_r \cap R_s\|}{R_s}$$

where $R_r$ is the result of our highest-scoring structured query or the result return by XReal and $R_s$ is the result of the manually created structured query (i.e. in XQuery format).

*6.1.5 Discussion.* All the queries in Table II were evaluated. The third column represents user search intention obtained by manually constructed queries; fourth column represents the nodes returned by the proposed technique, while the last column represents the set of nodes returned by XReal. Figure 9 illustrates the experimental results comparing the performance of XReal and XKQSS in terms of identifying the return node. We observe that our technique achieves better search performance than the method XReal. Figure (9a) shows that the XKQSS is able to infer about 70 per cent of the true return nodes while XReal only 40 per cent on Sigmod dataset, while 9b shows the XKQSS infers about 100 per cent of the return nodes and XReal about 80 per cent on IMDB dataset.

XKQSS perform better than XReal because with XReal, there are two factors to consider for a node to be a returned node. These factor are $1 + \Pi_{k \in q} f_k^T$ and $\gamma^{depth(T)}$ where $f_k^T$ is the number of T_type nodes containing k and $\gamma$ is the reduction factor in the interval (0, 1]. We use $\gamma = 0.8$. Using query MS9 i.e. " proud family ", let's analyse why XReal infer "title" node as the return node and not "movie". With respect to "movie" and "title" node, the second factor $0.8^{depth(movie)} = 0.8$ and $0.8^{depth(title)} = 0.64$ respectively. However, for the first factors; $(1 + f_{\{proud,\ family\}}^{title})$ is greater than $(1 + f_{\{proud,\ family\}}^{movie})$ so much that multiplying the first and second factors will not change the inequality. Therefore,

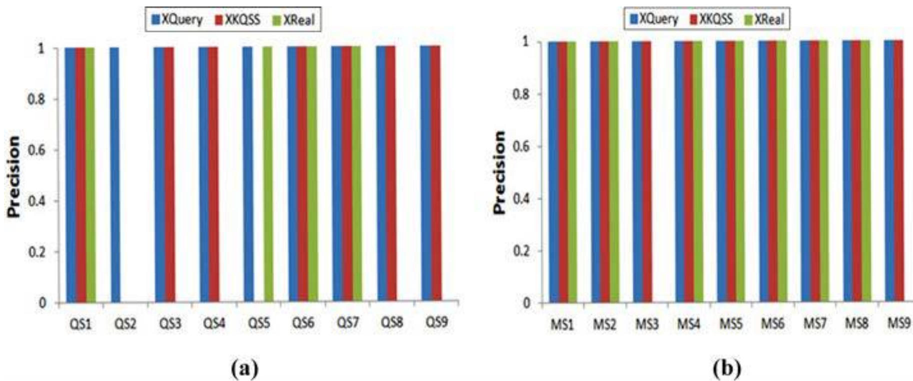


**Figure 9.** Precision comparison

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)

because ( $1 + f^{title}_{\{\text{proud family}\}}$ )* $0.8^{depth(title)}$ greater, XReal selects the "title" node instead of "movie" node.

## 7. Conclusion

In this paper, a query structuring technique called XKQSS that automatically converts a keyword query to a set of structured queries is presented. Given a keyword query as input, the technique used named entity tagger to categorize query keywords and exploits statistics derived from the document to infer search predicates. Further, the approach used a proposed algorithm to find entity nodes, which represent real-world entities in documents based on dataset instead of DTD. It considers these entity nodes as users search for nodes. Then, it combines the predicates and entity nodes according to the underlying structure of the XML document to construct candidate structured queries. Using a ranking scheme we proposed, our technique returns the structured queries in a ranked order. We compare the performance of the highest-scoring structured queries with XReal. The results demonstrate our technique that includes NER to resolve keyword queries ambiguities is feasible and that it is quite effective.

As future work, we plan to enhance query structuring technique. Specifically, the technique would be improved to handle very large datasets. The algorithm for computing entity nodes from XML data would also be enhanced by other heuristics for determining entities from XML dataset.

## References

Bao, Z., Lu, J., Ling, T.W. and Chen, B. (2010), "Towards an effective XML keyword search: knowledge and data engineering", *IEEE Transactions on*, Vol. 22 No. 8, pp. 1077-1092.

Cohen, S., Mamou, J., Kanza, Y. and Sagiv, Y. (2003), "XSEarch: a semantic search engine for XML", *Proceedings of the 29th International Conference on Very Large Data Bases,* Vol. 29, pp. 45-56, VLDB Endowment.

Finkel, J.R., Grenager, T. and Manning, C. (2005), "Incorporating non-local information into information extraction systems by Gibbs sampling", Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, pp. 363-370, Association for Computational Linguistics, available at: http://nlp.stanford.edu/~manning/papers/gibbscrf3.pdf

Goh, H.N., Soon, L.K. and Haw, S.C. (2013), "Automatic discovery of person-related named-entity in news articles based on verb analysis", *Multimedia Tools and Applications*, Vol. 74 No. 215, pp. 1-24.

Guo, J., Xu, G., Cheng, X. and Li, H. (2009), "Named entity recognition in query", *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, New York, NY, pp. 267-274.

Guo, L., Shao, F., Botev, C. and Shanmugasundaram, J. (2003), "XRANK: ranked keyword search over XML documents", *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY*, pp. 16-27.

Hummel, F.D.C., Da Silva, A.S., Moro, M.M. and Laender, A.H. (2011), "Automatically generating structured queries in XML keyword search", *Comparative Evaluation of Focused Retrieval*, Springer Berlin Heidelberg, Vol. 6932 No. INEX2010, pp. 194-205.

Keyaki, A., Hatano, K. and Miyzaki, J. (2011), "Result-reconstruction method to return useful XML elements", *International Journal of Web Information Systems*, Vol. 7 No. 4, pp. 360-380.

Li, G., Feng, J., Wang, J. and Zhou, L. (2007), “Effective keyword search for valuable lcas over xml documents”, *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, *ACM, New York, NY*, pp. 31-40.

Li, J., Liu, C., Zhou, R. and Ning, B. (2009), “Processing xml keyword search by constructing effective structured queries”, *Advances in Data and Web Management*, Springer Berlin Heidelberg, Vol. 5446 No. APWeb/WAIM 2009, pp. 88-99.

Li, X., Li, Z., Wang, P. and Chen, Q. (2010), “XIOF: finding XIO for effective keyword search in XML documents”, *2010 2nd International Workshop on Intelligent Systems and Applications (ISA*), IEEE, Wuhan, pp. 1-6.

Liu, Z. and Chen, Y. (2007), “Identifying meaningful return information for XML keyword search”, *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ACM, *New York, NY*, pp. 329-340.

Lou, Y., Li, Z. and Chen, Q. (2012), “Semantic relevance ranking for XML keyword search”, *Information Sciences*, Vol. 190, pp. 127-143.

Lou, Y., Wu, Q., Ji, B., Zheng, R., Zhang, M. and Wei, W. (2013), “Effective entity unit for XML keyword search”, *Journal of Computational Information Systems*, Vol. 9 No. 17, pp. 6811-6818.

Petkova, D., Croft, W.B. and Diao, Y. (2009), “Refining keyword queries for xml retrieval by combining content and structure”, *Advances in Information Retrieval*, Springer Berlin Heidelberg, Toulouse, pp. 662-669.

Sun, C., Chan, C.Y. and Goenka, A.K. (2007), “Multiway SLCA-based keyword search in xml data”, *Proceedings of the 16th international conference on World Wide Web*, ACM, *New York, NY*, pp. 1043-1052.

Termehchy, A. and Winslett, M. (2011), “Using structural information in XML keyword search effectively”, *ACM Transactions on Database Systems (TODS*), Vol. 36 No. 1, p. 4.

Xu, Y. and Papakonstantinou, Y. (2005), “Efficient keyword search for smallest LCAs in XML databases”, *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, ACM, *New York, NY*, pp. 527-538.

Zeng, Y., Bao, Z., Li, G. and Ling, T.W. (2013), “Removing the mismatch headache in XML keyword search”, *Proceeding of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, *New York, NY*, pp. 1109-1110.

**Corresponding author**
Abubakar Roko can be contacted at: abroko@yahoo.com

For instructions on how to order reprints of this article, please visit our website:
**www.emeraldgrouppublishing.com/licensing/reprints.htm**
Or contact us for further details: **permissions@emeraldinsight.com**

Downloaded by TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES At 02:04 09 November 2016 (PT)