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 13 (4): 643-651, 2014
ISSN 1812-5638 / DOI: 10.3923/itj.2014.643.651

Effective XML Keyword Search Using Dual Indexing Technique

S. Selvaganesan, Su-Cheng Haw and Lay-Ki Soon
Faculty of Computing and Informatics, Multimedia University, 63100, Cyberjaya, Malaysia

Abstract: Achieving the effectiveness in relation to the relevance of query result is the most crucial part of XML keyword search. Developing an XML Keyword search approach which addresses the user search intention, keyword ambiguity problems and query result grading problem is still challenging. In this study, a new keyword search approach, named XDMA (XML keyword search dual indexing and mutual summation algorithm), for XML databases based on dual indexing, is proposed to resolve these problems. This approach includes design of mutual score, entropy-based similarity measure so as to find the relevant results for a given XML keyword query and, grading score to grade the query results. The algorithm for the new approach is presented and finally, space and time complexity of the algorithm have been analyzed to show the effectiveness of the algorithm for XML keyword search. The proposed algorithm XDMA can retrieve the relevant results for XML keyword query more effectively.

Key words: XML keyword search, XML databases, indexing, mutual score, similarity score

INTRODUCTION

In XML keyword search, the accurate identification of user search intention and grading of the result in the presence of keyword ambiguities have been challenging problems. The keyword search approach proposed by Bao *et al.* (2009, 2010a) made use of the numerical facts or data of the XML database to address these problems. However, the approach suffers from the following limitations and problems. For the purpose of expediting the query processing, two indices are constructed. Of these indices, keyword inverted list is the index constructed at first. For each keyword inverted list, an index like B+tree is built on its top increasing the space requirement. Moreover, keyword inverted list does not provide information about leaf tag containing data values. Subsequently, the frequency table which is the other index, is constructed. The frequency stored in this table is the number of XML T-typed nodes having keyword in their sub trees. Also, the frequency table does not consider data nodes of XML databases. Hence the query processing has been made more complex. In both indices, frequency of text value contained in a leaf node is not considered. This approach does not identify the query keyword in the frequency table as tag name of nodes or text value of data node.

In order to address the problems mentioned earlier in this section, a novel dual indexing approach, XDMA, which builds dual indices, namely, tag_info table for XML structural nodes and data_info table for data nodes in XML databases, is proposed. Hence, the query processing will be made simplified. In this approach, a new keyword searching technique involving two-level matching between two indices has been considered. By utilizing the dependence of two indices and concept of mutual sum, the mutual score is defined to determine precise T-typed node. Also, a similarity function is proposed using which a formula for similarity score is obtained to determine the precise data.

In recent times, there have been much research activities in XML keyword search. Here, the related researches for XML keyword search conducted in the tree data model are concisely reviewed. In addition, the work done on the result grading is reviewed.

In the approach (Schmidt *et al.*, 2001) utilizing the XML tree structure, an operator is exclusively proposed to enable XML databases to be queried and moreover, determining the Lowest Common Ancestor (LCA) of nodes is the basis of this work. In their approach the result type of the query is dependent on the database instance and not specific. Li *et al.* (2004) developed a framework enabling users to query XML data with partial knowledge of the document schema and also devised a stack-based Meaningful LCA (MLCA) searching algorithm. Then, XKSearch System (Xu and Papakonstantinou, 2005) proposes the concept of smallest LCA (SLCA) and furthermore contributes three algorithms to determine SLCAs in an effective manner. Hristidis *et al.* (2006) introduce keyword proximity queries in their work and also provide algorithms to determine minimum connecting trees. Sun *et al.* (2007) investigate keyword query processing based on SLCA and subsequently

Corresponding Author: S. Selvaganesan, Faculty of Computing and Informatics, Multimedia University, 63100, Cyberjaya, Malaysia

propose an efficient multiway-SLCA approach. Also, this approach deals with keyword queries using both AND and OR semantics. All these approaches, based on LCA and variants of LCA, do not solve the search intention problems and the ranking problems.

Other works in the tree data model include XSeek, presented by Liu and Chen (2007) which creates return nodes by examining the match patterns of query keyword and the structure of XML data. Nonetheless, XSeek has not addressed both keyword ambiguity and ranking problems. MaxMatch, Liu and Chen (2008) enhances the appropriateness of matches and also result quality. On the other hand, Li *et al.* (2009a) proposed an adaptive XML keyword search approach, XBridge, to process keyword search by constructing a set of effective structured queries. Also, in their work, they have considered XML schema as tree patterns whereby the reference relationships of XML data have not been considered. XKMis (Li *et al.*, 2009b) decreases the false positives considerably and also generates query results that are notably more meaningful. Li and Wang (2009) presented an interactive XML keyword search system, XQSuggest, allowing for query suggestion and enabling users to express their queries more clearly. They proposed an algorithm for finding the results of the transformed query. Moreover, two optimization techniques were used in the algorithm to speed up query processing.

Bao *et al.* (2009) introduced an approach making use of the numerical facts of XML database to address the challenges, namely, identification of user search intention, result retrieval and relevance based result ranking as a single problem for XML keyword search. They presented an XML keyword search engine prototype called XReal which is implemented to achieve effective identification of user search intention and relevance based result ranking. Bao *et al.* (2010b) presented object-level semantics to retrieve relevant results for queries in XML data. Bao *et al.* (2010a) proposed several updates to (Bao *et al.*, 2009) as an extension. In the work (Bao *et al.*, 2010a), they added the popularity of the results having comparable relevance scores.

Here, the result ranking schemes adopted in the existing XML keyword search techniques are described briefly. XRank (Guo *et al.*, 2003) computed LCAs to process XML keyword proximity queries and it has a ranking mechanism which returns document fragments as answers. XRANK is designed to generalize and apply the PageRank algorithm of Google to XML element level to rank all LCA results. However, an empirical study has not been carried out to demonstrate the effectiveness of XRank's ranking function. XSEarch (Cohen *et al.*, 2003) gives results which are fragments that bear semantic relationship. But XSearch requires that users have a basic knowledge about XML schema information. This, in turn, limits the query flexibility. EASE, proposed by Li *et al.* (2008), considers both the TF-IDF based ranking and structural compactness based ranking, for indexing and querying heterogeneous data. Termehchy and Winslett (2011) proposed a ranking method, coherency ranking, for XML keyword queries that is based on statistical measures of their cohesiveness.

Motivated by the research on XML keyword search by Bao *et al.* (2010a), a new keyword search approach for XML databases has been proposed, addressing the problems mentioned earlier in this section. The main objective of this research is to design and develop an efficient approach for keyword search that is considered to be an effective information discovery from XML data. The proposed approach primarily focuses on solving all the keyword ambiguity problems (Bao *et al.*, 2010a) in order to retrieve the most relevant information from the XML database. In addition, the grading of retrieved information is essentially needed to find the most suitable results for the query keyword given for searching. In order to handle these challenges, a grading measure is devised that will sort out the retrieved results to identify the better one.

METHODOLOGY

The keyword search approach for XML databases, namely, XDMA is presented based on dual indexing. The proposed approach identifies the type of query keywords using dual indexing technique and selects possible T-typed nodes. Furthermore, it includes design of mutual score, entropy-based similarity measure so as to find query results for a given XML keyword query and, grading score to grade the query results. The block diagram representing the stages of the dual indexing based XML keyword search approach, XDMA, is shown in Fig. 1.

The main contributions in the proposed approach are as follows:

- An indexing approach that builds two indices namely, tag_info table and data_info table for structural node and data node in XML database respectively, to simplify the query processing and distinguish the keyword as tag or data while searching a query keyword
- A keyword searching technique to select all possible T-typed nodes for a given query using the two-level matching between two indices
- By utilizing the logarithmic and probability functions, a terminology that defines the Mutual Score between selected tags and query keywords to find the exact T-typed node

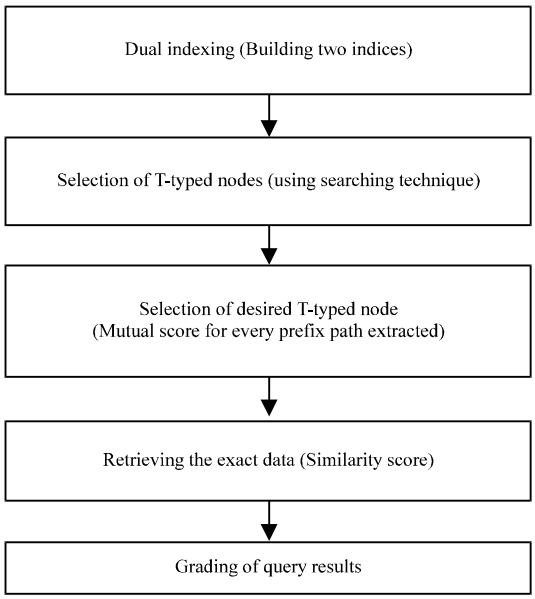

Fig. 1: Block diagram representing the stages of the dual indexing based XML keyword search approach

- An expression to count the similarity measure between the leaf nodes of XML data and the query, so as to retrieve the exact data through the selected T-typed node
- Grading for the query results that have comparable relevance score, based on the similarity measure of the query results

The proposed keyword search approach for XML databases, XDMA and its various stages are described in the following section.

XDMA: PROPOSED APPROACH

This section presents the proposed approach, XDMA, for searching query keywords in XML databases based on dual indexing. The indexing scheme includes a new keyword searching technique, design of mutual score measure, similarity function and similarity score measure to determine relevant path for a XML keyword query and grading score to rank the query results.

Dual indexing: In the earlier works (Bao *et al*., 2009, 2010a), with a frequency table, it is uncertain whether an input query keyword is a tag or data (text) value, causing query processing more complicated. To overcome this, the proposed approach XDMA could be used. The proposed approach constructs two indices, namely, tag_info table for tags and data_info table for data values. Moreover, these two indices are different from the indices in (Bao *et al*., 2009, 2010a).

For each tag in XML database, the tag_info table stores respective tag information, namely, tag name t_{sn}, frequency of occurrences of tag in T-typed nodes and their subtrees f_t and prefix path of the corresponding T-typed node, path. Also, for each data value in XML database, the data_info table stores the information of respective data value, namely, data value d, name of leaf tag containing data value t_{ln}, identification id of data value d contained in leaf tag t_{ln} and frequency of occurrences of data node contained in the corresponding leaf tag f_d. It is noted that both data_info table and tag_info table mutually share information with reference to tag name. The proposed approach deals separately with each tag of structural nodes and data (text) values of XML database so as to simplify the processing of query.

Identification of query keywords: For XML query search, each keyword of the query is first searched in the tag_info table. If it finds match(es) in the tag_info table, it is identified as tag keyword (k_t) and subsequently, information about keyword matching tag(s), i.e., t_{sn}, f_t and path will be retrieved from the tag_info table. Otherwise, query keyword is searched in the data_info table. If it matches with data value(s), it is identified as data keyword (k_d). For keyword matching data value(s), leaf tag t_{ln} will be obtained from the data_info table and subsequently, information about t_{ln}, i.e., t_{ln}, f_t and path will be retrieved from the tag_info table.

Finding T-typed node: For a given XML keyword query, T-typed nodes and the corresponding prefix paths will be determined based on the dual indexing technique explained earlier in this section. In XML database, the keyword matching tag (structural node) and the keyword matching data (text) value may occur once or many times. The Fig. 2 shows an example data tree with tag names and data values for dblp XML database. The keyword ambiguities (Bao *et al*., 2010a) in XML keyword queries can be explained with the following examples. As shown in Fig. 2, a keyword "September" appears as a data value of month in the path dblp, phdthesis and title in the path dblp, proceedings. A keyword "year" as shown in Fig. 2 appears as the XML tag name of phdthesis and proceedings. Also in dblp XML database, a keyword "journal" appear as both an XML tag name in the paths dblp, proceedings and dblp, article and a data value of title in the path dblp, inproceedings. As explained earlier in this section, the two separate indices for structural nodes (tags) and data (text) values of XML databases are built in the approach. Hence, XML keyword searching occurs only in these two indices.

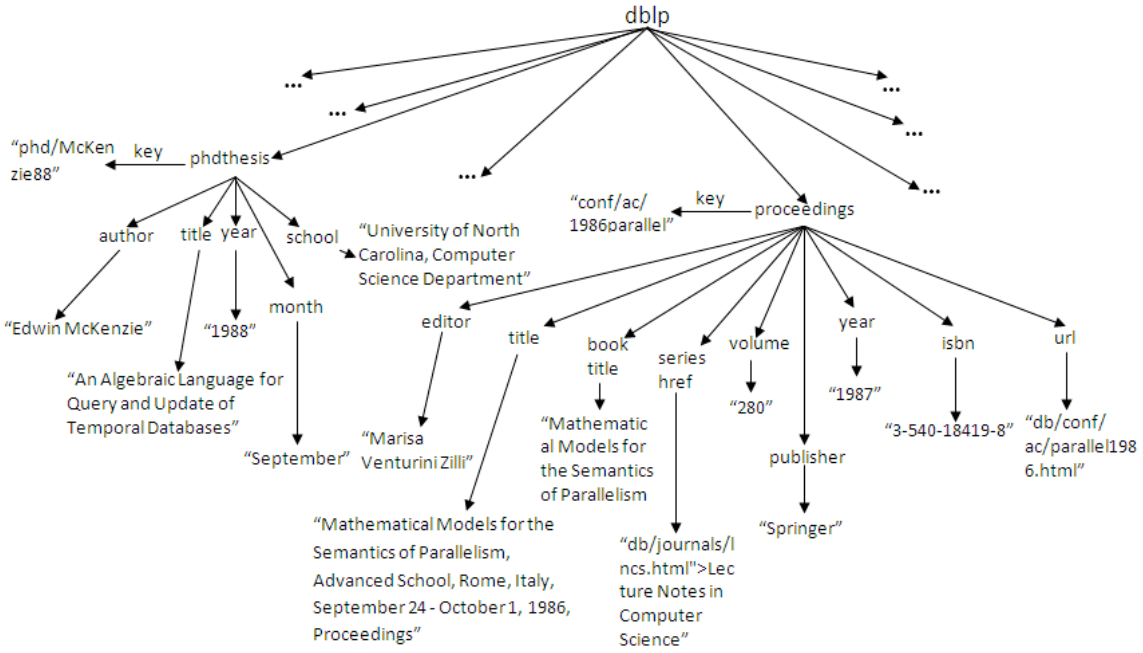

Fig. 2: An example XML data tree showing tag names and data values. Data values are given within quotes

If a query keyword finds more number of matches in XML database, query processing will be complicated. For that reason, mathematical formulae need to be devised to filter out the optimum T-typed node for a given keyword query. For a keyword query, the approach retrieves tag(s) matching with keyword and tag(s) containing data values matching with keyword from the two indices, namely, tag_info table and data_info table, respectively. In the approach, the two indices are dependent and share information.

Mutual summation is defined for a pair of random variables X and Y as follows:

$$\text{Msum}(X,Y) = \sum \log_{a}(c) \tag{1}$$

$$\text{Msum}(X,Y) = \sum \log_{(p(x)p(y))}(p(x,y)) \tag{2}$$

where c = p(x, y) is the joint probability distribution function of X and Y and a = p(x) p(y) are the marginal probability distribution functions of X and Y, respectively.

By applying change of base rule in (2):

$$\text{Msum}(X,Y) = \sum \frac{\log_{b}[p(x,y)]}{\log_{b}[p(x)p(y)]} \tag{3}$$

To make the mutual sum definition simple, a base of 10 to the log function could be used:

$$\text{Msum}(X,Y) = \sum \frac{\log_{10}[p(x,y)]}{\log_{10}[p(x)p(y)]} \tag{4}$$

By incorporating the dependence of two indices and concept of mutual sum, the mutual score between selected tags and query keywords is defined to find the exact T-typed node as follows:

$$\text{Mscore} = \left[\frac{\log_{10} f(\text{tag},\text{data})}{\log_{10}\left(\sum_{k \in q \cap T_t} f(\text{tag}) \sum_{k \in q \cap D} f(\text{data})\right)}\right] \times \text{rf}^{d(T)} \tag{5}$$

$$\text{Mscore} = \left[\frac{\log_{10} f(t_k,t_d)}{\log_{10}\left(\sum_{k \in q \cap T_t} f(t_k) \sum_{k \in q \cap D} f(t_d)\right)}\right] \times \text{rf}^{d(T)} \tag{6}$$

where, k stands for a keyword in the given query q, T_t represents tags in the tag_info table and D represents data values in the data_info table. In Eq. 6, $f(t_k)$ is the frequency of tags matching with keyword in each prefix path and $f(t_d)$ is the frequency of tags (leaf nodes) containing keyword matching data value. $f(t_k,t_d)$ is the frequency based on combination of tag and data keywords in the given query. rf is a reduction factor which can take a value between 0 and 1 and d(T) is the depth of T-typed nodes in XML documents. $\text{rf}^{d(T)}$ in Eq. 6 is used to reduce mutual score of the deeply nested node types in XML databases.

As explained earlier in this section, for each keyword matching tag in the tag_info table, the approach will retrieve tag names of structural nodes, frequency of the occurrence of each structural node in either T-typed nodes or their subtrees and, prefix path of the corresponding T-typed nodes from the tag_info table. The approach will add together frequency of each tag, matching with keyword in T-typed nodes and their subtrees. If the given query keyword matches with data value in the data_info table, the approach will retrieve the tag name of the leaf node of keyword matching data. For each retrieved tag, information such as tag name, frequency of the occurrence of each structural node in either T-typed nodes or their subtrees and prefix path will be retrieved from the tag_info table. Subsequently, the approach will add together frequency of tags (leaf nodes) of each keyword matching data. In such a way, both indices are dependent and share the information.

An input query contains one or more keywords matching with tag or data values. Accordingly, combined frequency f(tag, data) will be computed from the two indices. For every prefix path extracted, mutual score between retrieved tags will be calculated using Eq. 6. As mentioned earlier, both indices are in such a way that one index is dependent on other. Obviously, mutual score measures the information that the two indices share; also it measures how much knowing the tag_info table reduces uncertainty about the data_info table. The node with the highest mutual score is obviously selected as the desired T-typed node. However, data value matching with keyword and frequency of the data value contained in the leaf tag and the prefix path are not exclusively taken into consideration in the computation of mutual score. So, it is not certain that query keywords are in the prefix path with highest mutual score. Hence, mutual score is incorporated into similarity score as described in the following section.

Finding optimum T-typed node and relevant path: For a given query, XDMA will look for the similar data (text) values in the data_info table and then, the relevant tag of leaf nodes of these data (text) values will be found. Subsequently, the approach will collate the tag with the highest frequency with paths of selected T-typed nodes using the tag_info table. Moreover, the approach proposes a similarity score formula to estimate the similarity between nodes and keyword successively for each prefix path extracted with respect to query keyword.

For a random variable 'X' with n outcomes $\{x_i: i = 1\ldots n\}$, a similarity function S(x) is proposed and it is defined as follows:

$$S(x) = \sum_{i=1}^{n} [p(x_i) + \log_b(p(x_i))] \quad (7)$$

where, $p(x_i)$ is the probability mass function of outcome x_i.

By assuming the above similarity function, a new equation is defined to measure similarity among the leaf nodes of XML data and the query so as to determine the precise data through the selected node of T-type as follows:

$$S(x) = \sum_{i=1}^{n} [f_{q_i}^{T_a} + \log_{10}(f_{q_i}^{T_a})] \quad (8)$$

$$\text{Similarity Score} = \text{Mscore} + S(x) \quad (9)$$

where, in Eq. 8, n denotes the number of keywords in a query q and data node is denoted by a, whereas T_a represents the type of data node's parent node; $f_{q_i}^{T_a}$ represents the number of q_i with respect to T_a. In Eq. 9, Mscore is the mutual score of a pair of keyword matching tag and tag containing keyword matching data value. The proposed similarity score measure evaluates the accuracy of occurrence of query keywords path-wise in order to find relevant path for a query.

Grading: Grading of query results will be mapped to address the results containing comparable score. The grading of query is presented as the measure of the highest similarity score for each prefix path extracted with respect to query keyword.

$$\text{Grading (G)} = \text{Max } \{\text{similarity score}\}$$
$$G = \text{Max } \{\text{Mscore} + S(x)\} \quad (10)$$

where, in Eq. 10, Mscore and S(x) contain the same notations as in Eq. 6 and 8, respectively.

Therefore, Grading can be represented by:

$$G = \text{Max}\left\{\left[\frac{\log_{10} f(t_k, t_d)}{\log_{10}\left(\sum_{k \in q \cap T_t} f(t_k) \sum_{k \in q \cap D} f(t_d)\right)}\right] \times rf^{d(T)} + \sum_{i=1}^{n} [f_{q_i}^{T_a} + \log_{10}(f_{q_i}^{T_a})]\right\} \quad (11)$$

where, grading for the query results having comparable relevance scores is determined using Eq. 11. Path with maximum similarity score is graded as the most relevant and other paths will be graded successively based on similarity scores.

ALGORITHMS AND COMPLEXITY ANALYSIS

Data processing and index construction: The approach parses the input XML document and extracts the

information, namely, tag name of each structural node t_{sn}, prefix path for each structural node, path and data value of each data node d. Also, frequency of occurrences of structural node (path-wise) in T-typed nodes and their subtrees f_t and frequency of occurrences of a data value contained in a leaf node f_d, are computed. After preprocessing the XML document, two indices are built in the approach to expedite the keyword query processing. The first index built is called tag_info table, which stores tag name (structural node name) t_{sn}, frequency of occurrences of structural node (path-wise) in T-typed nodes and their subtrees f_t and prefix path of T-typed node, path. The worst case space complexity is $O(N_{sn})$, where N_{sn} is the number of structural nodes in XML database. Number of structural nodes in XML database is considerably high.

The second index built is called data_info table, which stores data value of data node d, leaf tag (node) containing data value t_{ln}, id of data value d contained in leaf tag t_{ln} and frequency of occurrences of data value contained in the corresponding leaf node f_d. In addition, a path_info table for data node is built to store path of d contained in t_{ln} based on id. Its worst case the space complexity is $O(D\times N_{ln})$ where D is the number of data nodes and N_{ln} is the number of leaf nodes in XML database. As mentioned earlier in the previous section, the data_info table stores d, t_{ln} and f_d. Hence, the size of the data_info table is reasonably higher than the tag_info table.

In addition, a path_info table for data node is built to store path of d contained in t_{ln} based on id.

Query processing: The main algorithm for the proposed approach XDMA is shown in Algorithm 1. It consists of three procedures, namely, get_IndexInfo, get_MutualScore and get_SimilarityScore. Initially, the get_IndexInfo gets the information about keyword matching tags and tag containing keyword matching data values for each query keyword. For each extracted prefix path, the get_MutualScore procedure compute the mutual score between keyword matching tags and leaf tags containing keyword matching data value. The procedure get_SimilarityScore calculates the similarity among the leaf nodes of XML data and the query to grade the relevant nodes.

Algorithm 1: Keyword search for desired node type

Input: Query *Q* containing *n* keywords
1 get_IndexInfo(*query keywords*)
2 get_MutualScore(*keyword match*)
3 get_SimilarityScore(*data keywords*)

The algorithm for finding index information, get_IndexInfo is presented in Algorithm 2. To identify whether each keyword is a tag or a data value, two-level matching between both indices is used. By calling function tagExist, the procedure get_IndexInfo checks whether a keyword exists in the tag_info table and also, by calling Function dataExist, it checks whether a keyword is in the data_info table (line 1-5). Then for each tag keyword (k_t), the procedure obtains the information of keyword matching tags from the tag_info table by calling function getFromTag_info and stores them in both taglist and slist (line 6-10). Also for each data keyword (k_d), it selects the name of leaf tag containing keyword matching data values from the data_info table and obtains leaf tag information from the tag_info table by calling function getFromData_info and, stores them in both datalist and slist (line 11-16).

Algorithm 2: Get_IndexInfo(query)

```
input: Query Q containing n keywords
//Identify each keyword as tag or data
1 for each keyword k_i ∈ Query do
2     if tagExist(k_i) = true then
3                 { k_i is a tag keyword}
4     else if dataExist(k_i) = true then
5                 {k_i is a data keyword}
//Store information of each tag in a taglist()
6 for each tag keyword ∈ Query do
7     rs = getFromTag_info (t_k)
8     while (rs.next)
9     result = t_sn + f_t + path
10    taglist(result); slist(result)
//Store information of each data in a datalist()
11 for each data keyword ∈ Query do
12    rs=getFromData_info(a_i)
13    while (rs.next)
14    result = t_ln + f_t + path
15    datalist (result); slist (result)
16 return slist();
```

It can be proved that the get_IndexInfo algorithm presented in Algorithm 2 precisely gets the information of keyword matching tags and leaf tags containing keyword matching data values for a given query as described in the previous section. Let the query be Q containing n keywords. The time complexity of the get_IndexInfo algorithm is $O(n+n_t\times k_{mt}+n_d\times k_{md})$. In the time complexity of get_IndexInfo, n denotes the number of query keywords, n_t is the number of k_t and k_{mt} is the number of keyword matching tags in the tag_info table. Here, n_d is the number of k_d and k_{md} is the number of leaf tags (in the tag_info table) containing keyword matching data value (in the data_info table). For the worst case only, the time complexity of the get_IndexInfo algorithm is computed.

The algorithm for computing the mutual score, get_MutualScore, is shown in Algorithm 3. The procedure get_MutualScore computes the mutual information between query keyword matching tags and leaf tags containing query keyword matching data values. First, it computes the sum of frequencies of keyword matching tags $f(t_k)$ by calling function getTagFrequency (line 1-2). Second, it computes the sum of frequencies of leaf tag

containing keyword matching data values $f(t_d)$ by calling function getDataFrequency (line 3-4). Then, it calls function getCombinedFrequency to compute the combined frequency of the tags t_k and t_d by executing the designed structured query based on the combination of tag and data keywords in input keyword query (line 5-11). Using the Eq. 6, it calculates the mutual score between t_k and t_d for every prefix path extracted (line 12-13). Finally it adds path and path-wise mutual score value to mlist (line 14).

Algorithm 3: Get_MutualScore(query, keyword match)

```
//Compute sum of frequencies of keyword matching tags
1 for each keyword matching tag ∈ Query do
2     f(t_k) =getTagFrequency(query)
// Compute sum of frequencies of leaf tag containing keyword matching data values
3 for each keyword matching data value ∈ Query do
4     f(t_d) =getDataFrequency(query)
//Compute combined frequency
5 Based on combination of tag and data in query, assign a value to cvalue
6 for each result ∈ slist() do
7     Split path from slist(); tagpath(path)
8     Based on cvalue, Construct queryString ; queries(queryString)
9     for (each queryString ∈ queries( )) do
10                d = getCombinedFrequency(queryString)
11                f(t_k,t_d) + = d
12    msum = log10(f(t_k,t_d))/log10(f(t_k)*f(t_d))
13 mscore = Msum*p^depth(T)
14    mlist(path, mscore)
15 return mlist();
```

It can be proved that the get_MutualScore algorithm shown in Algorithm 3 computes the mutual score between the retrieved tags and query key words to find desired T-typed node. As explained in the previous section, the time complexity of the algorithm is $O(n_t+n_d+k_m\times q_{cf})$ where k_m is the number of keyword matching tags and leaf tags containing keyword matching data values and q_{cf} is the number of queries to compute the combined frequency $f(t_k,t_d)$ based on the combination of tag and data in the query. In the worst case, for each k_t, frequency of keyword matching tags are added together to get $f(t_k)$. For each k_d, frequency of leaf tag containing keyword matching data values are added together to get $f(t_d)$. For each prefix path extracted, queries are executed to compute $f(t_k,t_d)$ based on keyword combination in query. It is noted that this is the worst case complexity and the get_MutualScore algorithm can finish earlier.

The algorithm for computing the similarity score, get_SimilarityScore, is shown in Algorithm 4. The procedure get_SimilarityScore looks for the similar data (text) values for every k_d from the data_info table by calling getFromData_info and stores d, t_{ln}, f_d and id of these data (text) values in dsimilar (line 1-5). For each similar data value, f_d of each keyword matching data value d stored in dsimilarl is compared with each other and leaf tag with biggest frequency is stored in stag in the form of $< t_{ln}, f_d, id>$ (line 6-8). For each leaf tag and its information stored in stag, the function callReferences is called to obtain f_d and path, through which leaf tag occurs, from path_info (line 9-12). This path is compared with each path stored in mlist. If it matches with path stored in mlist, the corresponding minfo is extracted from mlist (line 13-15). In line 16, frequency of occurrences of keyword matching data value contained in leaf node f_d is obtained. Using Eq. 9, Similarity Score is computed (line 17-18). Grading of query results is determined based on Eq. 10 (line 19).

Algorithm 4: Get_SimiliarityScore(data keyword)

```
//Find relevant tag for each data keyword
1 for each datakeyword d_ki ∈ Query do
2     rs = getFromData_info(d_ki)
3     while (rs.next)
4     result = d + f_d + t_ln + id
5     dsimilar(result)
6     for each result ∈ dsimilar() do
7     Get tag with biggest frequency; ltag= t_ln + f_d + id
8     stag(ltag)
//Compare each tag with selected Path
9 for each ltag ∈ stag do
10    rs = callReferences(id)
11    while (rs.next)
12 Get frequency f_d; Get path of ltag from path_info
13    for each data ∈ mlist do
14    if path in mlist = path from path_info then
15    Get mscore from mlist
16                f = f_d
17                S(x)= f + log10 f
18                SimilarityScore = mscore +S(x);
19 Grading = Max(SimilarityScore)
20 return SimilarityScore
```

The get_SimilarityScore algorithm shown in Algorithm 4 calculates the similarity score among the leaf nodes of XML data and the query. The similarity score is subsequently used to determine the precise data via the selected T-type node as explained in the previous section. Consequently, the time complexity of the get_SimilarityScore algorithm is $O(n_d\times k_{sd}\times d_s+n_{ttag}\times n_{path}\times p_{mscore})$. In the time complexity of get_SimilarityScore, n_d is the number of k_d and k_{sd} is the number of similar data (text) values in the data_info table for every k_d. Here, d_s is the number of similar data (text) values for all k_d and n_{ttag} is the number of tags selected from similar data (text) values for all k_d. In the time complexity of get_SimilarityScore, n_{path} is the number of selected tag paths and p_{mscore} is the number of the extracted path along with the computed mutual score. Also, the time complexity of the get_SimilarityScore algorithm is calculated for the worst case only.

In the proposed approach, XDMA, it is estimated that the worst case space complexity of tag_info table is $O(N_{sn})$ and the worst case space complexity of data_info table is

$O(D \times N_{ln})$. In XReal (Bao *et al*., 2010a), the frequency table has a worst-case space complexity of $O(K \times N)$. In this, K represents the number of keywords that are distinct and N represents the number of node types. It is shown that by using XDMA, in comparison with XReal, space requirement can be considerably reduced.

It is computed that, in the worst case, the time complexities of get_IndexInfo, get_MutualScore and get_Similarity Score of XDMA are $O(n+n_t \times k_{mt}+n_d \times k_{md})$, $O(n_t+n_d+k_m \times q_{cf})$ and $O(n_d \times k_{sd} \times d_s+n_{ttag} \times n_{path} \times p_{mscore})$ respectively. With regard to time complexity analysis, the comparison of algorithms could not be dealt in depth. This is because the time complexity of algorithms of XReal (Bao *et al*., 2010a) has not been given. However, the time complexity of XDMA can be shown to finish in less time. The proposed approach XDMA against the previous studies (Xu and Papakonstantinou, 2005; Liu and Chen, 2007; Bao *et al*., 2009, 2010a) in XML keyword search will be experimentally compared to demonstrate the effectiveness of XDMA.

CONCLUSION AND FUTURE WORKS

A dual indexing and mutual summation based keyword search algorithm, namely, XDMA for XML databases have been designed and developed. XDMA constructs two indices, namely, tag_info table and data_info table for each tag of structural node and data value of data nodes in XML databases respectively. Unlike earlier works in XML keyword search, the approach XDMA identifies whether each query keyword is a tag or a data value and moreover, deals with each (structural node) tag and data (text) value in XML database separately using the two indices in order to simplify query processing and resolve key word ambiguity problems. A searching technique of selecting all possible T-typed nodes for a given query applying the matching between both indices is developed. By incorporating the dependence of two indices and mutual summation, a formula is defined to compute mutual score of node type to be a search for and a similarity grading scheme to captivate the graded structure of XML databases. The future work includes experimentation on different XML datasets with various XML keyword search algorithms and evaluation of the results to prove the effectiveness of the proposed approach.

REFERENCES

Bao, Z., J. Lu, T.W. Ling and B. Chen, 2010a. Towards an effective XML keyword search. IEEE Trans. Knowledge Data Engine., 22: 1077-1092.

Bao, Z., J. Lu, T.W. Ling, L. Xu and H. Wu, 2010b. An effective object-level XML keyword search. Proceedings of the 15th International Conference on Database Systems for Advanced Applications, April 1-4, 2010, Tsukuba, Japan, pp: 93-109.

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.

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.

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.

Hristidis, V., N. Koudas, Y. Papakonstantinou and D. Srivastava, 2006. Keyword proximity search in XML trees. IEEE Trans. Knowledge Data Engine., 18: 525-539.

Li, G., B.C. Ooi, J. Feng, J. Wang and L. Zhou, 2008. EASE: An effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. Proceedings of the International Conference on Management of Data, June 9-12, 2008, Vancouver, Canada, pp: 903-914.

Li, J. and J. Wang, 2009. XQSuggest: An interactive XML keyword search system. Proceedings of the 20th International Conference on Database and Expert Systems Applications, August 31-September 4, 2009, Linz, Austria, pp: 340-347.

Li, J., C. Liu, R. Zhou and B. Ning, 2009a. Processing XML keyword search by constructing effective structured queries. Proceedings of the Joint International Conferences on Advances in Data and Web Management, April 2-4, 2009, Suzhou, China, pp: 88-99.

Li, J., J. Wang and M. Huang, 2009b. XKMis: Effective and efficient keyword search in XML databases. Proceedings of the International Database Engineering and Applications Symposium, September 16-18, 2009, Calabria, Italy, pp: 121-130.

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.

Liu, Z. and Y. Cher, 2008. Reasoning and identifying relevant matches for XML keyword search. Proc. VLDB Endowment, 1: 921-932.

Schmidt, A., M.L. Kersten and M. Windhouwer, 2001. Querying XML documents made easy: Nearest concept queries. Proceedings of the International Council for Open and Distance Education, March 14, 2001, Hong Kong, pp: 321-329.

Sun, C., C.Y. Chan and A.K. Goenka, 2007. Multiway SLCA-based keyword search in XML data. Proceedings of the 16th International Conference on World Wide Web, May 8-12, 2007, ACM New York, USA., pp: 1043-1052.

Termehchy, A. and M. Winslett, 2011. Using structural information in XML keyword search effectively. ACM Trans. Database Syst., Vol. 36.

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.