International Journal of Advanced Computer Technology (IJACT)
ISSN: 2319-7900 Volume VIII, Issue IV, August 2019

N-Gram Based Query Structuring System For Effective XML Retrieval

Roko Abubakar, Department of Mathematics, Faculty of Science, Usmanu Danfodiyo University, Sokoto, Nigeria.
Asma'u Shehu, Department of Mathematics, Faculty of Science, Usmanu Danfodiyo University, Sokoto, Nigeria;
Aminu Muhammad Bui, Department of Mathematics, Faculty of Science, Usmanu Danfodiyo University, Sokoto, Nigeria
Ibrahim Saidu, Department of ICT, Faculty of Engineering & Environmental Studies, Usmanu Danfodiyo University, Sokoto, Nigeria.

Abstract

Query structuring systems are keyword search systems recently used for the effective retrieval of XML documents. Existing systems fail to put keyword query ambiguity problems into consideration during query pre-processing and return irrelevant predicate nodes. As a result, these systems return irrelevant results. In this research, an XML keyword search system, called N-gram based XML query structuring system (NBXQSS) is developed to improve the performance of keyword searches. The NBXQSS uses an N-gram Based Query Segmentation (NBQS) method which interprets a user query as a list of semantic units to help resolve ambiguity. The system also introduces an improved predicate identification algorithm (IPIA) to return relevant predicates. The IPIA uses a proposed function to compute the query term proximity and ordering. The effectiveness of the NBXQS is demonstrated through experimental performance study on some real-world XML documents. The results show that the developed system performs better compared to the existing system in terms of precision.

Introduction

XML (Extensible Markup Language) refers to a standard for representing; publishing and exchanging data over the Internet, several documents are now represented and stored in XML format on the Web. These documents contain textual information and logical structures to highlight the underlying semantic. However, content-oriented XML retrieval becomes a challenge due to difficulty in selecting highly relevant elements that satisfy user's information needs. Thus, there is a need for a user-friendly and effective method for searching XML data over the Internet. These documents are searched using queries formed using traditional query languages such as XPath or XQuery (Nguyen & Cao (2012). Queries composed using query languages are called structured queries. Although searching using structured queries is effective, using query languages to express queries proves 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 be familiar with keyword searches and therefore prefer to use a keyword search query interface to search XML data. Keyword queries are inherently ambiguous, and it is difficult for users to state their intentions clearly. The ambiguity of the keyword query may cause a large number of results to be returned and thus makes keyword query not effective [10]. Therefore, search systems that enhanced the effectiveness of keyword queries are highly needed.

The question now is in order to improve the effectiveness of keyword queries, can we develop systems that allow users to enter keyword queries and relegate the task of generating the structured queries to the systems? The answer is yes, and such systems are currently called Query Structuring Systems [1, 2, 6, 8, 910, 12]. Query structuring systems convert a user keyword query into a set of structured queries and select the best-structured query or queries that match the given input query. These systems focus on how to represent user queries, identify user search intention, and ranking algorithms to improve keyword search. However, with respect to query representation, existing systems firstly return irrelevant interpretation of user query because the systems fail to put query keyword ambiguities into consideration. Specifically, none of the systems consider the following ambiguities: (I) a query term can appear as the text values of different XML nodes and having different semantics (ii) a query term can appear as both a tag name and as part of the text content of some node.

In addition, existing systems [2, 6] also return inappropriate predicate nodes due to the use of term frequency (tf) only to compute the predicate node but ignores query term proximity.

In this paper, an N-gram Based XML Query Structuring System (NBXQS) is developed to return the relevant interpretation of user query and predicate nodes. The NBXQSS uses N-gram Based Query Segmentation (NBQS) method to break user queries into segments in order to return the best user query interpretation. Also, the system employs an improved predicate identification algorithm (IPIA) that uses the output of NBQS as input and a proposed formula to infer relevant predicate nodes.

The rest of the paper is organized as follows: Section 2 describes the related works. Section 3 describes the proposed Query Term Proximity and ordering. Section 4 describes the proposed N-gram Based Query segmentation (NBQS). Section 5 presents the proposed Improved Predicate Identification Algorithm (IPIA). While Section 6 presents the proposed NBXQSS system. Section 7 presents a performance evaluation. Finally, Section 8 concludes the study.

Related Works

In this section, related works on Query structuring systems are presented. Several systems proposed to improve the effectiveness of keyword search are reviewed by highlighting the operational procedure of each system as well as its pros and cons. These systems are as follows:

The study by [8] designed an XML keyword search approach which they called XBridge. This system derives the semantic of keyword query and then generates a set of structured queries that are evaluated using an XML database. However, the ranking function of this system is based on tf-idf which ignores the semantics of XML data and the query. The function favors only fragments with high term frequencies.

The study [9], proposed a concept called XIO (smallest meaningful XML twig) and an algorithm (XIOF) to compute a set of XIO from a keyword user query. This algorithm improves both precision and recalls for a keyword search. However, the result returned is not in any particular order because it has no ranking function.

The study by [10], designed an XML keyword search approach that derives keyword query and generates a set of structured queries by analyzing a given keyword query and schemas of XML data sources. An algorithm for computing the ranking score of structured queries was also developed. Although the approach returns a ranked list of results, it also returns irrelevant results with higher term frequency due to the use of traditional tf-idf ranking function.

The study by [6] proposed a method called StruX to derive structured XML queries automatically from keyword-based queries. In the method, users are to specify a schema-independent and unstructured keyword-based query that generates a top-k ranking of schema-aware queries that are based on a target XML database. It also splits a user query keyword into a sequence of segments, where each segment consists of a query keyword or a sequence of keywords. However, the system ignores the fact that a query keyword can appear in different parts of an XML document having different semantics because it considers a query just as a sequence of keywords not as a sequence of semantically related terms. In addition, it also returns irrelevant results due to failure to consider the weight of a query term based on its position and query terms proximity.

The study [2] developed an Effective Keyword Query Structuring using NER for the XML retrieval system (we call it EKQS) which uses the EBQS method to interpret the user input query as a list of keywords and named entities. The method uses the NER tagger [7] to mark a sequence of query keywords that describes an entity and treat the sequence as a unit. However, the method cannot identify more than one named-entity of the same type that appears as a contiguous subsequence of query keywords because it marks all the sequence of keywords describing an entity as one entity. However, the method returns irrelevant interpretations of the user query. Consequently, the system returns irrelevant results to users.
To date, none of these systems has proved capable of further improving the effectiveness of the systems. The NBXQSS system proposed differs from the existing Query Structuring Systems because it introduces NBQS and IPIA methods to return relevant XML nodes.

A. Query Term Proximity and Ordering (QTPO)

This section describes the proposed QTPO function, which computes query term proximity score, similar to the BM25TP model [4]. Recall BM25TP model score query term proximity score based on an intuition that, in a relevant document, query terms appear relatively close to each other and not in completely unrelated parts of the document. Recall that the BM25TP model works as follows.

Given a user query$q = \{t_1, t_2, \dots, t_n\}$. With every query term t_i the model associates an accumulator $acc_d(t_i)$ that accumulates the term's proximity score within the current document d. For every query term t_x, the model grabs its posting list and computes the distance (number of postings) between this posting and the previous posting belonging to the term t_y. If $t_x \neq t_y$, then increment the accumulators for both terms according to the following formulas

$$acc_d(t_x) = acc_d(t_x) + w_{t_x} \frac{1}{(P_{t_{x,d}} - P_{t_{y,d}})^2} \qquad (1)$$

$$acc_d(t_y) = acc_d(t_y) + w_{t_y} \frac{1}{(P_{t_{y,d}} - P_{t_{x,d}})^2} \qquad (2)$$

where w_{tx} and w_{ty} is the inverse document frequency (idf) for query terms t_x and t_y respectively. $P_{t_{x,d}}$ and $P_{t_{y,d}}$ are the position of query term t_x and t_y in document d respectively. The accumulators in Equations (1) and (2) will be incremented only if $t_x \neq t_y$.

B. QTPO

This section illustrates the weakness BM25TP and shows how QTPO is derived. QTPO is based on the intuition that a relevant XML element is one that contains the query keyword in close proximity and in the same order as they appear in the query. First let's illustrate the weakness of BM25TP. Substituting $t_x = t_y$ in Equation (1), the equation becomes

$$acc_d(t_y) = acc_d(t_y) + w_{t_y} \frac{1}{(P_{t_{x,d}} - P_{t_{y,d}})^2} \qquad (3)$$

Subtracting Equation (3) from Equation (2), it implies that

$$(P_{t_{x,d}} - P_{t_{y,d}})^2 = (P_{t_{y,d}} - P_{t_{x,d}})^2 \qquad (4)$$

Equation (4) means that BM25TP ignores query terms ordering. As a result it assigns scores to the documents regardless of the query terms ordering and so it cannot differentiate between the queries: *Qiuyue Wang XML Retrieval* and *Wang Qiuyue Retrieval XML.* This restricts its functions and the restriction is solved using the proposed Equation (5). To address the query terms ordering problem in this paper, the proximity of query terms is calculated as follows: The function $d\left(P_{t_{x,e}}, P_{t_{y,e}}\right)$ of Equation (5) is proposed to replace the square difference $(P_{t_{x,e}} - P_{t_{y,e}})^2$ in Equations (1) and (2).

$$d(P_{t_{x,e}}, P_{t_{y,e}}) = \sqrt{\frac{(P_{t_{x,e}} - ave)^2 + (P_{t_{y,e}} - ave)^2}{2}} \qquad (5)$$

where $P_{t_{x,e}}$ and $P_{t_{y,e}}$ represent position of term t_x and t_y in element (e) respectively and *ave* is the harmonic mean of the two positions, which can be computed using the following formula, $ave = \frac{(\beta^2+1)P_{t_{x,e}}P_{t_{y,e}}}{\beta^2 P_{t_{x,e}} + P_{t_{y,e}}}$

(6)

We set β = 1.1, which is greater than 1 to ensure that query term t_x is clustered around t_y. The proposed function $d(P_{t_{x,e}}, P_{t_{y,e}})$ computes how scattered the terms in a query are within an XML leaf element. A small value of $d(P_{t_{x,e}}, P_{t_{y,e}})$, means the query terms are clustered together, while a large value indicates the query terms are widely separated. Therefore, by replacing the square difference $(P_{t_{x,e}} - P_{t_{y,e}})^2$ in Equations (1) and (2) by the function $d\left(P_{t_{x,e}}, P_{t_{y,e}}\right)$, the equations are transformed as follows:

$$acc'_d(t_x) = acc'_d(t_x) + w_{t_x} \frac{1}{d(P_{t_{x,e}}, P_{t_{y,e}})} \qquad (7)$$

$$acc'_d(t_y) = acc'_d(t_y) + w_{t_y} \frac{1}{d(P_{t_{x,e}}, P_{t_{y,e}})} \qquad (8)$$

In this paper, since in XML retrieval the granularity of search is XML elements (e), the term proximity score of a query term t_x is calculated at element (e) level using Equation (9).

$$ac'_e(t_x) = \begin{cases} acc'_e(t_x) + W_{t_x} \frac{1}{d(P_{t_{x,e}}, P_{t_{y,e}})}, t_x \neq t_y, t_x, t_y \in e, \\ acc'_e(t_x), otherwise \end{cases} \qquad (9)$$

C. N-gram Based Query segmentation (NBQS)

This section presents NBQS algorithm to return relevant interpretation of user input query. The algorithm first assumes that queries consist of phrases or semantic units and then re-express these queries as a list of semantic units. It uses the Web as a corpus of potential query phrases because the Google n-gram corpus contains the largest Web phrases [13]. The corpus consists of 1-gram, 2-gram, ..., 5-gram phrases from the 2006

Google index along with number of times each n-gram appears. Based on this number of times the algorithm scores each query's segmentation and outputs the "best" segmentation. The NBQS algorithm is shown in Algorithm 1 and works as follows:

Algorithm 1: NBQS

```
Input:   keyword query  -- q
Output:  segmentation   -- best_sgmtt

1.  tblSgmts = null       // segmentations and their scores
2.  segmentations =  genSegmtations (q)
3.  for each segmentation S in segmentations
4.      sscore = Score (S)      // using Equation 1.
5.      tblSgmts.add(S, sscore)
6     end for
7.  best_sgmtt = find_Max (tblSgmts)
8.  return best_sgmtt
```

Given a user query q consisting of n keywords, where $q = \{k_1, k_2, \ldots, k_n\}$, at line 2, the algorithm call the *genSegmtations()* method, which generates a set of valid 2^{n-1} segmentations from q. For example, the algorithm returns the second column of Table 1, for query *q = {xml retrieval Mounia Lalma}*. A Segmentation S consists of a list of segments and is valid if the concatenation of its segments equals q. Each segmentation S is assigned a score. Then, at line 3 all valid segmentations are listed, and for each segmentation S, line 4 computes a score for S according to Equation (10):

$$score(S) = \sum_{s\varepsilon S, |s|>2} |S|^{|s|} \; count(s) \qquad (10)$$

The multiplier $|S|^{|s|}$ rewards long segments compared to shorter ones in order to compensate the power law distribution of occurrence frequencies on the Web. For example, "Fox Development" has a much larger count than "Fox Development Team Microsoft", so that the multiplier helps us to avoid segmentations like "Fox Development", "Microsoft". The scores are shown in column 3 of Table 1. Line 5 stores the segmentation S and its score in a list called tblSgmts. At line 6, the algorithm iterates back to line 3 and takes the next segmentation S and repeats the process as done with the previous segmentation. When all the segmentations are considered, the algorithm transfers control to line 7. Line 7 calls the find_Max() method which returns the segmentation with highest score. Line 8 returns the segmentation. The computed segmentation would be used as input to IPIA. For example, for the query Q={xml retrieval Mounia Lalma}, the method generates eight segmentations after line 6 shown in Table 1. The third column of Table 1 represents the scores computed at line 4 using Equation (10). Therefore, the algorithm returns '*xml retrieval/Mounia Lalmas*' as the best segmentation on lines 7-8.

Table 1: Segmentations and scores

SN	Segmentation	Score
1	xml / retrieval / Mounia / Lalmas	0.00000
2	xml retrieval Mounia / Lalmas	0.000002
3	xml retrieval / Mounia Lalmas	3.104
4	xml /retrieval Mounia Lalmas	00097
5	xml retrieval Mounia Lalmas	0
6	xml /retrieval Mounia / Lalmas	0.000004
7	xml retrieval/ Mounia / Lalmas	2.3808
8	xml /retrieval/ Mounia Lalmas	0.7232

D. Improved Predicate Identification Algorithm (IPIA)

This section presents IPIA which resolves the problem of irrelevant predicates. It accepts list of segments as input. This algorithm is based on the intuition that relevant predicate is a predicate node that contains at least one keyword in a segment and contain the segment keywords in close proximity. Consequently, we firstly, propose Equation (11) to compute relevant predicates for a given segment.

$$score(n,s) = \log_e(1 + \sum_{k=s} acc_e{}'(t_k)\, f_{n,k}) \qquad (11)$$

Where n is a predicate node, s is a segment, and k is a keyword in s. The $acc_e{}'(t_k)$ is the proximity score of t_x in element e (see Equation (9)) and is the query term proximity score of term k in element n of segment s in node n. In the summation, the second multiplier ($f_{n,k}$) in Equation (11) computes XML nodes containing at least one of the keyword in s. While the first multiplier rewards keywords in the segment that appear in close proximity and in the order in node n. Then, IPIA uses the proposed Equation (11) to compute all relevant predicates. It accepts the best segments (segmentation) computed by the NBQS as input and outputs a list of predicates (predList) as output. IPIA is shown as Algorithm 2. The algorithm retrieves all the nodes in the XML database at line 1, and for each node, its relevant segments are computed from the given list of segments at lines 2-8.

Line 8 selects the best segments while lines 9-10 first check if the list of segments is non-empty then it generates a list of related node/segment pairs called predicates. This process continues until all the nodes are considered. Finally, line 12 selects the best predicates and returns the list of best predicates as the answer.

```
Algorithm 2:  EPIA

Input:    segmentation
Output:   predList

 1.  allnodes =  getAllNodes ()
 2.  for each node  n  in allnodes do
 3.      rsegments = null;        // list of relevant segments for node n
 4.      for each  segment s in segmentation do
 5.          $score = score(n, s)$        // using  Eq.11
 6.          if (score > 0) then rsegments.add(s, score)
 7.      end for
 8.      segment = selectBestSegment (rsegments)
 9.      if   (  rsegments ! = null)  then
10.          predList.add(n , segment, score)
11.  end for
12.  return    genbestPredicates (predList)
```

Example: Demonstrating how algorithm 2 works using Figure 1. Figure 1(a) shows *s1*, *s2*, and *s3* as a list of three segments (*segments*) passed as input to IPIA. Line1 call getAllNodes() method which retrieves all the nodes in the XML document. Suppose the method returns two nodes (*author* and *title*) shown in Figure 1(b).

(a) s1: xml retrieval
s2: John Michael
s3: retrieval John Michael

(b)
SigmodRecord/issues/issue/articles/article/authors/author
SigmodRecord/issues/issue/articles/article/title

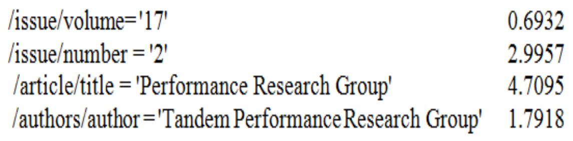
Figure 1: Three segments and two nodes

Line 2 iterates through all the nodes in *allnodes* and for each node n, a list of relevant segments (Rsegments) is computed from segments. Therefore on line 2, the IPIA takes node n = SigmodRecord/issues /issue/articles/article /authors/author, and then tries to find all relevant segments for n as follows: Line 3 sets rsegments to empty, meaning that relevant segments are not yet found. At line 4, IPIA considers segment s1and line 5 computes a score for s1 with respect to n. Lines 6 updates the Rsegments if the score is greater than zero on line 6, otherwise the algorithm iterates back to line 4. At line 7, IPIA iterates back to line 4 and considers s2.Then, line 5 computes a score for s2 with respect to n. Lines 6 updates Rsegments if the score is greater than zero on line 6, otherwise the algorithm iterates back to line 4. Again, at line 7, IPIA iterates back to line 4 and considers s3. Then on line 5 computes a score for s3 with respect to n. Lines 6 updates Rsegments if the score is greater than zero, otherwise the algorithm iterates back to line 4. Again, at executing line 7, IPIA iterates back to line 4. This time no segment is found and therefore the algorithm goes to line 8. At line 8 the algorithm to compute best segments for the underlying node n. It then stores n and its segments in predList as predicate at line 10. At line 11, the algorithm iterates back to line 2 to consider the next node. At line 2, n = SigmodRecord/issues/issue/articles/article /title. It then uses line 3-8 to compute its relevant segments as done for the previous node.The new node n and its segments are stored in predList as predicate at line 10. The process is repeated until all the nodes are considered in which case the algorithm move to line 12. At line 12 the algorithm selects the best set of predicates. Line 13 returns these predicates. Consequently, lines 5-6 compute a score for n and s2 pair. At line 6, if the score is non-negative, it mean we have found a relevant segment (s2) for node n, and s2 as well as its score are added to the list of relevant segments (Rsegments) for node n. This leaves Rsegments = {'John Michael': 2.66789}. This time, no segment is found. At this point, all candidate relevant segments for node n are computed and line 8 selects best segments. Since s2 is the only segment, line 9 returns s2 as the best segments. Lines 12-13 return a list of n-segment pairs called predicates. Line 12 stores each predicate in the list of predicates (predList). This leaves predList = {SigmodRecord/issues/issue/articles/article /authors/author = 'John Michael': 2.66789}, as a list of one predicate with the number 2.66789 as its corresponding segment score. At line 11, the algorithm iterates back to line 2 to consider the next node. At line 2, n=sigmodRecord/issues/issue/articles/article/title. Line 3 sets Rsegments to empty. Then, IPIA tries to find all relevant segments for the new node n. Line 4 takes segment s1 and tries to find if it is relevant to n on line 5. Since on line 5 Type(n) = Other, then s1 is related to n if all the keywords in s1 are of type 'Other'. All the keywords in s1 are of type 'Other', which implies that n and s1 are semantically related. Consequently, lines 6-7 compute a score for n and s1 pair. On line 6, if the score is non-negative, s1 as well as its score are added to the list of relevant segments (Rsegments) for node n. This leaves Rsegments={'xml retrieval': 3.0025}. Now, Rsegments is a list of one segment 'xml retrieval' with its score '3.0025'. Then IPIA iterates back again to line 4 and considers s2. Line 5 discovers that none of the keywords in s2 is of type 'Other', so n and s2 are not semantically related. Thus, all the keywords in s2 are removed, which leads to s2 = null. Since s2 = null, IPIA ignores s2 and iterates back to line 4 to take the next segment s3. The line 5 pre-processes s3 by removing all keywords in s3 that are not of type 'Other'. This caused the last two keywords in s3 to be removed which changes s3 to s3 = retrieval. Since the new s3 is not empty, lines 6-7 compute a score for n and the new segment s3. At line 6, if the score is non-negative, the new s3 as well as its score are added to the list of relevant segments (Rsegments) for node n. This generates Rsegments = { 'xml retrieval': 3.00125, 'retrieval': 2.0504}. Now Rsegments is a list of two segments 'xml retrieval' and 'retrieval'. The algorithm iterates back again to line 4 and found no segment, which means all candidate relevant segments for node n are computed. IPIA goes to line 9 to find the best segment out of the two relevant segments in Rsegments. To do that Line 8 calls the *selectBestSegment()* method which finds best segment. The method returns s1 = 'xml retrieval': 3.0025 as the best segment because it has the highest score. Lines 10-11 add this segment together with n as predicate in predList. This leaves us with predList={SigmodRecord/issues/issue/articles/article /authors/author = 'John Michael': 2.66789; SigmodRecord/issues/issue/articles/article/title = 'xml retrieval': 3.0025}, as a list of two predicates, where the first predicate is the one obtained in the first iteration. From line 11 the IPIA iterates back to line 2 to consider the next node. However, based on Figure 1(b), all the nodes are considered and therefore it goes to line 12. Line 12 calls the genbestPredicates() method which finds best predicates in predList. Both predicates are selected since both are of different type and both contain data of different type and thus line 12 returns predList={SigmodRecord/issues/issue/articles/article/authors/author = 'John Michael': 2.66789; SigmodRecord/issues/issue/articles/article/title = 'xml retrieval': 3.0025}. The following example demonstrates the effect of the proposed P_s^n in Equation (1).

Example: Consider a query '*17 2 Tandem Performance Research Group*' issued on Sigmod data, and the query is intended to search for an issue whose volume '17' and number '2' and one of the authors is 'Tandem Performance Research Group'.

Case 1: using Equation (2) without P_s^n

Figure 2 shows four predicates each with its scores returned by PIA algorithm.



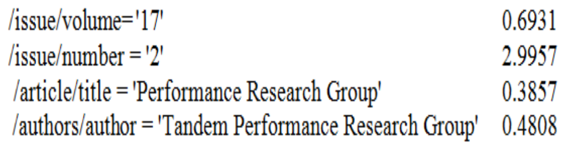
/issue/volume='17'	0.6932
/issue/number = '2'	2.9957
/article/title = 'Performance Research Group'	4.7095
/authors/author = 'Tandem Performance Research Group'	1.7918

Figure 2: Predicates and their scores

In this case, the third predicates got the highest score because there are more /article/title nodes in the Sigmod XML that contain the segment keywords than the /authors/author nodes. But the user search intention does not include /article/title node. 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. The user refers to those keywords in the '/authors/author' node. Meaning that the '/authors/author' node is supposed to have the highest score not /article/title node. Therefore, an irrelevant predicate is returned because the equation fails to consider the semantics of the keywords (i.e. ambiguity ii).

Case 2: using Equation (2) with P_s^n

Figure 3 shows four predicates each with its scores returned by IPIA algorithm.



/issue/volume='17'	0.6931
/issue/number = '2'	2.9957
/article/title = 'Performance Research Group'	0.3857
/authors/author = 'Tandem Performance Research Group'	0.4808

Figure 3: Improved predicates and their scores

In this case, the /authors/author node now has the highest score when compared with /article/title node. This is because the segment keywords 'Tandem Performance Research Group' appear closer to each other in the /authors/author node while the segment keyword 'Performance Research Group' appear scattered in the /article/title node. Therefore, the /authors/author node got the highest score of P_s^n, which helps to improve its score in Figure 2.

E. N-gram Based XML Query Structuring System (NBXQSS)

This section presents the proposed NBXQSS which returns relevant XML elements. The system is shown in Figure 4. It consists of a pipeline of other algorithms namely: N-gram based Segmentation (NBQS) algorithm, Predicate Identification Algorithm (IPIA), Compute Return Node Algorithm (CRNA), Query Formulation Algorithm (QRYFv). The pseudo code of the NBXQSS is shown in Algorithm 3.

Algorithm 3: NBXQSS

```
Input:  document collection, keyword query -- qry
Output: ranked list of XML fragments -- sub trees

1.  segments = NBQS( qry)
2.  predicates  =  EPIA (segments)
3.  return_node =  CRNA ( predicates)              //adopted
4.  strucQueries = QRYFv (return_node, predicates) //adopted
5.  sub trees  =  evaluate (strucQueries[k])
```

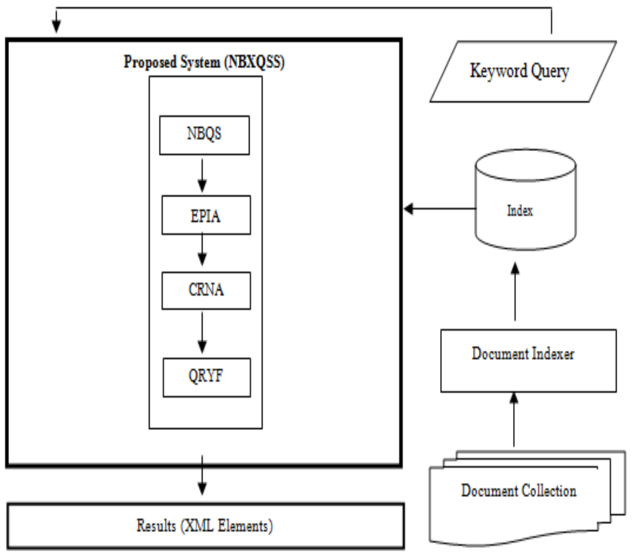


Figure 4: The NBQXSS System

The algorithm consists of three stages: Search intention identification, query formation and query processing. Firstly, lines 1-3 present search intention identification, which for a given user query finds the predicates and the return node. Secondly, line 4 presents the Query formation, which uses the QRYFv algorithm to generate a set of structured queries and returns a ranked list of structured queries. Finally, line 5 evaluates the queries on a target XML database. The CRNA and QRYFv are adopted from our previous work in Roko et al (2015). We refer our esteem readers to the article. The CRNA computes the returned node which is the target XML node the user is searching for. The QRYF receives a set of predicate nodes and return nodes. It then converts the set of predicate nodes and returned nodes to a set of structured queries in XQuery syntax. The structured queries are used for the retrieval of results.

F. Experiments

The experiments are conducted using real dataset and query sets to compare the performance of the proposed NBXQSS and EKQS.

Experimental Setups

The two systems are implemented in Java and run on a 3.2.0 GHz an Intel (R) Core (I3) machine with 4GB of RAM running Windows 10 Professional 64 bit operating system. MySQL database is used to store the proposed index while Berkeley DB for XML (Brian, 2006) used to store and query the XML documents.

Dataset

Sigmod and IMDB document collections are used as the datasets. Since the NBXQSS returns entity nodes from the datasets as the nodes users are searching node, the IMDB dataset is pre-processed before indexing. This is because in the IMDB dataset, every document root, such as <person> or <movie> is intuitively an entity node. The proposed system considers a root element as not suitable to be an entity node. Consequently, to return a <person> as an entity node, we automatically create a dummy root node <persons> to contain all the <person> elements. An entity node consists of a set of related leaf nodes. For example, for the IMDB XML dataset there are 42 leaf nodes. According to (Cohen, S., Mamou, J., Kanza, Y., & Sagiv, 2003), the smaller the number of leaf nodes in an entity node the closer the leaf nodes and therefore more meaningful. So, in this paper, five (5) leaf nodes are selected out of the 42 leaf nodes based on a survey involving twenty postgraduate students. In the survey, each respondent is given the list of the 42 leaf elements and asked to select the ones often used to describe a movie or an actor. .

Query and judgments

To get the queries and relevant judgments, 25 keyword queries are randomly selected as done in XIOF (X. Li et al., 2010) and a survey involving thirty people is also conducted. In the survey, the people were asked to write the target XML predicate nodes and XML return nodes that would be return by each query. The survey result is summarized and the 10 queries where more that 70% of the participants agree on the same returned nodes are selected. Tables 3 and 4 shows the queries selected on columns 2 and their corresponding returned nodes (i.e. relevant judgments) on columns 3 for IMDB and Sigmod datasets. Also, each keyword query is manually changed to its corresponding XQuery expression and the new query is used to retrieve relevant XML fragments. These retrieved relevant fragments are used to judge the quality of the XML elements return the proposed system. This paper assumes that a query keyword has at least one occurrence in the XML data being searched. Some of these queries contain ambiguities; query QI5 contain ambiguity (ii); Queries QS5 and QS6 contain ambiguity (I). All the datasets are indexed as in (Roko A., et al., 2018).

Results and Discussions

The section describes experimental results. It first presents the returned nodes obtained by the EKQS and the proposed NBXQSS. Then, it also presents the quality of the XML elements returned by the NBXQSS and the EKQS systems.

Quality of the returned nodes

Tables 2 and 3 also show the query evaluation results on IMDB and Sigmod datasets, respectively. In the Tables, the second columns represents the queries, the third columns show the target returned nodes (i.e. the relevant judgments), the fourth columns show the returned nodes obtained by the proposed NBXQSS, the fifth columns show the return nodes computed by the EKQS. The Tables show that the proposed NBXQSS outperforms the EKQS in terms of the quality of target returned nodes.

Table 3: Query evaluation result on IMDB dataset

Qid	Query	Returned node	NBXQSS	EKQS
QI1	Chris I Arnett Super Fly	Person	Person	Person
QI2	murder McGill Bruce	Movie	Movie	Movie
QI3	Sheltered Life 2008	Movie	Movie	Movie
QI4	countdown Lewinsky Monica	Movie	Movie	Movie
QI5	Garafano Michelle director	Movie	Movie	director
QI6	Planet Revenge Evil Ninja	Movie	Movie	Movie
QI7	Morris Haviland actor	Movie	Movie	Movie
QI8	Zion Brother accident	Movie	Movie	Movie
QI9	Proudly family	Movie	Movie	Movie
QI10	Briggs Johnny Martin Joe	Movie	Movie	Movie

Table 1: Query evaluation result on Sigmoid dataset

QId	Query	Returned node	NBXQSS	EKQS
QS1	semantic database Victor Vianu	article	article	article
QS2	David DeWitt	article	article	author
QS3	multimedia object manager	article	article	articles
QS4	2 client server	issue	issue	issue
QS5	Fox Development Team Microsoft	article	article	authors
QS6	17 2 Tandem Performance Group	issue	issue	issue
QS7	24 2 Georges Gardarin	issue	issue	issue
QS8	Performance evaluation	article	article	article
QS9	Server	article	article	articles
QS10	databases Won Kim Willis Luther	article	article	article

Figure 4 and 5 show the precision comparisons of the IMDB and the Sigmod datasets, for the proposed NBXQSS and EKQS, respectively. The Figures show that the proposed NBXQSS achieves superior search performance than EKQS system because of the following reasons:
During query processing, the proposed NBXQSS establishes relationships between the keywords to convert the query into a list of semantic units using its query segmentation method. After this, it then finds predicate nodes containing the semantic units via its IPIA algorithm. IPIA includes a formula, which ensures that a score is assigned to each predicate node such that predicate nodes containing the semantic units as they appear

in the query are rewarded with higher scores and are selected as the best predicate nodes. Figures 5 and 6 illustrate the precision comparison of the proposed NBXQSS and EKQS using Sigmod and IMDB datasets, respectively. The Figure shows that the proposed NBXQSS achieves better search performance than the EKQS. Figure 5 shows that the NBXQSS is able to infer about 100% of the true return nodes while the EKQS only 90% on IMDB dataset. While Figure 6 demonstrates that the NBXQSS infers about 100% of the return nodes and the EKQS about 70%, on Sigmod dataset. The proposed NBXQSS achieves superior search performance than the EKQS system because of the following reasons.

The reasons for poor performance of EKQS are: (1) EKQS used heuristics to compute entity nodes present in the data. However, some of the entity nodes returned by EKQS are grouping nodes which are not entity nodes [11]. For example, for the queries *QS3* and *QS5*, the EKQS returns *articles* and *authors* nodes as the answer respectively. However, the two nodes are grouping nodes and hence are not entity nodes. While in the case of query QI5, EKQS returns “Director” node because it consider the whole query words as one entity.

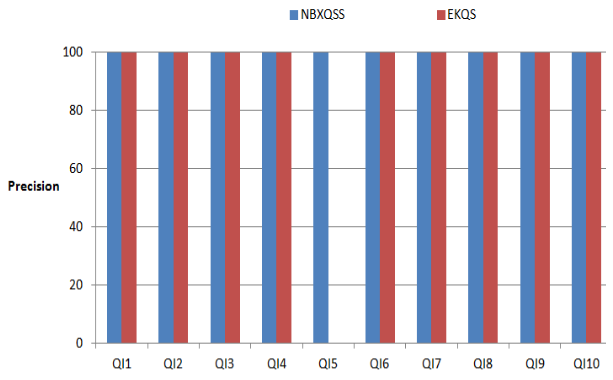


Figure 5: Precision comparison for IMDB.

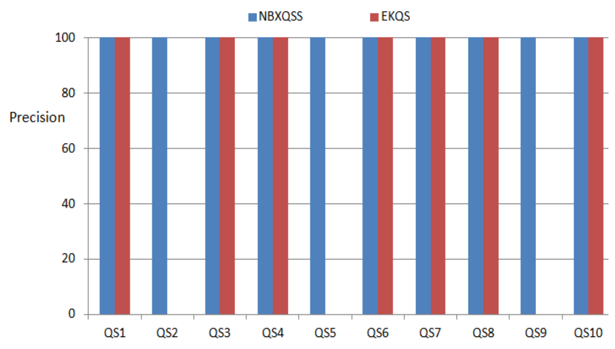


Figure 6: Precision comparison for Sigmod

G. Conclusion

In this paper, an NBXQSS system has been proposed for keyword search over XML document to improve the effectiveness of the returned results. The system employs NBQS method to resolve query keyword ambiguity. It also introduces an IPIA to return relevant predicates which helps to return informative entity nodes. Extensive experiments have been conducted to evaluate the performance of the proposed NBXQSS compared to the existing system. The results demonstrated that the NBXQSS out performs the compared EKQS in terms of the quality of the desired returned nodes.

References

[1] Abubakar Roko, et al, “Effective Keyword query structuring using NER for XML retrieval”. International Journal of Web Information Systems. Vol 11, PP 33-53 .,(2015). ISSN:1744-0084. DOI:10.1108/IJWIS-06-2014-0022.

[2] Abubakar Roko , et al, "Named Entity Based Ranking with Term Proximity for XML Retrieval”, International Journal of Information Retrieval Research. Vol 8. Issue 2. .,(2018). SSN: 2155-6377DOI: 10.4018/IJIRR.2018040104.

[3] Akritidis, L., Katsaros, D., & Bozanis, P. (2012). Improved retrieval effectiveness by efficient combination of term proximity and zone scoring: A simulation-based evaluation. *Simulation Modelling Practice and Theory*, *22*, 74–91. http://doi.org/ 10.1016/j.simpat.2011.12.002.

[4] Büttcher, S., Clarke, C. L. a., & Lushman, B. (2006). Term proximity scoring for ad-hoc retrieval on very large text collections. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR ’06* (p. 621). http://doi.org/10.1145 /1148170.1148285.

[5] Brian, D., “*The Definitive Guide to Berkeley DB XML*”. (N. Sixsmith, Ed.). New York, New York, USA: Apress. (2006)

[6] Da C. Hummel, F., Da Silva, A. S., Moro, M. M., & Laender, A. H. F. , “Automatically generating structured queries in XML keyword search”. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 6932 LNCS, pp. 194–205) 2011. http://doi.org/ 10.1007/978 -3-642.

[7] Jenny R. F., et al , “Incorporating Non-local Infor-

mation into Information Extraction Systems by Gibbs Sampling". Proceedings of the 43nd Annual Meeting of the Association for Computation Linguistics (ACL 2005), pp. 370. http://nlp.stanford.edu /papers/gibbscrf3.pdf.

[8] Li, J., Liu, C., Zhou, R., & Ning, B. (2009). Processing XML Keyword Search by Constructing Effective Structured Queries. *Advances in Data and Web Management.*

[9] Li, X., Li, Z., Chen, Q., & Li, N. (2011). XIOTR :A Terse Ranking of XIO for XML Keyword Search. *Journal of Software*, *6*(1), 156–163. http://doi.org/10.4304/jsw.6.1.156-163.

[10] Li, X., Li, Z., Wang, P., & Chen, Q. (2010). XIOF: Finding XIO for Effective Keyword Search in XML Documents. In *2010 2nd International Workshop on Intelligent Systems and Applications* (pp. 1–6). Ieee. http://doi.org/10.1109/IWISA.2010.5473249.

[11] Nguyen, K., & Cao, J. (2010). Exploit Keyword Query Semantics and Structure of Data for Effective XML Keyword Search. In *Proceedings of the Twenty-First Australasian Conference on Database Technologies* (Vol. 104, pp. 133–140).

[12] Petkova, D., Croft, W. B., & Diao, Y. (2009). Refining Keyword Queries for XML Retrieval by Combining Content and Structure. *Advances in Information Retrieval*.

[13] Lin, Y., Michel, J.-B., Aiden, E. L., Orwant, J., Brockman, W., & Petrov, S. (2012). Syntactic annotations for the Google Books Ngram Corpus. *Proceedings of the ACL 2012 System Demonstrations*, (July), 169–174. Retrieved from http://www.aclweb.org/anthology/P12-3029.

Biographies

ABUBAKR ROKO received his B.Sc. degree (1991). in Mathematics from the Usmanu Danfodiyo University, Sokoto, Nigeria, in 1991, the M.Ss. degree in Computer Science from the Abubakar Tabawa Balewa University Bauchi, Nigeria in 1995, and the Ph.D. degree in Computer Science from University Putra, Malaysia in 2016. Currently, He is a senior Lecturer in department of Mathematics Usmanu Danfodiyo University. His teaching and research areas include Algorithm analysis, Information Retrieval/Recommender Systems, and Sentiment Analysis Dr. Roko Abubakar may be reached at: abroko@yahoo.com, roko.abubakar@udusok.edu.ng.