www.ietdl.org

Published in IET Software
Received on 6th May 2011
Revised on 26th October 2011
doi: 10.1049/iet-sen.2011.0082

ISSN 1751-8806

Efficient top-*k* algorithm for eXtensible Markup Language keyword search

H. Yu Z.-H. Deng N. Gao

Key Laboratory of Machine Perception (Ministry of Education), School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, People's Republic of China
E-mail: zhdeng@cis.pku.edu.cn

Abstract: The ability to compute top-*k* matches to eXtensible Markup Language (XML) queries is gaining importance owing to the increasing of large XML repositories. Current work on top-*k* match to XML queries mainly focuses on employing XPath, XQuery or NEXI as the query language, whereas little work has concerned on top-*k* match to XML keyword search. In this study, the authors propose a novel two-layer-based index construction and associated algorithm for efficiently computing top-*k* results for XML keyword search. Our core contribution, the two-layer-based inverted Index and associated algorithm for XML keyword search take both score-sorted-sequence and Dewey ID-sorted-sequence into consideration, and thus gain performance benefits during querying process. The authors have conducted expensive experiments and our experimental results show efficiency advantages compared with existing approaches.

1 Introduction

In recent years, the ability to compute top-*k* answers to eXtensible Markup Language (XML) queries is gaining considerable attention owing to the rapid growth of XML repositories. Top-*k* query evaluation on exact answers is appropriate when the answers are large and users are only interested in the highest-quality matches. Currently research on top-*k* answers to XML queries mainly concentrate on structural query languages, such as W3C's XQuery 1.0 and XPath 2.0 Full-Text or NEXI [1–4] that combine content conditions on elements with structural conditions like tag names and paths. However, motivated by the prevailing search engines, such as google and yahoo, keyword search has been proven to be friendly and widely accepted by common users. Compared to structure-based queries, keyword-based queries do not request the users to acquaint the exact structure of the XML documents, or previous knowledge for the corresponding query language, which is accessible to naive users. Nevertheless, keyword query also has its own disadvantages: since the query only contains a bag of words, it lacks expressivity to offer structural information for searching results in hierarchical XML documents. In addition, keyword search would also induce inherent ambiguity when interpreting the semantics in XML data. Thus we need to automatically connect the matched nodes in a reasonable way in order to incorporate users' intention and effectively identify the desired return information. The results of XML keyword search have been deeply discussed. One essential point of view for keyword-query results to XML documents is that the query results should achieve both completeness and accuracy. To achieve this goal, started from the notion of Lowest Common Ancetor (LCA), many corresponding papers have been published to demonstrate query semantics, such as Exclusive Lowest Common Ancestor (ELCA) [5], Meaningful LCA (MLCA) [6], Smallest LCA (SLCA) [7], Grouped Distance Minimum Connecting Tree (GDMCT) [8], XSeek [9], Valuable LCA (VLCA) [10] and so on. These methods are raised to improve the efficiency and effectiveness of results for keyword queries in XML documents. All of these approaches concentrate on the granularity of nodes in XML documents, which is in contrast to the granularity of the whole documents in traditional html keyword search. Thus top-*k* applied in XML keyword query introduces some new challenges. Classic top-*k* algorithms could not be incorporated into XML keyword search seamlessly. One crucial problem is that we have to dynamically construct the potential structure for the sub-tree results through given keywords on query processing, which is seldom taken into consideration in former corresponding top-*k* research. This disadvantage calls for efficient strategy to implement sub-tree generation during query process.

So taking all the aspects raised above into account, we make the following contributions to implement adaptive top-*k* algorithm in SLCA-based XML keyword search:

1. We carefully design a two-layer-based inverted (TLI) index structure to store XML documents for efficiently querying top-*k* answers through XML keyword search.
2. Based on the index structure we raised, we introduce corresponding algorithm named XTop for XML keyword search, which generates top-*k* results through keyword queries efficiently.

3. We conduct extensive experiments and the experimental results show that our approach exhibits efficiency advantages compared to naive approach without top-k strategy or other realisations for top-k applied in XML keyword search.

Although XTop algorithm was first proposed briefly in [11], this paper makes additional progress as follows.

1. We thoroughly discuss the work related to Top-k systems and deeply introduce the prelimitaries such as the data model, query semantic and BM25 score function.
2. We explicitly introduce and discuss the TLI index and the associated algorithm XTop. The content used for explaining and analysing these two parts becomes doubled compared with [11].
3. Extended experimental results have been reported. We examine the efficiency of our Top-k algorithm with other two baselines on the dataset of DBLP and Wikipedia. The scalability of the algorithm is tested on four groups: keywords with low frequency, medium frequency, high frequecy and mixed frequency.

In Section 2, we discuss related work dealing with top-k applied in XML keyword search field. Section 3 mainly talks about some preliminaries such as data model, query semantic and corresponding score function for XML sub-tree results. In Section 4, we illustrate the approach for index construction, and associated XTop algorithm will be proposed in Section 5. Our experimental results will be discussed in Section 6. The final section, Section 7, will draw conclusion and discuss future work.

2 Related work

As far as we know, some previous works have taken efforts to resolve top-k applied in XML keyword search. We notice that Top-X system [3, 4] and XRank system [5] have discussed the similar issue.

2.1 Keyword queries applied in Top-X system

Top-X search engine is generally a framework for unified indexing and querying large collections of unstructured, semi-structured, and structured data. It has integrated efficient scheduling and dispatching algorithms for top-k ranked retrieval with corresponding scoring models. Top-X has discussed the issue of querying methods by two categories – content-and-structure (CAS) query and content-only (CO) query. As [2] has depicted, a typical example for a CAS query is like the NEXI query //article// section[about(., 'XML database') and (. // figure, 'architecture')] that searches for article sections about XML database (which is a content condition) that include a figure about the architecture (i.e. that has this term somewhere in its content). And a typical example for a CO or wildcard queries like //*[about (., 'XML')] do not restrict the tag of target elements – which is similar to keyword queries in text retrieval. Thus CO in Top-X system is generally an approximate approach to XML keyword search. However, since top-X system do not assign specific algorithm for CO query and CAS query, respectively, CO query still 'borrows' the structure-based CAS query process which performs O ($|S_1|\ldots|S_K|$) complexity to obtain the results within one document (suppose given a list of keyword $w_1, \ldots, w_K$ and $S_{i(1=<i<=k)}$ denotes the list of nodes that directly or indirectly contains w_i in one document). Since the prevailing XML keyword search algorithms such as Stack Algorithm [5] and IL (Index Lookup Eager Algorithm) [6] have gained linear complexity for generating sub-tree-based results such as SLCA and so on, thus it is not an efficient approach for top-X's realisation for top-k generation in XML keyword search. What is more, top-X could not construct sub-tree automatically without structural information, but merely returns individual nodes that contains the corresponding keywords instead of sub-tree results. So top-X is not an effective approach to generate top-k ranked results through keyword search.

2.2 Keyword queries applied in XRank system

XRank [5] is designed to query over a mix of HTML and XML documents. XRank is meant to handle some novel features of XML keyword search such as (a) the results are deeply nested XML elements instead of entire documents, (b) the notion ranking is at the granularity of an XML element instead of a document and (c) the notion of keyword proximity is more complex in the hierarchical XML data model. XRank system also attempts to obtain the top-k answers by employing Ranked Dewey Inverted List (RDIL) raised in [5]. In their estimation, the approach for applying top-k strategy to XML keyword search is to order the inverted lists by the score of each individual element, which is calculated previously. In this way, higher ranked elements are likely to appear first in the inverted lists, and query processing can usually be terminated without scanning all of the inverted lists. However, since the inverted lists are ordered by scores instead of by Dewey IDs, it calls for an extra construction for efficiently obtaining the common ancestor for elements. What is more, since the inverted lists are organised at the granularity of elements, it has to maintain a list of some length organised on elements and apply more random access to the inverted lists to obtain the corresponding nearest nodes to generate the sub-tree results [5]. As has discussed in [3], owing to the characteristic of hardware situation, random access should be carefully used in order not to cost efficiency loss. Taking these defects into account, in our approach, our algorithm takes advantage of the two-layer-based index construction and incorporates corresponding efficient algorithm for efficiently generating top-k results. Corresponding experimental comparison will be shown in Section 6.

3 Preliminaries

In this part, we mainly talk about three issues as preliminaries for our work. They are data model, query semantic and score function.

3.1 XML data model

The XML is a hierarchical format for data representation and exchange. An XML document consists of nested XML element starting with the root element. Each element can optionally have nested sub-elements, values or attributes. We treat the attributes and values as the children of the corresponding element. Fig. 1 is just an example of XML document, modelled as a dom tree.

We use Dewey ID to encode elements in XML documents. Dewey ID is a prefix-based encoding. With Dewey ID each element is assigned an id that represents its relative position

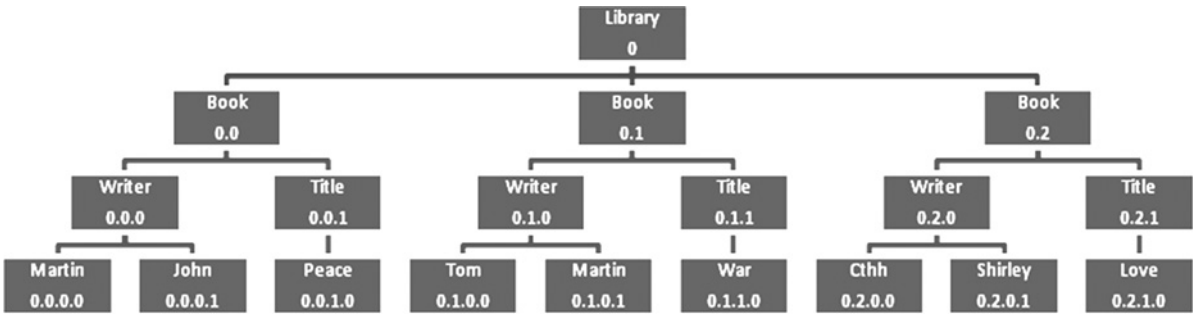

Fig. 1 *Library.xml*

among its siblings. The bit sequence from the root to an element unique identifies that element. Dewey ID captures ancestor-descendant information as well. With Dewey ID we could easily obtain the relationship of arbitrary two nodes. For example, in Fig. 1, 0.0.1 is 0.0.0's sibling node, and 0.0 is 0.0.1's ancestor. Dewey ID has been widely applied in many XML-related algorithms and demonstrated to be a good approach to represent and cope with XML data. In our index and corresponding top-k generating algorithm, we employ Dewey ID as our labelling method.

3.2 Query semantic

Keyword search in text documents takes the documents that are most relevant with the input keywords as answers. Respectively, for keyword search in XML documents, prevailing opinions on this issue focus on returning meaningful compact connected sub-trees which contain all the keywords as results. This calls for us to automatically connect the match nodes in a meaningful way and identify the desired return information. In tree-based model, LCA semantics [12] is the basic. Started from the notion of LCA, many corresponding papers have been published to demonstrate query semantic, such as ELCA, SLCA, VLCA and so on. These methods are raised to improve the efficiency and effectiveness of results for keyword queries in XML documents. Among all these structure-based query semantic, SLCA is a frequently referenced query semantic and serves as a crucial result option. Thus here let us have a glance at the definition of SLCA. The result of SLCA model must satisfy two restrictions:

1. The results of SLCA should contain all keywords either in their labels or in the labels of their descendant node.
2. They have no descendant node that also contains all keywords.

SLCA has its own reasons: more specific results connect keywords more closely and so they could offer a better explanation about the query. Take the XML tree in Fig. 1 as an example, each node is encoded with its corresponding Dewey ID. Suppose the query keywords are 'martin' and 'john', then node 0.0.0 with tag 'writer' is returned as SLCA result whereas node 0 with tag 'library' is not. Since node 0.0.0.0 contains keyword 'martin', node 0.0.0.1 contains keyword 'john', thus node 0.0.0 contains all these two keywords and compensates the 'smallest' principle according to SLCA definition. In the meanwhile, node 0 with tag 'library' is not taken as a SLCA node since one of its descendants (node 0.0.0) has already contains all keywords, thus node 0 should not meet up the 'smallest' requirement to emerge as a SLCA node. In addition to these attempts made on automatically connecting the match nodes in a meaningful way, Liu and Chen [9] have attempted to effectively identify the desired information that accord with the users' intention. However, existing approaches focus on node selection in the sub-trees generated SLCA nodes, acting as a post-processing after original sub-tree result generation. Thus in this paper, we mainly introduce how to incorporate SLCA query semantic into our top-k ranked result generation strategy. In fact our architecture can also be applied to the other sub-tree-based query semantics as we have listed above, since our realised approach is flexible and sub-tree generation process is, respectively, independent of other processing steps, which will be illustrated later in Section 6.

3.3 Score function

In a search engine, the top-k algorithm is used to improve the efficiency, and the score function is the key part determining the effectiveness (precision and recall). In particular, the critical issue of score function applied in XML documents is to evaluate the inner hierarchy of XML tree into consideration. The score function should reflect both content information and structural information to obtain an overall score for each term and its corresponding ancestor tags. Another factor we have to notice is that a basic restriction for prevailing top-k algorithms calls for monotonous scoring function to aggregate each individual keyword's score. Among the many proposed score functions, we consider XML-specific variant of Okapi BM25 referenced in [4] is an applicable approach that meet all the requirements raised above. The definition of Okapi BM25 is as follows

$$\text{score}(A//''t_1 \ldots t_m'', e) = \sum_{i=1}^{m} \frac{(k_1+1)\text{ftf}(t_i, e)}{K + \text{ftf}(t_i, e)} \times \log \frac{N_A - \text{ef}_A(t_i) + 0.5}{\text{ef}_A(t_i) + 0.5}$$

$$\text{with } K = k_1\left((1-b) + b\frac{\text{length}(e)}{\text{avg}_{e'}\{\text{length}(e') | e' \text{ with tag } A\}}\right)$$

Here we will not bother to talk about its principle and realisation any more. More detailed information could be found in [4]. Thus in our realisation, we employ the score function which is also applied in [3, 4]. In the following parts, we will simply use S(tag, term) to denote score computed for a pair of tag and term, and our index construction and system algorithm will also be stated based on pre-computed scores.

doi: 10.1049/iet-sen.2011.0082

4 Index construction

4.1 Pre-processing on single large document

For dealing with various data sets published in the web, we often meet up with some document that is pretty large in size for a single XML document. For example, the DBLP document is 487 M in a single document, whereasthe Wikipedia (English version) is over 1 G in size. Since the query results are often just a small sub-tree or a fragment of the XML document tree, it is inaccessible to return the whole large document or a sub-tree that contains too many elements. We attempt to take some pre-measures to deal with this kind of single large XML document. One intuitive method is to split the large XML documents into small ones. However, this needs careful check in order not to break the inner relationship and structural information that the XML documents hold. We have attempted to employ some IR concepts dealing with mark-ups in XML documents for automatically applying file-splitting. Researches on applying IR concepts to XML data has been studied for several years [1, 8, 13] and introduces several concepts into XML contexts and mark-ups. Here we employ some ideas in [13]. As [13] have defined, node papers in Fig. 2 is a structural node since it is labelled with a tag name and paper nodes are multi-valued nodes since it has more than one occurrence within its parent node. We consider that the parent of multi-valued nodes is just taking structural function and uniting the nodes with the same tag. Based on this thought we parse single large XML document and split them into small pieces by taking multi-valued nodes as the root of individual small XML documents. Take DBLP dataset as an example. The size of the entire DBLP data set is 487 M in a single document. The root of the original DBLP document is 'dblp'. The root node with tag 'dblp' has 296 282 children nodes. Among the 296 862 documents, 212 271 are rooted with node 'inproceedings'; 79 615 are rooted with node 'article'; 3007 are rooted with node 'proceedings'; 1009 are rooted with node 'incollection'; 845 are rooted with node 'book'; 72 are rooted with node phdthesis; 37 are rooted with node 'www' and four are rooted with node 'masterthesis'. Through parsing dblp data set and split it into small pieces by taking multi-valued nodes as the root of individual small XML documents, we obtain 29 6282 small documents, which range from 1 k to 10 M. However, the effectiveness for this operation lies on the structural characteristic of the original XML documents, but it indicates to be a beneficial approach for resolving the defects for dealing with single large document.

4.2 Index construction

In this part we will propose our TLI index design for indexing XML documents. The aim of the index construction should not only contain the structural information of the XML documents but also support efficiently obtaining top-k query results according to associated query algorithm. Motivated by this thought, we extend the classical per-term inverted index to a TLI index construction for XML keyword search, as shown in Fig. 3.

The left part in Fig. 3 is the first level of index construction, which is the same as the classical per-term inverted index with keywords as index entry. Each item in the inverted list is a table containing three elements (docID, max score, offset), in which docID is the identifier of XML documents in data set, and max score is the highest score in the corresponding Document block which we will illustrate later, and offset is an integer to tag the starting position of its corresponding Document block as shown in the right part in Fig. 3. The inverted list is organised in descending order with regard to maxscore. In the following we will first illustrate the construction and content of the Document Block. Then we will show how to generate maxscore for corresponding docID.

The Document block is the second level of index construction. Generally each Document block is corresponding to document docID and it is also an inverted list in ascending order with regard to Dewey ID. Each item in the inverted list of Document block contains two elements and each item is associated with a linked list that records the scores of all its ancestors. Let us take the XML document in Fig. 4 as an example to illustrate how to generate Document block with keyword 'forward' as index entry.

To begin with, we parse this document and compute scores for each tag-term pair. Suppose now the keyword that we are dealing with is 'forward'. We compute a series of scores for all elements that directly or indirectly contains keyword

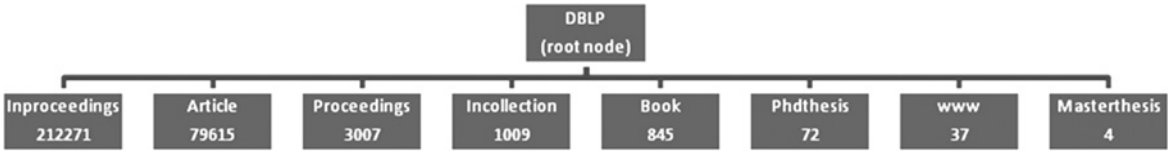

Fig. 2 *Statistics of DBLP data set*

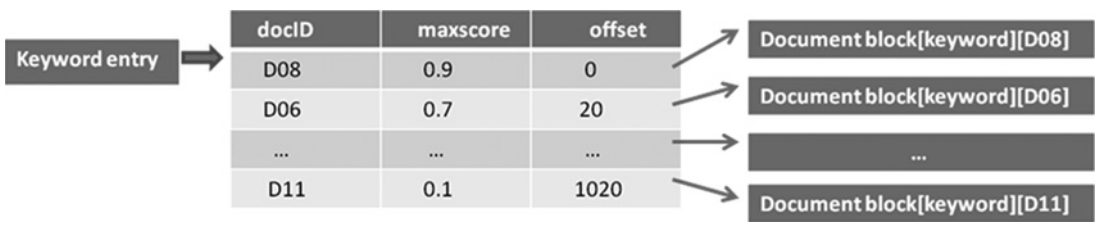

Fig. 3 *TLI index construction*

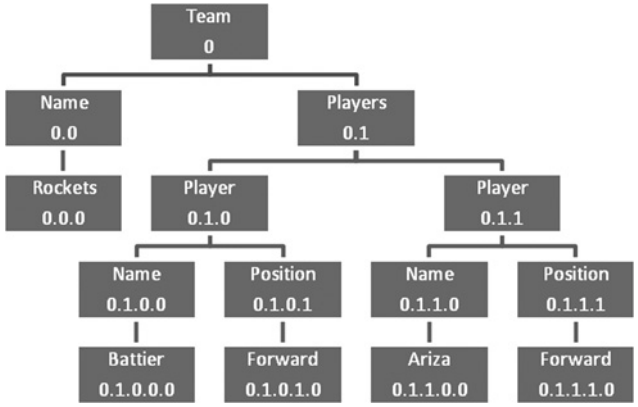

Fig. 4 *Example of an XML document to index*

Table 1 Scores of tag-term pairs for keyword 'forward'

S (tag, term)	*S* (Dewey ID, term)	Score
S('position', ' *forward*')	*S* (0.1.0.1, '*forward*')	0.5
	S (0.1.1.1, '*forward*')	0.5
S('player', ' *forward*')	*S* (0.1.0, '*forward*')	0.4
	S (0.1.1, '*forward*')	0.4
S('players', ' *forward*')	*S* (0.1, '*forward*')	0.3
S('team', ' *forward*')	*S* (0, '*forward*')	0.3

'forward'. We compute *S*('position', 'forward'), *S*('player', 'forward'), *S*('players', 'forward'), *S*('team', 'forward'). We can also use Dewey ID to represent the associated tag name. However, one tag name may correspond to several Dewey IDs if tag name occurs more than once. The scores for keyword 'forward' are shown in (Table 1).

In Fig. 4, keyword 'forward' appears twice, hence there are two items in the Document Block for keyword 'forward'. Each item is a table (Dewey ID, text), here text is the text content of the corresponding node of Dewey ID. We order the two items in ascending order by their Dewey ID for Document block. Each item maintains a linked list to record the scores for all its ancestors. The scores in the linked lists are computed from *S*(tag, keyword) and the sequence of the linked list is in descendant-ancestor order. The following Fig. 5 is the construction of Document block for keyword 'forward', denoted as Document block['forward']['rocket.xml'].

To calculate the maxscore of the Document block, we just need to obtain the maximal score of all scores in the linked lists. In Fig. 5, maxscore of keyword 'forward' for the Document block is 0.5. This maxscore is also added into the inverted list for corresponding document docID in Fig. 3.

5 Algorithm

After demonstrating the index construction for XML data, we present the corresponding XTop algorithm that copes with top-k generating process through keyword search (Fig. 6). The input of the algorithm is a bag of keywords and the output is sub-tree results associated with the query keywords. The algorithm process is constituted of four steps:

1. Access to keyword inverted lists.
2. Index scheduling.
3. Sub-tree generation.
4. Top-k answers selection.

In the following we will demonstrate algorithm processing of our algorithm based on data flow.

1. Access to keyword inverted lists: Our query processing method is based on pre-computed TLI index and the first level is sorted in descending order with regard to maxscore in each item. On the run time our algorithm sees an item t in one inverted list at each scan step, and then the algorithm performs random access to pick items in other inverted lists with the same docID as t.docID. For each inverted list we assign a thread to implement sequential access and random access, and each thread is in charge of recording the current scan position and ready to fetch the next item.
2. Index scheduling: Besides the two thread proposed in step 1, there is also another thread that is in charge of scheduling decision in interleaved manner. At the end of each scan step, this part delivers all the corresponding items of the same docID together with their associated Document block to the sub-tree generator module. What is more, at each scan step, the algorithm also makes a prediction for the potential best score that the unscanned items may reach. The predication for bestscore is calculated as follows

$$\text{bestscore} = \sum_{i=1\ldots m} \text{high}_i$$

3. Here m is the number of inverted lists and high_i is the maxscore of each item located at the upper bound in the unvisited parts of the index lists. We will use bestscore together with worstscore generated from step 3 to implement candidate pruning and early termination.
4. Sub-tree generation: The sub-tree generator module is to generate the corresponding SLCA nodes from Dewey ID lists and also calculates the corresponding score for the SLCA result nodes by summarising all the individual score of each keyword. The sub-tree generator is implemented in IL (Index Lookup Eager Algorithm) algorithm raised in [7]. The IL algorithm calculates the SLCA results of several

Dewey ID	text
0.1.0.1.0	forward
0.1.1.1.0	forward

Linked list for all ancestor tags			
name	player	players	Team
0.1.0.1	0.1.0	0.1	0
0.5	0.4	0.3	0.1
name	player	players	Team
0.1.1.1	0.1.1	0.1	0
0.5	0.4	0.3	0.1

Fig. 5 *Example of document block for keyword 'forward'*

doi: 10.1049/iet-sen.2011.0082

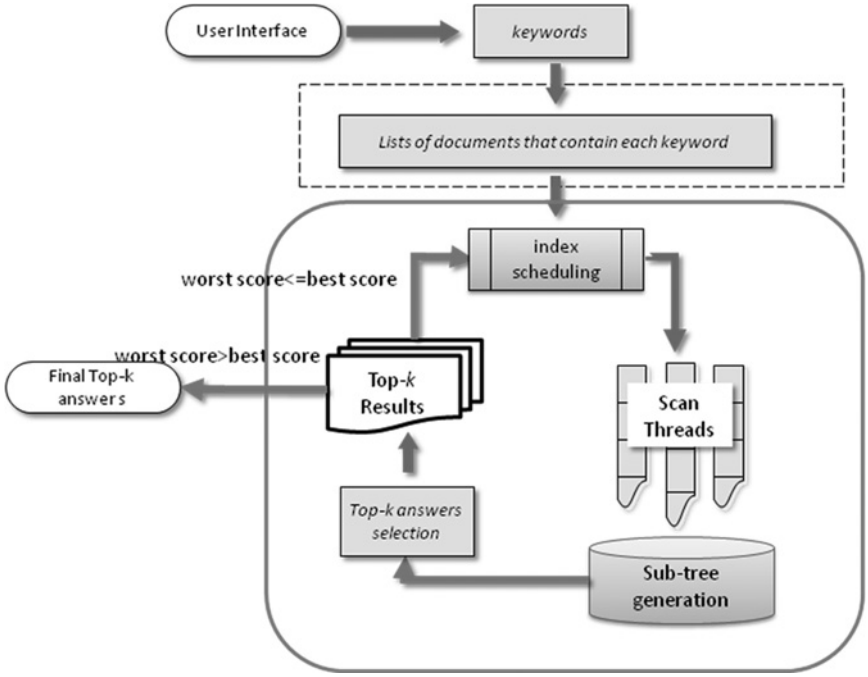

Fig. 6 *XTop algorithm structure*

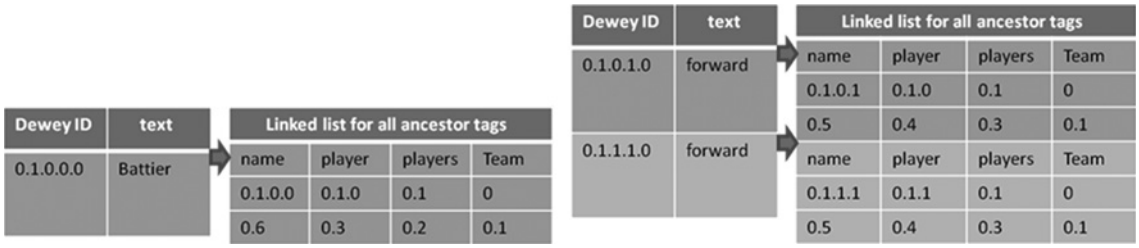

Fig. 7 *Document blocks for keyword 'Battier' and 'forward'*

nodes with their Dewey IDs. The input of the IL algorithm is several lists of Dewey IDs that are in ascending order each, just as we have organised for each Document Block. Thus we could directly take the Document blocks as input of the algorithm. At the same time, to obtain the score for each SLCA result, we aggregate its partial score for each keyword by tracing the linked list for all ancestor tags corresponded to each item in Document blocks. As an example, suppose two query keywords are 'Battier' and 'forward' in Fig. 4, and their corresponding Document Blocks are shown as follows in Fig. 7.

5. Through IL algorithm, we would know that 0.1.0 is their SLCA node generated from 0.1.0.0.0 of 'Battier' and 0.1.0.1.0 of 'forward'. So to calculate the score for the SLCA node 0.1.0, we trace along both linked list of '0.1.0.0.0' and linked list of '0.1.0.1.0' to find node 0.1.0 and its corresponding score. Thus we find the final score for node 0.1.0 is 0.7, summarised from 0.3 for keyword 'Battier' and score 0.4 for keyword 'forward'.

6. Top-k answers selection: The top-k answer selection is to preserve the k highest scores evaluated at current scan step. In our realisation, we employ a min-heap for dynamic value change to generate current top-k results from last scan step's top-k results combined with the newly calculated results from sub-tree generator. We take the minimal score in the top-k answers at current scan step as worstscore. Now we obtain the worstscore of the current top-k results and the bestscore for predication of best score among unvisited items from step 1. Thus each time at the end of step 3 we would apply candidate pruning and test for possible early termination. If worstscore is no less than bestscore, the algorithm can safely terminate and output the top-k results in top-k answer selector. Or else the algorithm goes back to step 1 to continue the next scan step.

XTop Algorithm

```
1  for all index lists Li (i=1 …m) do
2     item := (docID, maxscore, offset)    // scan next item
3     id := item.docID
4     for j = 1…m
5        itemj := RA(listj, id)            // get all items with docID =id
6        Sj := getDocumentBlock(itemj)
7        DListj := getDeweyList(Sj)
8     end for
9     bestscore := Σj=1…m high_j
10    SLCAResult := IL(DList1, ..., Dlistm)
11    getScore(SLCAResult)
12    TopKResult := getTopK(TopKResultOld, SLCAResult)
13    worstscore := getMin(TopKResult)
14    TopKResultOld := TopKResult
15    if worstscore <bestscore
16       Output(TopKResult)
17    else
18    return to 2
19 end for
```

Fig. 8 *Xtop algorithm*

The corresponding algorithm pseudo code is shown in Fig. 8.

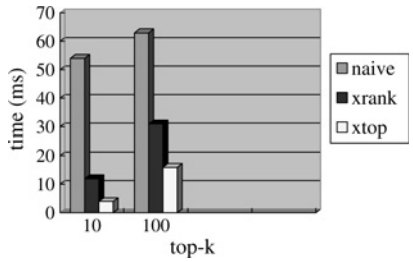

Fig. 9 *Comparison with keywords of low frequency on DBLP*

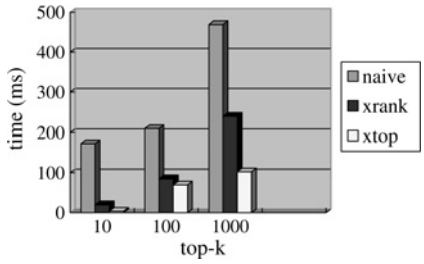

Fig. 10 *Comparison with keywords of medium frequency on DBLP*

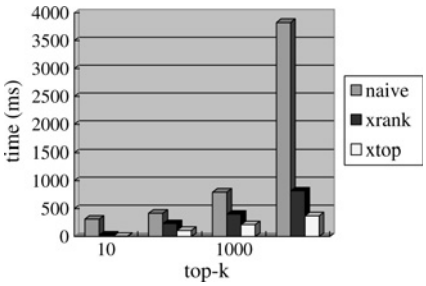

Fig. 11 *Comparison with keywords of high frequency on DBLP*

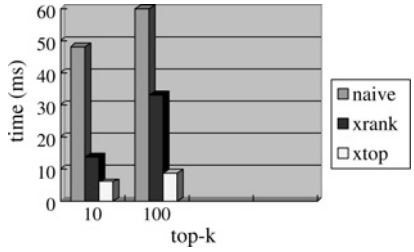

Fig. 12 *Comparison with keywords of low frequency on Wikipedia*

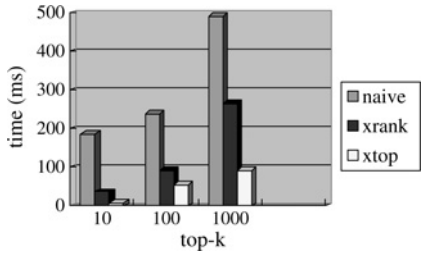

Fig. 13 *Comparison with keywords of medium frequency on Wikipedia*

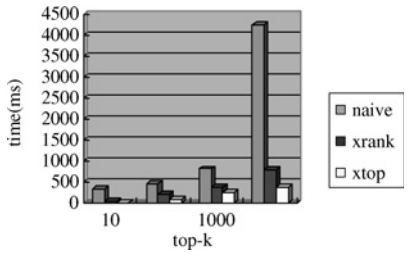

Fig. 14 *Comparison with keywords of high frequency on Wikipedia*

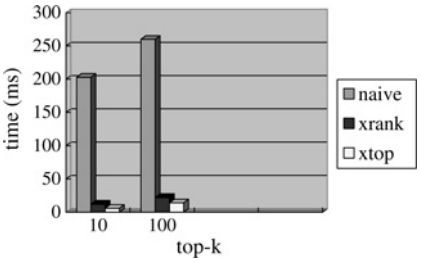

Fig. 15 *Comparison with keywords of mixed frequencies on DBLP*

6 Experiments

In this part we evaluate our algorithm presented in this paper through different groups of experiments. We use java for our implementation on a 2.00 GHz server with 8.00 GB of RAM. The data set we make use of for our experiments are DBLP and Wikipedia (English version). The size of DBLP data set is 487 M in size and Wikipedia is 1 G. Through pre-processing on single large document proposed in Section 4.1, we have split the original large single documents into small pieces of document fragments. What is more, through statistical analysis on data sets, we have devided keywords into three groups, which are of small frequency (ranging from 1 to 500), of medium frequency (ranging from 500 to 5000) and of high frequency (ranging more than 5000). We have evaluated and compared naive approach to obtain top-k results (naive approach means just going through all the inverted lists to obtain global scores of all items for obtaining the top-k results), XRank and our XTop algorithm with SLCA as query semantic (other query semantics could be easily incorporated into our system as we have demonstrated in Section 5) by a variety of keywords with different frequencies. We have chosen several queries of different frequencies for experimental evaluation.

Figs. 9–14 show the response–time comparison of naive approach, XRank and XTop algorithm for different groups of keyword frequencies on DBLP and Wikipedia data set, respectively. Each query contains keywords of the same frequency group and the response time for Figs. 9–11 is the average of response time after several executions. Figs. 9–11 are corresponding to low, medium and high frequency keyword queries, respectively. The same happens in Figs. 12–14. From these three diagrams we could see that XTop algorithm and XRank perform better in generating top-k results, and as the keyword frequency becomes higher, this trend exhibits more obvious. This conclusion is intuitive and accord with our original estimation since naive approach just goes through all inverted lists and then picks the top-k evaluated results,

doi: 10.1049/iet-sen.2011.0082

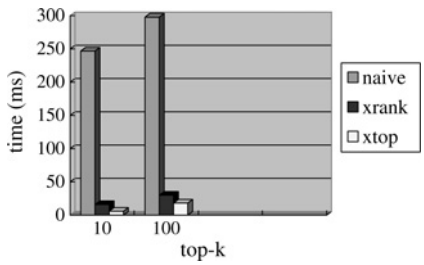

Fig. 16 *Comparison with keywords of mixed frequencies on Wikipedia*

whereas XTop and XRank benefits from early termination on processing instead of scanning through all indexes.

In Figs. 15 and 16, each query contains several keywords from different frequency groups. From Figs. 15 and 16 we could see that with this combination of keywords, XTop algorithm and XRank also perform better than naive approach. The experimental results on different groups show the good scalability of our XTop algorithm.

In the meanwhile, Figs. 9–16 all indicate that XTop algorithm performs better compared to XRank, but not as obvious as compared to naive approach. We consider this happens due to that since XRank makes use of top-*k* strategy so as to make candidate pruning, thus it also benefits from early termination without scanning through all indexes and performs better on query processing than naive approach. But our algorithm benefit from the advantage of obtaining all results of one document at a time, dealing with a batch of elements at linear complexity cost, whereas XRank just copes with one element at a time, applying too much random access to index lists. Thus our algorithm gains superiority at querying response time.

7 Conclusion and future work

We have presented the design, implementation and evaluation of our algorithm for adaptive top-*k* method applied in keyword search. Our experimental evaluation shows that our index for XML data and associated algorithm offer significant performance benefits and outperform former approaches.

There are several interesting issues for future work. First, our algorithm considers the XML data is hierarchical, taken as tree-based model. For structured (or semi-structured) data, the XML documents may be normalised, in which case the data model may be a graph with the consideration of IDREFs and XLinks [5]. Second, we assume that our algorithm can both apply sequential and random access to index lists; however, there may be some situation that restricts random access or totally forbidden. Thus more flexible top-*k* strategy is about to study in the future. Third, we will extend our algorithm to combine various score functions. Final, we will also further discuss the possibility that relaxing search terms and, in particular, tag names, by using ontology or thesaurus-based similarities.

8 Acknowledgments

This work was partially supported by Project 61170091 supported by National Natural Science Foundation of China and Project 2009AA01Z136 supported by the National High Technology Research and Development Program of China (863 Program). We are also grateful to the anonymous reviewers for their comments.

9 References

1 Chinenyanga, T., Kushmerick, N.: 'Expressive retrieval from XML documents'. Proc. Int. Conf. Research and Development in Information Retrieval (SIGIR 2001), Louisiana, USA, September 2001, pp. 163–171

2 Amer-Yahia, S., Lakshmanan, L., Pandit, S.: 'FleXPath: Flexible structure and full-text querying for XML'. Proc. Int. Conf. Management of Data (SIGMOD 2004), Paris, France, June 2004, pp. 83–94

3 Theobald, M., Schenkel, R., Weikum, G.: 'An efficient and versatile query engine for TopX search'. Proc. Int. Conf. Very Large Data Bases (VLDB 2005), Trondheim, Norway, August–September 2005, pp. 625–636

4 Theobald, M., Bast, H., Majumdar, D., Schenkel, R., Weikum, G.: 'TopX: efficient and versatile Top-*k* query processing for semi-structured data', *VLDB J.*, 2008, **17**, (1), pp. 81–115

5 Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: 'XRANK: ranked keyword search over XML documents'. Proc. Int. Conf. Management of Data (SIGMOD 2003), CA, USA, June 2003, pp. 16–27

6 Li, Y., Yu, C., Jagadish, H.: 'Schema-free XQuery'. Proc. Int. Conf. Very Large Data Bases (VLDB 2004), Toronto, Canada, August–September 2004, pp. 72–83

7 Xu, Y., Papakonstantinou, Y.: 'Efficient keyword search for smallest LCAs in XML database'. Proc. Int. Conf. Management of Data, MD, USA, June 2005, pp. 537–538

8 Hristidis, V., Koudas, N., Papakonstantinou, Y., Srivastava, D.: 'Keyword proximity search in xml trees', *IEEE Trans. Knowl. Data Eng.*, 2006, **18**, (4), pp. 525–539

9 Liu, Z., Chen, Y.: 'Identifying meaningful return information for xml keyword search'. Proc. Int. Conf. Management of Data (SIGMOD 2007), Beijing, China, June 2007, pp. 329–340

10 Li, G., Feng, J., Wang, J., Zhou, L.: 'Efficient keyword search for valuable LCAs over XML documents'. Proc. Int. Conf. Information and Knowledge Management (CIKM 2007), Lisbon, Portugal, November 2007, pp. 31–40

11 Yu, H., Deng, Z., Xiang, Y., Gao, N., Zhang, M., Tang, S.: 'Adaptive top-*k* algorithm in SLCA-based XML keyword search'. Proc. Int. Conf. Asia-Pacific Web (APWeb 2010), Busan, Korea, April 2010, pp. 364–366

12 Schieber, B., Vishkin, U.: 'On finding lowest common ancestors: simplification and parallelization', *SIAM J. Computing*, 1988, **17**, (6), pp. 1253–1262

13 Bao, Z., Ling, T., Chen, B., Lu, J.: 'Effective XML keyword search with relevance oriented ranking'. Proc. Int. Conf. Data Engineering (ICDE 2009), Shanghai, China, March–April 2009, pp. 517–528