

A Survey: Top-Down Xml Keyword Query Processing

1.PRAVALIKA 2. B.PRUDHVI 3. DR. K NAGESWARARAO

¹Pg Scholar, Department of CSE, Mother Teresa Institute of Science and Technology, Sathupally
pravalikapothureddy@gmail.com

²Assistant Professor , Department of CSE, Mother Teresa Institute of Science and Technology, Sathupally
prithvireddy@gmail.com

³ Professor & HOD, Department of CSE, Mother Teresa Institute of Science and Technology, Sathupally
nageswararaokapu@yahoo.com.

Abstract— Efficiently answering XML keyword queries has attracted much research effort in the last decade. The key factors resulting in the inefficiency of existing methods are the common-ancestor-repetition (CAR) and visiting-useless-nodes (VUN) problems. To address the CAR problem, we propose a generic top-down processing strategy to answer a given keyword query w.r.t. LCA/SLCA/ELCA semantics. By “top-down”, we mean that we visit all common ancestor (CA) nodes in a depth-first, left-to-right order; by “generic”, we mean that our method is independent of the query semantics. To address the VUN problem, we propose to use child nodes, rather than descendant nodes to test the satisfiability of a node v w.r.t. the given semantics. We propose two algorithms that are based on either traditional inverted lists or our newly proposed LLists to improve the overall performance. We further propose several algorithms that are based on hash search to simplify the operation of finding CA nodes from all involved LLists. The

experimental results verify the benefits of our methods according to various evaluation metrics.

1 INTRODUCTION

XML has been successfully used in many applications, such as that in scientific and business domains, as the standard format for storing, publishing and exchanging data. Compared with structured query languages, such as XPath and XQuery, keyword search is also gained popularity on XML data as it relieves users from understanding the complex query languages and the structure of the underlying data, and has received much attention due to that results are not the entire documents anymore but nested fragments. Typically, an XML document can be modeled as a node labeled tree T . For a given keyword query Q , several semantics have been proposed to define meaningful results, for which the basic semantics is Lowest Common Ancestor. Based on LCA, the most widely adopted query semantics are Exclusive LCA (ELCA) [2], and Smallest LCA (SLCA) [5], [7], [8], [9], [11]. SLCA

defines a subset of LCA nodes, of which no LCA is the ancestor of any other LCA. As a comparison, ELCA tries to capture more meaningful results, it may take some LCAs that are not SLCAs as meaningful results. Assume that for a given query $Q = \{k_1, k_2, \dots, k_m\}$, each keyword appears at least once in the given XML document. Intuitively, to get all CA nodes of Q , our method takes all nodes in the set of inverted ID Dewey label lists as leaf nodes of an XML tree T_v rooted at node v , and checks whether each node of T_v contains all keywords of Q in a “top-down” way. The “top-down” means that if T_v contains all keywords of Q , then v must be a CA node. We then remove v and get a forest $F_v = \{T_{v_1}, T_{v_2}, \dots, T_{v_n}\}$ of subtrees rooted at the n child nodes of v . Based on F_v , we further find the set of subtrees $F_{CA}(v) = F_v$, where each subtree $T_{v_i} \in F_{CA}(v)$ contains every keyword of Q at least once, i.e., node v_i is a CA node. If $F_{CA}(v) = \emptyset$, it means that for T_v , only v is a CA node, then we can safely skip all nodes of T_v from being processed; otherwise, for each subtree $T_{v_i} \in F_{CA}(v)$, we recursively compute its subtree set $F_{CA}(v_i)$ until $F_{CA}(v_i) = \emptyset$. Let $S_{child}(v)$ denote, for v , the set of child nodes that contain k_i , $S_{child}(v)$ the set of child CA nodes

of v , and $S_{CA}(v)$ the set of CA nodes in T_v . Formula 2 means that the set of CA nodes of Q equals the set of CA nodes in T_r , where r is the document root node. $S_{CA}(v)$ can be recursively computed according to Formula 3. Formula 3 means that for a given CA node v , the set of CA nodes in T_v is equal to the union of $S_{CA}(v)$ and the set of CA nodes in subtrees rooted at v 's child CA nodes, which can be further computed by Formula

2 RELATED WORKS

DIL [2] sequentially processes all involved Dewey labels in document order, its performance is linear to the number of involved Dewey labels. IS [3] sequentially processes all Dewey labels of the shortest list L_1 one by one. In each iteration, it picks from L_1 a Dewey label l and uses it to probe other lists to get a candidate ELCA node. As the basic operations of the two algorithms are OP1 and OP2, they heavily suffer from both the CAR and VUN problems. JDewey-E [7] computes ELCA results by performing set intersection operation on all lists of each tree depth from the leaf to the root. For all lists of each level, after finding the set of common nodes, it needs to recursively delete all ancestor nodes in all lists of higher levels. As a node could be a parent node of

many other CA nodes, and the deletion operation needs to process each parent-child relationship separately, JDewey-E suffers from the CAR problem. Meanwhile, as it performs set intersection on all lists of each tree depth from the leaf to the root, they will firstly visit nodes of V_2 for Q_2 , thus it also suffers from the VUN problem. As some node IDs appear in many different ID Dewey labels of the same inverted list, and HC [4] processes each ID Dewey label of the shortest list separately, it still suffers from the CAR problem. Moreover, HC needs to push each component of every ID Dewey label of the shortest inverted list into a stack, it also suffers the VUN problem when the pushed components are UNs.

3 THE BASELINE ALGORITHM

Our baseline ELCA algorithm recursively gets all CA nodes in a top-down way, then checks the satisfiability of each CA node, which works on the traditional inverted lists of labels w.r.t. Dewey or one of its variants. To do so, it needs to solve two problems: δP_1 identify the set of child CA nodes for each CA node v , δP_2 check v 's satisfiability w.r.t. ELCA semantics. For P_1 , given a query Q with m keywords, we know that $\delta_i \subseteq \{1; m\}$; $S_{\text{ca}}(v) \subseteq S_i(v)$. Thus given a CA node v and its subtree set $F_v \subseteq T_v$;

$T_{v_2}; \dots; T_{v_n}$, to get $S_{\text{ca}}(v)$, we do not need to check whether each subtree contains all query keywords; instead, we just need to check whether each node in $S_{\text{min}}(v)$, which contains least number of child nodes of v w.r.t. k_{min} , appears in $S_i(v)$. Even if we know the lengths of all child lists, it's difficult to know which one is $S_{\text{min}}(v)$. Fortunately, as all node IDs in each child list of v are sorted in ascending order, our newly proposed set intersection algorithm guarantees that the number of processed child nodes for each CA node v is bounded by $|S_{\text{min}}(v)|$. For P_2 , we use the following Lemma to check the satisfiability of v , which is similar to Lemma 1. Given a query $Q = \{k_1; k_2; \dots; k_m\}$ and CA node v ,

3.1 The Algorithm

Based on the above description, Algorithm 1 recursively gets all CA nodes in a top-down way. For each CA node v , it finds out the number of occurrences of each query keyword in its subtree, i.e., the length of each of its child list, then gets v 's child CA nodes by intersecting v 's child lists using binary search operation. After that, it checks the satisfiability of v by Lemma 1. To do so, each inverted list L_i is associated

with a cursor C_i pointing to some ID Dewey label of L_i , $C_i^{1/2}x_$ denotes the x th component of the ID Dewey label that C_i points to, and $\text{pos}(C_i)$ is used to denote C_i 's position in L_i . Given a node v , we use $l(v)$ to denote the ID Dewey label of v , $v.N_i$ denotes the number of keyword occurrences w.r.t. k_i in the subtree rooted v . As shown in Algorithm 1, it firstly initializes the subtree rooted at the root node of the given XML tree in line 1, then calls the procedure `processCANode` to recursively get all CA nodes in line 2.

Algorithm 1. TDELCA($Q = \{k_1, k_2, \dots, k_m\}$)

1 initialize v as the root, $L_i(v) = L_i$, $v.N_i = |L_i(v)|$, and C_i points to the first ID Dewey label of $L_i(v)$.

2 processCANode(v)

Procedure processCANode(v)

1 $chL \leftarrow |l(v)| + 1$

2 **while** ($\neg \text{eof}(v)$) **do**

3 $u \leftarrow \text{getNextChildCA}(v, chL)$

4 **if** ($u = -1$) **then break**

5 initializeChildCA(v, chL, u)

6 $v.[N_1, \dots, N_m] \leftarrow v.[N_1, \dots, N_m] - u.[N_1, \dots, N_m]$

7 processCANode(u)

8 **if** ($\forall i \in [1, m], v.N_i > 0$) **then**

9 output v as an ELCA node

Function getNextChildCA(v, chL)

1 $j \leftarrow 1; n \leftarrow 1; x \leftarrow \text{argmax}_i \{C_i[chL]\}$

2 **while** ($n < m$) **do**

3 **if** ($j = x$) **then** $j \leftarrow j + 1$

4 use $C_x[chL]$ as the eliminator to do binary search on the chL^{th} level of $L_j(v)$

5 **if** ($\text{pos}(C_j)$ is out of $L_j(v)$) **then return** -1

6 **if** ($C_x[chL] = C_j[chL]$) **then** $j \leftarrow j + 1; n \leftarrow n + 1$

7 **else** $x \leftarrow j; j \leftarrow 1; n \leftarrow 1$

8 **return** $C_x[chL]$

Procedure initializeChildCA(v, chL, u)

1 **for each** ($i \in [1, m]$) **do**

2 set the start position of $L_i(u)$ as $\text{pos}(C_i)$

3 binary search the end position of $L_i(u)$ by using $u + 1$ to probe the chL^{th} level of $L_i(v)$

4 $u.N_i \leftarrow |L_i(u)|$

Function eof(v)

1 **if** (exists $L_i(v)$, such that all nodes are processed) **then**

2 **return** TRUE

3 **return** FALSE

The procedure `processCANode` works as follows. It firstly gets the depth of v 's child nodes in line 1. In lines 2-7, it repeatedly gets all child CA nodes of v . For each child CA node u got in line 3, it firstly gets the values of variables associated with v in line

5; in line 6, it excludes the occurrences of all query keywords under u from that under v . In line 7, it calls `processCANode` to recursively process u . After processing all childCA nodes of v , if $\delta_i \geq \frac{1}{2}m_i$; $v.N_i > 0$, according to Lemma 1, v is an ELCA node and outputted in line

4. THE HASH SEARCH BASED ALGORITHMS

Even though TDELCA-L reduces the time complexity compared with TDELCA, it relies on the probe operation (implemented by binary search) to align the cursors of inverted lists. To further improve the overall performance, we consider the existence of additional hash indexes [4], [11], [17] on inverted lists, such that each probe operation takes time without using binary search operation. the first hash table HF records the number of nodes in each L_i , which is used to choose the shortest LList. For each L_i , another hash table H_i records, for each node of L_i , the number of its child nodes that contain k_i . Note that H_i in our methods is different with that of [4], [11], [17], where H_i records, for each node v , the number of v 's descendant nodes that directly contain k_i . we know that the number of nodes of L_1 is 11, which can be denoted as $HF \frac{1}{2}k_1 \frac{1}{4} 11$. According node 1 has three child nodes

containing k_1 ("Tom"), which can be denoted as $H_1 \frac{1}{2}l_1 \frac{1}{4} 3$. Node 5 does not have child nodes containing k_1 , thus $H_1 \frac{1}{2}5 \frac{1}{4} 0$. Similarly, node 3 does not contain k_1 , which is denoted as $3 \frac{1}{2} H_1$.

4.1 The Baseline Hash Search Algorithm

Assume that l_1, l_2, \dots, l_m , the main idea of our baseline hash search algorithm is: take the shortest LList L_1 as the working list and recursively process all CA nodes in top-down way. For each CA node v , sequentially check whether each of its child nodes in L_1 is a CA node, then output v if it is an ELCA result.

Algorithm 2. TDELCA-H($Q = \{k_1, k_2, \dots, k_m\}$)

```

1 initialize  $v$  as the root,  $L_1(v) = \mathcal{L}_1^2$ , and  $C_1$  points to the first
  node of  $L_1(v)$ 
2  $v.[N_1, N_2, \dots, N_m] \leftarrow [H_1[v], H_2[v], \dots, H_k[v]]$ 
3 processCANode( $v$ )
Procedure processCANode( $v$ )
1  $chL \leftarrow |l(v)| + 1$ ;  $N_{chCA} \leftarrow 0$ 
2 while ( $C_1 \in L_1(v)$ ) do
3   if (isCA( $C_1$ ) = TRUE) then
4      $N_{chCA} \leftarrow N_{chCA} + 1$ 
5     InitializeChildCA( $C_1, \mathcal{L}_1^{chL}$ )
6     processCANode( $C_1$ )
7     advance( $C_1$ )
8 if ( $N_{chCA} = 0$  or  $\forall i \in [1, m], v.N_i > N_{chCA}$ ) then
9   output  $v$  as an ELCA node
Function is CA( $u$ )
1 for each ( $i \in [2, m]$ ) do
2   if ( $u \notin H_i$ ) then return FALSE
3    $u.N_i \leftarrow H_i[u]$ 
4 return TRUE
Procedure InitializeChildCA ( $u, \mathcal{L}_1^{chL}$ )
1 get  $L_1(u)$  from  $\mathcal{L}_1^{chL}$  and set  $C_1$  to the first node of  $L_1(u)$ 
2  $u.N_1 \leftarrow H_1[u]$ 

```

Algorithm 2 shows the detailed description of the TDELCA-H algorithm. Compared

with TDELCA and TDELCA-L, for a given query Q, TDELCA-H only needs to process all CA nodes and their child nodes in L1. For each processed node v in L1, TDELCA-H checks whether v is a CA node by hash probe operations, rather than set intersection operations on a set of child lists.

CONCLUSIONS

Considering that the key factors resulting in the inefficiency for existing XML keyword search algorithms are the CAR and VUN problems, we proposed a generic top-down processing strategy that visits all CA nodes only once, thus avoids the CAR problem. We proved that the satisfiability of a node v w.r.t. the given semantics can be determined by v 's child nodes, based on which our methods avoid the VUN problem. Another salient feature is that our approach is independent of query semantics. We proposed two efficient algorithms that are based on either traditional inverted lists or our newly proposed LLists to improve the overall performance. Further, we proposed three hash search-based methods to reduce the time complexity. The experimental results demonstrate the performance advantages of our proposed methods over existing ones. One of our future work is studying disk-based index to facilitate XML

keyword query processing when the size of indexes becomes too large to be completely loaded into memory.

REFERENCES

- [1] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv, "XSearch: A semantic search engine for XML," in Proc. 29th Int. Conf. Very Large Data Bases, 2003, pp. 45–56.
- [2] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, "Xrank: Ranked keyword search over XML documents," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2003, pp. 16–27.
- [3] Y. Xu and Y. Papakonstantinou, "Efficient LCA based keyword search in XML data," in Proc. 11th Int. Conf. Extending Database Techn.: Adv. Database Technol., 2008, pp. 535–546.
- [4] R. Zhou, C. Liu, and J. Li, "Fast ELCA computation for keyword queries on XML data," in Proc. 13th Int. Conf. Extending Database Technol., 2010, pp. 549–560.
- [5] Y. Xu and Y. Papakonstantinou, "Efficient keyword search for smallest LCAS in XML databases," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2005, pp. 537–538.
- [6] Y. Li, C. Yu, and H. V. Jagadish, "Schema-free xquery," in Proc. 13th Int.

Conf. Very Large Data Bases, 2004, pp. 72–83.

[7] L. J. Chen and Y. Papakonstantinou, “Supporting top-K keyword search in XML databases,” in Proc. 26th Int. Conf. Data Eng., 2010, pp. 689–700.

[8] C. Sun, C. Y. Chan, and A. K. Goenka, “Multiway SLCA-based keyword search in XML data,” in Proc. 16th Int. Conf. World Wide Web, 2007, pp. 1043–1052.

[9] Z. Liu and Y. Chen, “Reasoning and identifying relevant matches for XML keyword search,” J. Proc. Very Large Data Bases Endowment, vol. 1, no. 1, pp. 921–932, 2008.

[10] G. Li, J. Feng, J. Wang, and L. Zhou, “Effective keyword search for valuable LCAS over XML documents,” in Proc. 16th ACM Conf. Inform. Knowl. Manage., 2007, pp. 31–40.

[11] W. Wang, X. Wang, and A. Zhou, “Hash-search: An efficient SLCA-based keyword search algorithm on XML documents,” in Proc. 14th Int. Conf. Database Syst. Adv. Appl., 2009, pp. 496–510.

[12] Y. Chen, W. Wang, and Z. Liu, “Keyword-based search and exploration on databases,” in Proc. IEEE 27th Int. Conf. Data Eng., 2011, pp. 1380–1383.

[13] B. Q. Truong, S. S. Bhowmick, C. E. Dyreson, and A. Sun, “MESSIAH: Missing element-conscious SLCA nodes search in XML data,” in Proc. SIGMOD, 2013, pp. 37–48.

[14] L. Kong, R. Gilleron, and A. Lemay, “Retrieving meaningful relaxed tightest fragments for XML keyword search,” in Proc. 12th Int. Conf. Extending Database Technol.: Adv. Database Technol., 20 pp. 815–826.

[15] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava, “Keyword proximity search in XML trees,” IEEE Trans. Knowl. Data Eng., vol. 18, no. 4, pp. 525–539, 2006.

[16] J. Zhou, Z. Bao, W. Wang, T. W. Ling, Z. Chen, X. Lin, and J. Guo, “Fast SLCA and ELCA computation for XML keyword queries based on set intersection,” in Proc. 28th Int. Conf. Data Eng., 2012, pp. 905–916.

[17] J. Zhou, Z. Bao, W. Wang, J. Zhao, and X. Meng, “Efficient query processing for XML keyword queries based on the idlist index,” Int. J. Very Large Data Bases, vol. 23, no. 1, pp. 25–50, 2014.