An algorithm for generating MCCTree from XML document

UMMU SULAIM SAZALY
MOHD HASAN SELAMAT, SALMI BAHAROM, MARZANAH AB. JABAR
Department of Information System
Universiti Putra Malaysia
43400 Serdang, Selangor
MALAYSIA
ummusulaim2311@gmail.com
{hasan, salmi, marzanah}@fsktm.upm.edu.my

Abstract: - Keyword query is an important research in information retrieval. Although many approaches have been proposed, searching for efficient and accurate approach to answer keyword query still continue. This paper presents a new algorithm to select and generate MCCTree from XML document called XMCCTree. This algorithm uses CLCA and MCLCA notions and considered as an enhancement of the previous CGTreeGenerator and MCCTreeGenerator algorithms. The objective of this algorithm’s development is to improve the efficiency on generating MCCTree to answer keyword query in XML document.

Key-Words: - XMCCTree, algorithm, MCLCA, CLCA, MCCTree, XML, keyword query.

1 Introduction
Since XML has been accepted in almost all applications to stores data, variety of notions and algorithms have been introduced. Each of them were develop with certain requirement that restrict to some application of interest [1]. When the XML is presented as a tree, the notion of Lowest Common Ancestor (LCA) [2] is very suitable to be used in selecting possible answer for keyword query. Many researchers also provide algorithm to implement their LCA notions. The XRank [3] used stack in its RDIL Query Processing algorithm. The enhance LCA by [4] and [5] used collection or list to implement their algorithms. Then, stack is reused in implementing notions of Compact LCA (CLCA) and Maximal CLCA (MCLCA) [6] in its proposed algorithm. At the end, the output of the algorithm must satisfy the notion that they introduced before.

The keyword query processing finds the nodes that contain keyword(s) in XML tree. Then, it will be sorted to find the LCA depending on the location of the nodes. Therefore, most algorithms in LCA research will use Dewey indexing method in their approaches. It can help in sorting the nodes since Dewey indexing method indexed the nodes based on its structure.

Algorithms proposed in [6] implement notions of CLCA and MCLCA. It generates a Compact Global Tree (CGTree) and selects a set of compact Maximal Compact Connected Tree (MCCTree) to answer keyword query. Besides the notions to process the query, the research also provides a ranking method. Unfortunately, the set of MCCTree produced by both algorithms are not suitable for its ranking method. Output from these algorithms cannot be used directly as an input for the ranking method.

In this paper, a new algorithm to generate a set of MCCTree is presented, called XMCCTree (eXtended MCCTree). Theoretically, the proposed algorithm is more efficient compared to the previous algorithms, the CGTreeGenerator and the MCCTreeGenerator. The XMCCTree is an enhancement of the previous algorithms which produce a set of MCCTree in a form that can be used directly as input for the ranking method. Section 2 explains the concepts, steps and pseudocode of the XMCCTree. Section 3 presents comparisons between XMCCTree and CGTreeGenerator+MCCTreeGenerator with some examples. Finally, conclusions were presented in Section 4.

2 The XMCCTree Algorithm

2.1 The concept of XMCCTree
The main purpose of XMCCTree is to generate the MCCTree in incompact structure. XMCCTree uses concept of set or collection. The 3 steps that can describe XMCCTree are defined, develop and select. XMCCTree starts with defining a subtree that contains keywords in it. Once determined, XMCCTree will generate the subtree until it reaches the deepest node that contains the keyword. When the subtree has been generated, the subtree
XMCCTree uses XML tree, a set of keyword nodes, \(K\), sorted by index, and a variable named \(CMSet\), which is declared as a node root. This root node will then be assigned to variable \(T\). The defining steps define whether \(T\) have child by reading set \(K\) and detect the child, \(c_i\), which have starting index with root \(T\). Then, \(c_i\), is added as a child for \(T\).

Before developing the subtree of each \(c_i\) under \(T\), XMCCTree will get a number of keyword(s) in each \(c_i\), \(CK\), and a number of keyword(s) in \(T\), \(CKT\). Then, \(c_i\) will be sorted in descending order of the number of keyword(s) contain in its subtree. If the \(CKT\) is equal with \(CK\), the \(c_i\) will be eliminated from \(T\) and added into \(CMSet\). Otherwise, the \(T\) is possibly a MCCTree. To generate the complete tree of \(T\), XMCCTree will generate child of \(T\) by searching index of keyword \(K\), which has starting index at the root \(T\). The variable node will follow the generated child until it reach the deepest keyword that contain node in the branch. If \(K\) still has nodes under \(T\), node will travel back through the branch and continue generating another branch to the deepest keyword nodes.

The last step is to select the MCCTree. If \(T\) has only a subtree, the subtree of \(T\) is selected as MCCTree. Otherwise, the tree \(T\) itself will be MCCTree. All the steps continued until \(CMSet\) is empty. XMCCTree will return a set of MCCTree to be used in ranking method proposed by [6] to answer XML keyword query.

2.3 The algorithm of XMCCTree
The XMCCTree is part of processes in keyword query processing. Before the XMCCTree can be executed, the XML document must be parsed and indexed using Dewey indexing method. After parsing the document, the XML data must be transformed into an XML tree. All these procedures are independent, depends on the programmer’s choice on how to implement it. The XMCCTree requires an input of indexed XML tree and set of indexes of keyword that contain nodes sorted in ascending order. The algorithm of the XMCCTree is as follows:

1 Begin
2
3 XMCCTreeSet \(\leftarrow\) null;
4 CMSet = 0; //node root
5 while CMSet is not empty do
6 \{ 
7 \(r = \text{root}(T \in \text{CMSet})\);
8 \(CMSet = CMSet - \{T\}\);
9 \(c_i = \text{read_keyword}()\);
10 foreach \(c_i\) 
11 \{ add \(c_i\) as child for \(T\)
12 \(/\text{get } |CK(c_i)|()\);
13 foreach \(c_i/C\text{ children}()//\text{in descending order by }|CK(c_i)|\) 
14 \{ \(t_i = c_i;\)
15 if \(|CK(T)| = |CK(t_i)|\) then
16 \(T = T - t_i;\)
17 \(CMSet = CMSet \bigcup \{t_i\};\)
18 else
19 break;
20 \}
21 \}
22 if \(|CK(T)| > 0\) then
23 foreach \(c_i\) of \(T\)
24 \{ \(K_{c_i} = \text{getKeyword}(c_i); //\text{keywords with } c_i \text{ as ancestor}\)
25 node = \(c_i;\)
26 while(node == ancestor of \(K_{c_i}\).
27 \text{first}() \&\& \(K_{c_i}\).
28 \text{first}() != \text{node})
29 \{ tc_i = tc_i \cup \text{child}; //add child to subtree rooted at \(c_i\)
30 node = \text{child;}
31 if (node == \(K_{c_i}\).
32 \text{first}())
33 \(K_{c_i}\).
34 \text{pop}();
35 if \(K_{c_i}\).
36 \text{eol}() 
37 break;
38 else
39 while (node != ancestor of \(K_{c_i}\).
40 \text{first}())
41 \{ node = node.\text{parent}; \}
42 if \(T\) has only 1 subtree
43 XMCCTreeSet = XMCCTreeSet \bigcup \{t_i\};
44 else
45 XMCCTreeSet = XMCCTreeSet \bigcup \{T\};
46 \}
47 \}
48 \}
49 \} 
50 return XMCCTreeSet;
51 end

The following example illustrates the execution of the XMCCTree algorithm. The query has 3 keywords, \(k_1\), \(k_2\) and \(k_3\). Assume that we have an XML tree and a set of keyword contain nodes, \(K = \{00000, 000100, 000101, 000111, 00020, 000210, 001, 010, 011, 021\}\), as shown in Fig. 1. The algorithm starts from the node root with index [0].

```
root = [0] //CMSet = [49]
XMCCTree detects number of child for [0] from K = [00000, 000100, 000101, 000111, 00020, 000210, 001, 010, 011, 021].
```
Then, the child [00], [01], [02] is added under [0]. For each child, XMCCTree detects the number of keywords that they have and sorts it in descending order. Based on the XML tree, the [00] have 3 keywords, [01] have 2 keywords, and [02] only have 1 keyword. The sorted list will be [00], [01], and [02]. Then, comparison between child and root will be done to select possible MCCTree. When the number of keywords contains under root is equal with the number of keyword contains under child, the child will be eliminated from the tree and added into CMSet. The graphical views of these steps are shown in Fig. 2.

Fig. 2 Comparison between child and root

When next child, [01] has number of keywords less than root (refer Fig. 2), tree rooted at root is possibly a MCCTree. Next, XMCCTree will generate this tree.

Nodes that contain keyword under ci is listed, $K_{ci} \rightarrow [01] = \{010, 011\}$. Index of $ci$ is compared with the first value of $K_{ci}$. XMCCTree generates nodes by node until the index of generated node is equal to the first value in $K_{ci}$. Then, $K_{ci}$ eliminate that value and start the comparison and generating nodes again until the same index in $K_{ci}$ is reached. If the next index of $K_{ci}$ does not have the same ancestor with current generated tree (means that the nodes must not be in the same branch), the pointer will reverse through the ancestor until reach a node with the same ancestor with value in $K_{ci}$. The steps can be visualized as in Fig. 3 as below:

Keyword under [01] = \{010, 011\}

Fig. 3 illustrates part of steps in generating a tree

When all the nodes under all children have been completely generated, the tree is added into a list of MCCTree. If the tree has only a branch of child, then the subtree rooted at the child will be selected as MCCTree. Otherwise, the tree rooted at root is returned as MCCTree.

### 3 XMCCTree vs Previous Algorithm

As an enhancement of the previous algorithms, XMCCTree maintains the notions that have been used before. However, in this paper we used the term CGTreeMCCTreeGen to indicate a combination of CGTreeGenerator and MCCTreeGenerator. XMCCTree produces the same result of MCCTree as CGTreeMCCTreeGen but differ in terms of the structure of the MCCTree.

CGTreeGenerator transforms XML tree into a compact global tree by eliminating linked nodes (nodes with only one subtree). Therefore, the structure of the compact global tree is in the compact structure. The process begins from the deepest nodes on the left in XML tree; which have the lowest Dewey index number. It is then inserted into a stack. The algorithm computes the LCA through travelling the nodes and the stack. It maintains a set of nodes to construct the compact global tree.

The compact global tree is then used as input for the MCCTreeGenerator. MCCTreeGenerator will select MCCTree in a top-down manner. Starting from root, MCCTreeGenerator will define numbers of each keyword contains in each child. Child which has same keywords and numbers of keyword with root will be eliminated from the tree to be another potential MCCTree. The remaining nodes in the tree
will be a potential MCCTree. Then, MCCTreeGenerator will travel through the subtree of each child to identify the structure of the tree. After that, it will return the tree as MCCTree. Note that the structure of the returned MCCTree is in a compact structure because it is define from a compact global tree.

XMCCTree can be defined as an enhanced MCCTreeGenerator. XMCCTree start generating tree from root with index [0], and identify its child from the keyword list. Then, XMCCTree add the identified child and compute numbers of each keyword in each child. Similar to MCCTreeGenerator, a child that contains the same numbers of keyword is eliminated from the tree as a potential MCCTree. After that, XMCCTree will start generating the descendant of each child until it reaches the keyword node.

Both XMCCTree and CGTreeMCCTreeGen algorithms use the list of keywords node index as a limit in constructing the tree. It is used to avoid them travelling the nodes and paths that are not related to the keyword. The differences between both algorithms are:

- The XMCCTree is faster than CGTreeMCCTreeGen by avoiding the generating of compact global tree to find the lowest LCA.
- The CGTreeMCCTreeGen produce a set of MCCTree in a compact structure, while XMCCTree produce a set of MCCTree in an incompact structure.
- The XMCCTree is more efficient since the result MCCTree can be used directly in the ranking method proposed by Feng et al. [6]

Comparison between the output of XML keyword query using XMCCTree and CGTreeMCCTreeGen is shown in Fig. 4.

The time complexity for CGTreeMCCTreeGen is $O(d \log m \sum_{i=1}^{m} | \tau |) + O(m \sum_{i=1}^{m} | \tau |)$ where $m$ is the number of keywords and $d$ is a depth of the XML document. Since XMCCTree is similar to MCCTreeGenerator, the complexity is also similar. In XMCCTree, each node $n$ needs to sort its children with respect to number of keywords contains in it, $m$, and for each child, algorithm needs to generate descendant into some level, $d$, until it reach the keyword nodes. Hence, the total complexity is $O(m \sum_{i=1}^{m} | \tau | * d)$.

4 Conclusion

In this paper, a new algorithm to select MCCTree from XML document is presented called XMCCTree. The XMCCTree is an enhancement of CGTreeGenerator and MCCTreeGenerator.
algorithms proposed in [6]. The algorithm enhanced the way to select and produce MCCTree in a way that can be use directly in ranking method. Previous algorithms produce MCCTree in a compact structure but \textit{XMCCTree} produce MCCTree in incompact structure. \textit{XMCCTree} maintain the notions and ranking methods used to answer XML keyword query but modify its algorithm to enhance the query process.

In a worst case, previous algorithms use $O(d \log m \sum_{i=1}^{m} | T_i |) + O(m \sum_{i=1}^{m} | T_i |)$, while \textit{XMCCTree} takes $O(m \sum_{i=1}^{m} | T_i | \ast d)$.

We plan to develop a prototype of both algorithms and run an experiment to prove that \textit{XMCCTree} is more efficient than the previous algorithm in selecting MCCTree from XML document.

\textbf{Acknowledgement:}

This research is supported by Fundamental Research Grant Scheme (FRGS), Research Fund Number: 03-04-10-867FR, Ministry of Higher Education, MALAYSIA.

\textbf{References:}


