Locating Valid SLCAs for XML Keyword Search with NOT Semantics

Rung-Ren Lin\textsuperscript{1}, Ya-Hui Chang\textsuperscript{2}; and Kun-Mao Chao\textsuperscript{1}

\textsuperscript{1}\textit{Department of Computer Science and Information Engineering}
National Taiwan University, Taipei, Taiwan.
\{r91054, kmchao\}@csie.ntu.edu.tw

\textsuperscript{2}\textit{Department of Computer Science and Engineering}
National Taiwan Ocean University, Keelung, Taiwan.
yahui@ntou.edu.tw

\textbf{ABSTRACT}

Keyword search provides an easy way for users to pose queries against XML documents, and it is important to support queries with arbitrary combinations of AND, OR, and NOT operators. The previous RELMN algorithm processed such kind of queries by extending the original SLCA definition in a straightforward way, but it did not work correctly in some cases. In this paper, we propose the concept of \textit{valid SLCAs} as query results. Basically, nodes in an XML document are classified according to their usages, which is further used to define the \textit{scope} affected by a negative keyword. Only \textit{valid} nodes, which are not affected by any negative keyword, are qualified to identify \textit{valid SLCAs}. The experimental results show that the proposed algorithm achieves higher precision and recall, and is more efficient than the previous work.

\section{INTRODUCTION}

Keyword search provides an easy way for users to obtain desired information from XML documents by just giving some keywords they are interested in, but irrelevant data may be returned due to lacking exact query semantics. Many researchers advocate the idea that each returning result should be based on an LCA (Lowest Common Ancestor), which contains every keyword under its subtree at least once \cite{1}\cite{3}\cite{4}\cite{5}\cite{6}\cite{7}. Among all the LCA-based techniques, the smallest-LCA (SLCA) \cite{6} concept is particularly popular since it can locate nodes semantically closer to the query keywords. Specifically, a node $n$ is said to be an SLCA node if: (i) $n$ is an LCA node, and (ii) none of $n$'s children are LCA nodes. Consider the sample XML tree depicted in Figure 1, which represents the information about course offerings at a particular school, and each node is identified by a Dewey encoding.

\begin{table}[h]
\centering
\caption{Sample keyword queries.}
\begin{tabular}{|c|c|}
\hline
Query & Description \\
\hline
$Q_1$ & Subject \& Friday \\
$Q_2$ & Subject \& Friday \& \textasciitilde R101 \\
$Q_3$ & 2010 \& Subject \& \textasciitilde R101 \\
$Q_4$ & Red Wood \& Subject \& Friday \& \textasciitilde R103 \\
$Q_5$ & Subject \& Friday \& \textasciitilde R102 \& 2010 \\
$Q_6$ & Subject \& Friday \& (R101 \coppypaste=true \& R103) \\
\hline
\end{tabular}
\end{table}

Query $Q_1$ in Table 1 is used to retrieve those subjects whose classes are held on Friday. Since only nodes 1.2.2 and 1.3.3 contain Subject and Friday under their subtrees and none of their children do, these two nodes are the SLCAs for $Q_1$.

The limitation of the works above is that they only concern the keywords with the AND operator. It is therefore an important research issue to process queries with arbitrary combinations of AND, OR, and NOT operators, and the challenge mainly lies in how to identify results satisfying the NOT semantics. A straightforward idea is that the returning SLCA should contain every “positive” keyword but no “negative” keyword (keywords with NOT operators). Consider the sample query $Q_2$ in Table 1, where the exclamation mark “!” is used to represent the NOT operator. The semantics of query $Q_2$ is similar to that of query $Q_1$, except that $Q_2$ should not output the courses held in room 101. Since the course rooted at node 1.2.2 contains the unwanted negative keyword “R101”, only the subtree rooted at 1.3.3 should be returned. Although intuitive, such approach is sometimes too restrictive. Consider query $Q_3$, asking for subjects whose classes are offered in 2010 but not held in room 101. We cannot find an SLCA satisfying the above definition, but actually node 1.2.3.1, whose class is held in room 103, is an answer reasonable for a human user, which we call \textit{human answers} in short. To avoid the occurrences of such false negatives, when
no satisfied nodes are found, our previously proposed RELMN algorithm [2] will identify SLCAs only based on positive keywords, and then prune out nodes according to negative keywords.

Although the RELMN algorithm basically performs well, the process of result identification is complex, and it still might cause false negatives or false positives in some cases, as will be discussed in detail in Section 2. Therefore, we propose a new approach to improve the effectiveness and efficiency of processing XML keyword search with NOT semantics. The main idea is to classify all nodes into several groups based on their usage. This classification is used to define the scope that a negative keyword can affect. We then propose to return valid SLCA as the query result, where a valid SLCA contains every positive keyword under its subtree, and each of which is not affected by any negative keyword. Consider query \( Q_3 \) again. Node 1.2 is the only SLCA based on positive keywords. Since its descendant nodes 1.2.1.1 and 1.2.3.1 match the positive keywords, and are not within the scope affected by the negative keyword \( R101 \), node 1.2 is qualified as a valid SLCA, which contains human answers and should be returned. In summary, the contributions of this paper are as follows:

- Based on a simple observation of XML documents, we propose a set of definitions which identify nodes which should not be output due to negative keywords.
- We design an efficient algorithm called ValidSLCA to process queries with arbitrary combinations of AND, OR, and NOT operators.
- We perform an empirical evaluation by measuring the precision, recall, and processing time of two approaches. The experimental results show that the new ValidSLCA algorithm is more efficient and gives better query results than the previous RELMN algorithm [2].

The rest of this paper is organized as follows:

Section 2 briefly describes the RELMN algorithm. The basic observation and definitions underlying the new approach are presented in Section 3, and the corresponding algorithm is discussed in Section 4. At last, experimental studies and conclusions are given in Section 5 and Section 6, respectively.

2. THE RELMN APPROACH

We briefly describe the RELMN algorithm in this section, and direct the readers to [2] for detailed discussion.

The RELMN algorithm basically consists of two steps. First, it searches the satisfied SLCAs of the input query. Specifically, a node \( n \) satisfies the query \( Q \) if \( n \) contains every “positive” keyword under its subtree but does not contain any “negative” keyword. Further, \( n \) is a satisfied SLCA if \( n \) satisfies \( Q \) but none of \( n \)’s children do. As discussed in Section 1, node 1.3.3 is a satisfied SLCA for query \( Q_2 \). When no satisfied SLCAs are found, the RELMN algorithm will locate SLCAs purely based on positive keywords. Second, for every subtree rooted at the identified SLCAs, it then prunes the unwanted part based on the following rule: node \( n_1 \) is able to prune its sibling node \( n_2 \) if \( n_1 \) contains more positive keywords or contains less negative keywords under its subtree than \( n_2 \) does. Consider query \( Q_3 \) again. RELMN will first identify node 1.2 based on the positive keywords “2010” and “Subject”, and then exclude the subtree rooted at node 1.2.2, since it has the negative keyword \( R101 \) and is pruned by node 1.2.3.

The time complexity of the above two steps are respectively \( O(d|M| \cdot w) \) and \( O(\sum_{i=1}^{z} b_i) \), where \( d \) is the depth of the XML tree, \( |M| \) is the sum of the keyword frequencies, \( w \) is the number of clauses of query \( Q \) in conjunctive normal form, \( z \) is the number of nodes in the match tree for \( Q \), and \( b_i \) is a match (in RELMN) is a node containing a positive/negative keyword. The match tree for a given query is a subtree of the XML tree, which consists of the nodes along the path from each match up to the root.
the number of siblings of the $i^{th}$ node.

We now discuss the cases where the RELMN algorithm might cause false negatives or false positives. Consider query $Q_4$ in Table 1, asking for the course information at the school “Red Wood”, and only the subjects whose classes are held on Friday but not in room 103 should be returned. Note that nodes 1.2.2 and 1.3.3 represent human answers. Nevertheless, RELMN will only return 1.3.3. The reason is that RELMN will first identify node 1 as the only SLCA, and then prune node 1.2 by node 1.3 because it contains the negative keyword $R103$. Hence, node 1.2.2 is excluded too. Take query $Q_5$ as another example, asking for subjects whose classes are held on Friday and in room 101 or 103. The human answer is empty. However, RELMN will return both nodes 1.2.2 and 1.3.3, since they are both identified as SLCAs based on positive keywords and cannot prune each other.

3. BASIC OBSERVATIONS AND DEFINITIONS

In this section, we explain the observation on XML documents which is underlying the new approach, and deliver the basic definitions. The main idea is to identify the scope which is affected by a negative keyword according to node types. Since the entity-relationship (ER) model has been widely used for data modeling, we borrow some terms from that model to classify the nodes in an XML tree based on their usage as follows:

1. A node is a text node if it is a value.
2. A node is an attribute node if it has only one child, which is a text node.
3. A node denotes an entity if its tag name is identical to its siblings, such as a *-node in the XML DTD (Document Type Definition).
4. A node is a dummy node if it is none of the three nodes mentioned above.

For example, nodes 1.1.1 (Red Wood), 1.2.3.2.1 (Monday), and 1.3.1.1 (2011) are text nodes. Nodes 1.2.1 (Year) and 1.2.2.3 (Room) are attribute nodes. Nodes 1.2 (Courses) and 1.2.3 (Course) are entity nodes. In addition, the closest entity of node $n$ refers to the lowest entity node that are the ancestor of $n$. For instance, node 1.3 (Courses) is the closest entity of node 1.3.1.1 (2011), and node 1.2.3 (Course) is the closest entity of node 1.2.3.2.1 (Monday).

We observe that a text node is generally used as the attribute value of its closest entity. For example, node 1.1.1 (Red Wood) is used to describe the name of the school (node 1), node 1.3.1.1 (2011) indicates the courses (node 1.3) belonging to year 2010, and node 1.2.3.2.1 (Monday) represents that the course (node 1.2.3) is held on Monday. Therefore, if a negative keyword matches a text node $n$, it is reasonable to assume that the subtree rooted at $n$’s closest entity is unwanted. For instance, if the negative keyword is $R101$, we assume that the subtrees rooted at nodes 1.2.2 and 1.3.2 should not be returned. In other words, we propose that the query result is composed of the subtrees which represent positive keywords and are not “affected” by the negative keywords. For ease of discussion, we assume that the query is in DNF in the rest of this section. Hence, the clauses in a query are connected with the OR operators, and keywords in the same clause are connected with the AND operators. The formal definitions are then given as follows:

Definition 1. If node $n$ is a text node and corresponds to a given negative keyword, the closest entity of $n$ is termed a negator.

Definition 2. A node is a match of keyword $k$ if its tag name or content contains $k$. Moreover, a match is valid if it has no ancestors that are negators formed within the same clause.

Consider the positive keyword Friday in query $Q_2$. Among the two matches 1.2.2.2.1 and 1.3.3.2.1, the former one will become invalid due to the negator (node 1.2.2) caused by the negative keyword $R101$.

Definition 3. For a given clause $c$, a node $n$ is said to be an SLCA if $n$ contains every positive keyword of $c$ at least once under its subtree and none of $n$’s children do. Furthermore, $n$ is a candidate valid SLCA (VSLCA) if $n$ contains at least one valid match for every positive keyword of $c$.

Continuing the running example, both nodes 1.2.2 and 1.3.3 are SLCAs in our framework, but only node 1.3.3 is a candidate VSLCA since node 1.2.2 has no valid matches for both Friday and Subject under its subtree.

Definition 4. Given a keyword query in DNF with only one clause, all the candidate VSLCAs $n$ are valid SLCAs. If there is more than one clause, $n$ is a valid SLCA only if $n$ has no descendants that are also candidate VSLCAs formed by other clauses.

The proposed new approach will return all identified valid SLCAs as the query result. Consider query $Q_6$, which searches the subjects whose classes are held on Friday and in room 101 or 103. The
DNF of $Q_4$ has two clauses: $(\text{Subject} \land \text{Friday} \land R101)$ and $(\text{Subject} \land \text{Friday} \land R103)$, and the candidate VSLCAs of these two clauses are node 1.2.2 and node 1.2, respectively. The query result will be node 1.2.2, since it is the descendant of node 1.2.

Note that in Definition 1, we only consider text nodes for negative keywords. The reason is that a negative non-text keyword is mainly used to describe unnecessary information, and does not affect the validity of an SLCA. For instance, suppose the negative non-text matches have no ancestor-descendant relationships with other positive keywords, such as in the query $(\text{Subject} \land R101 \land \lnot \text{Days})$. We can see that the SLCA node, 1.2.2, is a human answer and should not be affected by $\lnot \text{Days}$. Even if the negative non-text matches have ancestor-descendant relationships with other positive keywords, such as in the query $(\text{Subject} \land R101 \land \lnot \text{Room})$, node 1.2.2 (Course) should still be returned.

### 4. THE VALIDSLCA ALGORITHM

The proposed ValidSLCA algorithm is listed in Figure 2. Briefly, for the input query $Q$, we will convert it into DNF $Q'$ (line 1). To get the candidate VSLCAs of each clause, we first identify the SLCAs based on only positive keywords by the approach given in [6] (line 5). Then we collect the negators in line 6. The negators are used to determine the valid matches by procedure ValidateMatches. Namely, some of the matches will be removed if they are the descendants of the negators. Next, we identify candidate VSLCAs by ensuring that there exists at least one valid match for each positive keyword (lines 9-12). At last, those candidate VSLCAs that are ancestors of others are removed (line 13).

Observe that the procedures ValidateMatches and CombineSLCA are variants of merge sort algorithms and are therefore linear to the input. Besides, two B-tree indices are maintained for efficient processing. In the first index, the data associated with each keyword $k$ is a sorted list of Dewey numbers of all the nodes which contain keyword $k$. If a node is a text-node, we additionally record its closest entity. Hence, we can efficiently retrieve the match array of the keyword (lines 3-4) and negators (line 6). The second index is used to output meaningful answers for users (line 14), which retrieves the tag name by a given Dewey number. Both indices are created and accessed based on the Oracle Berkeley DB.

We next give the time complexity of ValidSLCA. The details are omitted due to space constraints. Let $d$ be the maximum depth of the XML tree and

$$\text{http://www.oracle.com/database/}$$

**Input:** A keyword query $Q$ of arbitrary combination with AND, OR, and NOT operators.

**Output:** All of the valid SLCAs of $Q$.

**ValidSLCA($Q$)**

```
1 convert $Q$ into DNF $Q'$
2 for each clause $c_i$ of $Q'$ do
3     for each positive keyword $k_i$ of $c_i$ do
4         $M_i$ ← all the matches of $k_i$
5     $S_i$ ← compute SLCAs by $\{M_1, M_2, M_3, \ldots\}$
6     $E$ ← collect the negators by the negative keywords of $c_i$
7     for each $M_i$ do
8         $M_i' ← \text{ValidateMatches}(M_i, E)$
9     for each $n \in S_i$ do
10        for each $M_i'$ do
11           if $n$ has no descendants belongs to $M_i'$ then
12              remove $n$ from $S_i$
13      $S ← \text{CombineSLCA}(S_1, S_2, S_3, \ldots)$
14 Output $S$ /* all the valid SLCAs of query $Q$ */
```

**ValidateMatches($M, E$)**

```
1 $i ← 1, j ← 1$
2 while $i \leq |M|$ and $j \leq |E|$ do
3     if $M[i]$ precedes $E[j]$ then
4         $i ← i + 1 /* M[i]$ is valid */
5     else if $E[j]$ is the ancestor of $M[i]$ then
6         remove $M[i]$ from $M$ /* $M[i]$ is invalid */
7         $i ← i + 1$
8     else
9         $j ← j + 1$
10 return $M$
```

**CombineSLCA($S_1, S_2, S_3, \ldots$)**

```
1 $S ←$ merge $(S_1, S_2, S_3, \ldots)$ by their Dewey Numbers
2 $i ← 1, j ← 2$
3 while $j \leq |S|$ do
4     if $S[i]$ is the ancestor of $S[j]$ then
5         remove $S[j]$ from $S$
6     else
7         $i ← j$
8         $j ← j + 1$
9 return $S$
```

**Figure 2:** The ValidSLCA algorithm.

$Q'$ be the DNF of the original input query $Q$. Suppose $Q'$ has $k$ clauses, and the $i^{th}$ clause has $p_i$ positive keywords and $n_i$ negative keywords. That is, the matches of positive keywords of $c_i$ are $M_1, M_2, \ldots, M_{p_i}$, and the matches of negative keywords of $c_i$ are $N_1, N_2, \ldots, N_{n_i}$. Let $|M^\text{sum}_i| = \sum_{j=1}^{p_i} M_j$ and $|N^\text{sum}_i| = \sum_{j=1}^{n_i} N_j$ be the sum of the size of positive and negative matches. The overall time complexity of algorithm ValidSLCA is $O(\sum_{i=1}^{k} |M_j| + |N^\text{sum}_i| \cdot d)$.

This section is concluded by showing how Algorithm ValidSLCA can correctly process queries $Q_4$ and $Q_5$. Recall that nodes 1.2.2 and 1.3.3 are two
We used four data sets, DBLP.xml\textsuperscript{2}, Mondial.xml\textsuperscript{2}, Reed.xml\textsuperscript{3}, and Baseball.xml\textsuperscript{4}, to perform the experiments. Some statistics about these data sets, including their sizes, are listed in Table 2 for comparison. Besides, the metrics to compare the two algorithms are precision, recall, and processing time. The first two are calculated based on human answers, which are obtained from three human experts with differences resolved by voting.

We first use the Baseball and Reed data sets to analyze the precision and recall of RELMN and ValidSLCA. Figure 3 shows the portions of these two data sets to assist in analyzing the outcomes of experiments. The test queries of these two data sets are listed in Table 3, and the precision and recall on the Baseball data set is displayed in Figure 4. We can see that ValidSLCA gets better precision than RELMN in $Q_{B2}$. In RELMN algorithm, all the nodes with “surname” tag names satisfy $Q_{B2}$ and will be returned. Since some of the players are starting pitchers or relief pitchers, it gets imperfect precision and 100% recall. However, those players who are starting pitchers or relief pitchers are considered as invalid matches in ValidSLCA. Hence, ValidSLCA gets 100% precision and recall in $Q_{B2}$. RELMN also gets imperfect precision in $Q_{B4}$ due to the same reason. On the other hand, the precision and recall on the Reed data set is displayed in Figure 5. The semantics of $Q_{R6}$ is to search the room of course “Love in Shakespeare” which does not start at 02:40PM. However, there only exists one course titled “Love in Shakespeare”, and it coincidentally starts at 02:40PM. That is, the answer should be empty. Refer to Figure 3 (b). The closest entity of 02:40PM is the node with tag name course. Hence, ValidSLCA returns empty while RELMN still returns course “Love in Shakespeare”. The similar situation happens in $Q_{R7}$, which makes RELMN get poor precision and recall, too.

5. EXPERIMENTAL EVALUATION

In this section, we compare the performance of RELMN and ValidSLCA algorithms. Both of them were implemented in C++, and the experiments were performed on a 1.67GHz dual-core CPU with 1.5GB RAM. Note that in order to be fair with the RELMN approach, after identifying valid SLCAs, we further adopt the pruning rules described in [5] to remove subtrees containing less positive keywords than their siblings. The time complexity of ValidSLCA therefore becomes $O(S_{i=1}^{k}p_{j=1}^{n}|M_{j}|+|N_{i}^{sum}|\cdot d) + O(S_{i=1}^{k}|M_{i}^{sum}|\cdot d \cdot 2^{p_{i}})$.
Due to space limitation, we do not explicitly list the test queries. As shown in Figure 6, we can see that our new approach is more efficient than the previous work in most cases. One reason is that the pruning rules defined in RELMN are very complex to process. Besides, as shown in the time complexity of the two algorithms, RELMN is affected by the branch factor of the XML trees ($b_i$), which may be very large. On the other hand, ValidSLCA is affected by the number of the query keywords ($p_i$), which is comparably small. Hence, the RELMN algorithm often takes more time than ValidSLCA does.

6. CONCLUSIONS

In this paper, we propose an algorithm to process keyword search with NOT semantics. The idea is to classify matches as valid or invalid based on negative keywords, and only valid nodes are qualified to identify valid SLCAs as query outputs. We empirically evaluate the precision, recall, and processing time by comparing the previous work and our new approach. The experiments show that our new approach gives more reasonable query results and is even more efficient. As part of our future work, we are interested in designing a novel ranking scheme to order the query results so that users can focus on most desirable ones.

7. REFERENCES