Efficient SLCA-Based Keyword Search on XML Databases: An Iterative-Skip Approach

Jiaqi Zhang, Zhenying He, Yue Tao, Xiansheng Wang, Hao Hu, and Wei Wang

School of Computer Science, Fudan University, Shanghai China
{09210240048,zhenying,09210240035,072021106,
06300720212,weiwang1}@fudan.edu.cn

Abstract. Keyword search is a popular way to discover information from XML data. To return meaningful results, SLCA (smallest lowest common ancestor) has been proposed to identify interesting data nodes. Since the SLCAs are obtained from a series of intermediate LCAs, it can often incur many unnecessary LCA computations even when the size of final results is small. In this paper, we propose an Iterative-Skip approach to address this challenge. We first study the relation between SLCA candidates and propose a series of properties. Then based on these properties, we design a novel skip strategy to skip more SLCA computations. Experimental results show the effectiveness of our approach.

1 Introduction

Keyword search on XML data has been recently received an increasing interest in database community. Lots of research has been conducted on the semantics and techniques to keyword search over XML databases.

In this paper we concentrated on query processing for SLCA-based keyword search on XML data. To the best of our knowledge, the state-of-art algorithm for SLCA-based keyword search is *Incremental Multiway-SLCA* (*IMS*)[10]. *IMS* algorithm computes each potential SLCA by taking one data node from each keyword list S_i in a single step. It picks an anchor node among the keyword lists to drive the SLCA computation. By doing so, *IMS* skips a lot of redundant nodes. However, some meaningless SLCA candidates for final results are still involved in computations as the following example illustrates.

Example 1. Consider the XML tree T_1 in Figure 1(a). The SLCA for the keywords $\{a, b\}$ in T_1 is $\{b_2\}$. According to IMS algorithm, anchor nodes are $\{b_1,b_2,b_3\}$. Here, anchor nodes b_1 and b_2 are meaningless for final results.

Another drawback of *IMS* algorithm is that it costs a lot in computing the corresponding match from anchor node. The process of obtaining the closest nodes costs $O(dlog(|S_i|))$ time using Dewey labels (d is the depth of the XML document and $|S_i|$ is the set's size), since it involves a binary search on keyword node set which corresponds to each input keywords.

P. García Bringas et al. (Eds.): DEXA 2010, Part I, LNCS 6261, pp. 298–305, 2010.

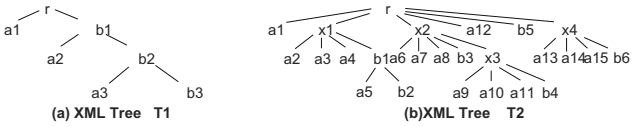

Fig. 1. Example XML Trees T_1 and T_2

We make the following technical contributions in this paper:

1. We study the relationship between LCA nodes and matches, then present effective rules to iteratively skip more redundant computations.
2. We propose the MMPS algorithm to answer SLCA-based queries and implement IMS and MMPS with Dewy and Region encoding schemes respectively. The experimental results demonstrate that our approach outperforms former work.

The rest of this paper is organized as follows: Section 2 introduces the notations used in this paper. Section 3 propose the iterative skip rules and SLCA finding methods, and analyzes the complexity of our approach. In Section 4, we discuss the implement of our approach and experimental results. We introduce related work briefly in Section 5 and give a conclusion in Section 6.

2 Notations

An XML document is viewed as a directed, rooted, ordered, and node-labeled tree. We assign to each node of the tree a numerical id $pre(v)$ which is compatible with pre-order numbering. Given two nodes u and v , $u \prec_p v$ donates $pre(u) < pre(v)$, and $u \preceq_p v$ donates $u \prec_p v$ or $u = v$. Similarly, $u \prec_a v$ donates node u is an ancestor of v, and $u \preceq_a v$ donates $u \prec_a v$ or $u = v$.

Let $K = (k_1, k_2 \dots k_n)$ donates a set of input keywords, where each k_i associated with S_i (set of XML data nodes, sorted in pre-order). $V(v_1, v_2 \dots v_n)$ is called a match, where $v_i \in S_i$ and vl is the leftmost node in match while vr is the rightmost node (with the property that $vl \preceq v_i \preceq vr$). Function $lca(V)$ (also represented as $lca(v_1, v_2, \dots, v_n)$) returns the LCA of nodes $v_1, v_2 \dots v_n$. Function $vl(V)$ (also $vl(v_1, v_2 \dots v_n)$) returns the leftmost node of V. $vr(V)$ (also $vr(v_1, v_2 \dots v_n)$) returns the rightmost node of V.

For matches $W(w_1, w_2 \dots w_n)$ and $V(v_1, v_2 \dots v_n)$. We say match W precedes match V (or V behind/succeds W) donated as $W \prec_p V$, if and only if W and V satisfy (1) $w_i \prec_p v_i$, $(1 \leq i \leq k)$, and (2) $W \neq V$.We say matches W and V share the same node, if there exists i, satisfied $w_i = v_i$.

3 Our Approach

In this section, we introduce properties and rules to skip matches and present key concepts *SMatch* and *SNMatch*. Then, we propose our *MMPS*(*Multi-way Match Progress SLCA*) algorithm in detail and give a complexity analyze on it.

3.1 Properties and Skipping Rules

Lemma 1. *Given a match* $V(v_1, v_2 \dots v_n)$. $lca(V) = lca(vl(V), vr(V))$

Lemma 2. *Given matches* $V(v_1, v_2 \dots v_i \dots v_n)$ *and* $V'(v_1, v_2 \dots v_{i'} \dots v_n)$. *If* $v_i \prec_a v_{i'}$, *then* $lca(V) \preceq_a lca(V')$

Lemma 3. *Given matches* V_1 *and* V_2. *If they share the same node* v *and* $lca(V_1)$ *is an SLCA, then there must be* $lca(V_2) \preceq_a lca(V_1)$

Lemma 4. *Given matches* V *and* W ($V \prec_p W$), *if* $lca(V) \not\preceq_a lca(W)$, *then for any match* X *that* $W \preceq_p X$ *there must be* $lca(V) \not\preceq_a lca(X)$

Lemma 5. *Given matches* V *and* W *that* $V \prec_p W$, *if* $lca(W) \not\preceq_a lca(V)$, *then for any match* X *that* $X \prec_p V$ *there must be* $lca(W) \not\preceq_a lca(X)$.

Rule 1. *For two nodes* u *and* v *in a keywords list* S_i, *if* $u \prec_a v$ *node* u *can be skipped in SLCA computations.*

Rule 2. *Let* vr *be the rightmost node in match* V, *for other node* v_i, *if there exists* $v_{i'}$ ($v_i \prec_p v_{i'} \prec_p vr$), *then node* v_i *can be skipped.*

Definition 1 (SMatch). *Given a match* $V(v_1, v_2, \dots, v_n)$, *it is a SMatch if and only if rule 1 and rule 2 can not skip any node of it*

Definition 2 (NMatch). *Given matches* V *and* W ($V \prec_p W$). *If there exists no match* X *satisfied* $V \prec_p X \prec_p W$, *then we say* W *is match* V*'s* $NMatch$.

Definition 3 (SNMatch). *Let* vl' *be the next node after* vl *in the keywords set* S_i. *We call the match* $V'(v_1, v_2, \dots vl', \dots, v_n)$ *the* $SNMatch$ *of* $SMatch$ V.

Example 2. Consider the keyword search {*a*, *b*} on tree T_2 in Figure 1(b). The keyword sets are $S_1 = \{a_1, a_2 \dots a_{15}\}$, $S_2 = \{b_1, b_2, \dots b_6\}$, respectively. The very beginning match is $\{a_1, b_1\}$ and matches $\{a_1, b_1\}$, $\{a_2, b_1\}$, and $\{a_3, b_1\}$ can be safely skipped according to rule 2. And node b_1 can be skipped as it has a descendant b_2 in S_2 according to rule 1. By repeatedly using rule 1 and rule 2, we finally get *SMatch*(smallest match) $\{a_5, b_2\}$ and *NMatch*es(next matches) of it are $\{a_6, b_2\}$ and $\{a_5, b_3\}$ in which $\{a_6, b_2\}$ is *SNMatch*(special next matches).

Since each SLCA corresponds to one *SMatch*, other matches are not considered in SLCA computation.However because not all *SMatch*es contribute to the final results, two methods are used to prune unnecessary *SMatch*es.

First, the LCA of *SNMatch* of the *SMatch* must be a non-descendant node of the LCA of *SMatch*, or progress to the next *SMatch*. We call such LCA node candidate SLCA node. This method guarantees that candidate SLCA node does not have a descendant LCA node of matches behind it. *NMatch*es (except*SNMatch*) have LCAs that are not descendant nodes of the LCA of *SMatch*, because *NMatch*es (except*SNMatch*) share the same *vl* node with *SMatch* but

have a big pre-order numbering *vr* node. So if the LCA of *SNMatch* is not a descendant node of the LCA of *SMatch*, none of the LCAs behind are descendants of the LCA of this *SMatch* according to lemma 4 with the fact that matches behind *SMatch* either succeed *NMatch* or are *NMatch* themselves.

Second, the LCA of this *SMatch* should not be an ancestor of (nor the same node with) the previous SLCA node. This guarantees that the LCA of this *SMatch* will not have any descendant LCA previously according to lemma 5.

Using this two methods, all unnecessary *SMatch*es are pruned and is sure to generate an SLCA node.

3.2 MMPS Algorithm

We use a cursor array C to maintain the current matches of which each $C[i]$ ($1 \leq i \leq k$, k is the number of input keywords) points to a node in $S[i]$. The *SMatch* is computed by repeatedly using rule 1 and 2 to skip nodes until no match can be skipped. The *SNMatch* is computed from an *SMatch* by progressing the *vl* node to the next node in the keyword set according to its definition.

Algorithm 1. *SMatch* && *SNMatch*

```
1: procedure SMatch(C[1], C[2] ... C[k])
2:     repeat                          ▷ use some variables to detect any match been skipped
3:         for each C[i] in C do
4:             repeat
5:                 tmp = C[i];C[i] = tmp++;                    ▷ tmp is a cursor
6:             until C[i] ≠_a tmp && tmp ≠_p vr(C)    ▷ Using rule 1 and 2 to skip
7:         end for                                          ▷ ↪ unnecessary matches
8:     until no match can be skipped
9:     return C
10: end procedure

11: procedure SNMatch(C[1], C[2] ... C[k])
12:     Let C[l] be the vl node of C
13:     C[l]++;
14:     return C
15: end procedure
```

The general idea of our approach is showed in algorithm 2 MMPS. First, we get an initial match (at the very begin is $\{S[1][1], S[2][1] \ldots S[k][1]\}$). Then we use *SMatch* and *SNMatch* to get a candidate SLCA node in line 3-6 and validate the candidate on line 7, and update the final result in line 8 if true. From line 10 to 14, we calculate the next initial match, in which each node is a non-descendant of α_a. By doing this repeatedly, we generate final results. This loop stops when any $C[i]$ points to the end of $S[i]$ which is caught by *try/catch* structure.

Algorithm 2. MMPS $(S[1], S[2] \dots S[k])$

Require: each $S[i]$ is the nodes list that responds to a input keywords k_i

establish a cursor array C; n=0; α=*null* ▷ each $C[i]$ points to the first node of
repeat ▷ ↪ $S[i]$, α maintains the final result
  repeat
    α_a=$lca(SMatch(C[1], C[2] \dots C[k]))$
    α_b=$lca(SNMatch(C[1], C[2] \dots C[k]))$
  until $\alpha_a \not\leq_a \alpha_b$
  if n==0 or $\alpha_a \not\leq_a \alpha[n]$ **then**
    $\alpha[n{+}{+}] = \alpha_a$;
  end if
  for each $C[i]$ **do** ▷ calculate the next initial match
    repeat
      $C[i]{+}{+}$
    until $\alpha_a \not\leq_p C[i]$
  end for
until some $C[i]$ points to the end of $S[i]$ ▷ *try/catch* structure should be used
return α ▷ ↪to avoid boundary exceeded error

3.3 Analysis

Our approach is suitable for both Dewey and Region scheme but costs differently. For Dewey scheme, it costs $O(d)$ time to compute the relationships (ancestor/descendant and precede/succeed) of two nodes as well as the LCA of two nodes(d is the average depth of the XML tree), while it costs $O(1)$ time on Region scheme[1]. The vl and vr functions cost $O(kd)$ time to compute the leftmost(rightmost) node of a match on Dewey, which only costs $O(k)$ time on Region code(k is the keywords number). In MMPS, we do not enumerate all the cases but only consider the matches which are liable to be an SLCA candidate. Let S_{max} be the largest keyword node set and S_{min} be the size of the smallest set, then at most O($|S_{max}|$) matches are visited which costs $O(kd|S_{max}|)$ time on Dewey and $O(k|S_{max}|)$ time on Region, and the whole candidate $SLCA$ match number is no larger than $|S_{min}|$. So the time complexity of our approach is $O(kd|S_{max}| + d|S_{min}|)$ on Dewey code and $O(k|S_{max}| + |S_{min}|)$ on Region code. In comparison, the time complexity of IMS are $O(k|S_{min}|log(|S_{max}|))$ on Region code and $O(kd|S_{min}|log(|S_{max}|))$ on Dewey code, respectively.

4 Experimental Results

To verify the effectiveness of our approach, we implement both MMPS and IMS algorithm on Dewey and Region scheme and compare the results.

4.1 Experiment Setup

All Experiments are implemented in $C{+}{+}$ language.We run experiments on the data sets of DBLP[8] with 8.8 million data nodes and XMark[6] with 3.6 million

data nodes. These two data sets are encoded with Dewey scheme and Region scheme, respectively. Similar to other SLCA-based algorithms, all the data nodes are organized as a B+-tree where keys are the keywords of the data nodes and data associated with each key is a list of Dewey codes (or Region codes) of the data node contained that keyword. Our implementation use Berkeley DB (v4.8 C++ Edition)[5] to store the keyword data list. Berkeley DB is configured using 4KB size page and 1 GB size cache . All the experiments are done on a 2GHz dual-core laptop PC with 2 GB of RAM.

In particular, we pre-process the XML data and built an index to accelerate the *LCA* computation[2]. It costs 6.602 seconds to build the index for XMark data and 14.202 seconds for DBLP data. Before keyword search, index is loaded into memory first. It costs 4.218 seconds to load index of XMark and 9.839 seconds for DBLP. These extra works can be done off-line, and will not affect I/O time and query performance.

Similar to [9], experiments are carried out by using different classes of queries. Each class is written as $data\_kN\_L\_H$, where *data* denotes the XML data and N, L and H are three positive integer values: N shows the number of the keywords, L and H with $L \leq H$ represent two keyword frequency that one of the N keywords has the low frequency L while each of the remaining $N-1$ keywords has the high frequency H.

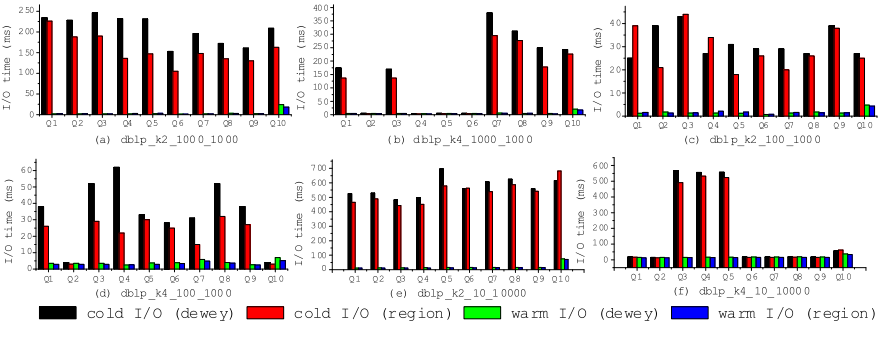

Fig. 2. I/O costs on DBLP

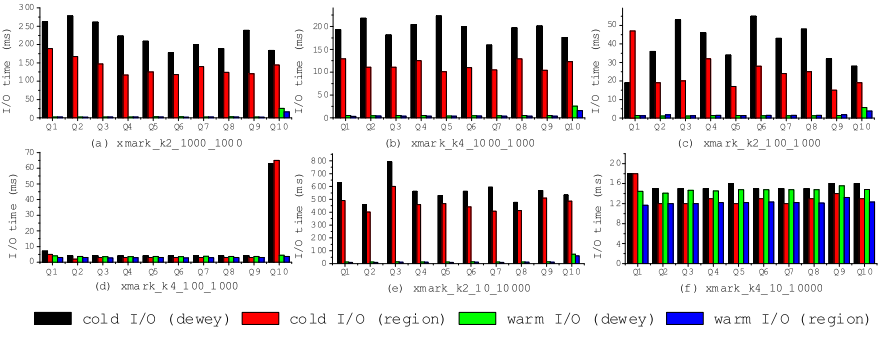

Fig. 3. I/O costs on XMark

4.2 Result Analysis

Figure 2 and 3 show the I/O cost of the two encoding schemes on DBLP and XMark. Generally speaking, MMPS on Dewey scheme costs more time than Region for the length of keywords indexed by Dewey scheme is not fixed and more time is needed when reading data from index. But there are exceptions in cold I/O for the initial position of the disk head is random. On the other hand, Region scheme has advantage in warm I/O because the initial position of the disk head is beside the data.

Figure 4 and 5 show the performances of two algorithms on both encoding schemes. First,it is clear that the *SLCA* calculation based on Dewey costs more time for in each comparison operation of data node or LCA computation, Dewey scheme costs d times than Region. Second, we can see that algorithm MMPS performs better than IMS in general. Though exceptions exist few, such as $Q5$ in figure 4(d) and $Q10$ in figure 5(e). However, in such cases, the time cost is few and IMS does not perform much better. Figure 4 shows the performances on DBLP and Figure 5 shows the performances on XMark. When the L value decreases, both algorithms trends to reduce the calculating time and MMPS performs better. In any cases, MMPS outperforms IMS.

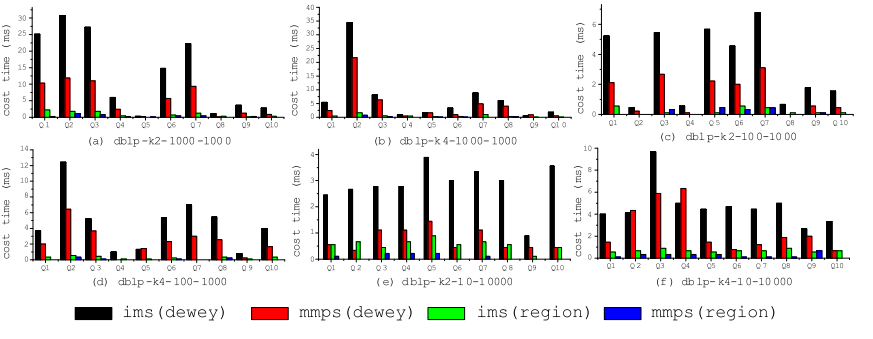

Fig. 4. Processing Time on DBLP

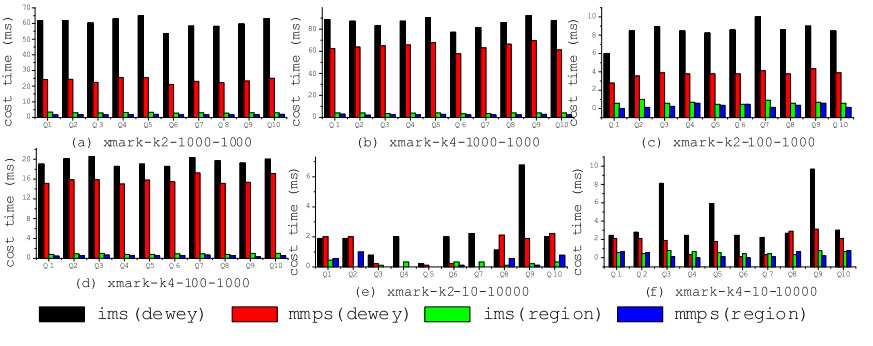

Fig. 5. Processing Time on XMark

5 Related Work

Recently, lots of research has been concentrated on semantics and techniques for keyword search over XML data, such as XRank[3], SLCA [10], VLCA [4], XSeek [7]. In these research, LCA computation is a common problem. Lots of research has been concentrated on the techniques for this problem [1,2].

Our work is closely related to two research work [10,9]. [10] proposes two efficient algorithms: the Scan Eager (SE) algorithm and the Index Lookup Eager (ILE) algorithm. [9] proposes a more efficient algorithm named IMS than the ILE and SE algorithms by using several optimization strategies to reduce the number of LCA computation. Moreover, [9] extendeds its algorithm to support more general search queries involving AND and OR semantics. In particular, the IMS algorithm is compared with in this paper.

6 Conclusions

In this paper, we propose a novel SLCA-based keyword search approach MMPS. Since MMPS utilize iterative skip to reduce redundant SLCA computations, it outperforms the state-of-the-art algorithm IMS. We also compare results of our method with Dewey and Region encoding schemes. Experimental results show that Region code is effective to compute SLCAs over XML data.

Acknowledgements

The work was supported by the National Natural Science Foundation of China under Grants No. 60703093 and No. 60673133, the National Grand Fundamental Research 973 Program of China under Grant No. 2005CB321905.

References

1. Bender, M.A., Farach-Colton, M.: The lca problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 123–125. Springer, Heidelberg (2000)
2. Fischer, J., Heun, V.: Theoretical and practical improvements on the rmq-problem, with applications to lca and lce. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 36–48. Springer, Heidelberg (2006)
3. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: Xrank: Ranked keyword search over xml documents. In: SIGMOD 2003 (2003)
4. Wang, J., Li, G., Feng, J., Zhou, L.: Effective keyword search for valuable lcas over xml documents. In: CIKM 2007 (2007)
5. Berkeley DB: `http://www.oracle.com/database/berkeleydb/index.html`
6. XMark Project, `http://www.xmlbenchmark.org/`
7. Liu, Z., Chen, Y.: Identifying meaningful return information for xml keyword search. In: SIGMOD 2007 (2007)
8. DBLP XML records, `http://dblp.unitrier.de/xml/`
9. Sun, C., Chan, C.-Y., Goenka, A.K.: Multiway slca-based keyword search in xml data. In: WWW 2007 (2007)
10. Xu, Y., Papakonstantinou, Y.: Efficient keyword search for smallest lcas in xml databases. In: SIGMOD 2005 (2005)