THE NATIONAL UNIVERSITY
of SINGAPORE

School of Computing
Computing 1, 13 Computing Drive, Singapore 117417

TRB3/12

Problems of LCA and Impact of ORA-semantics in XML Keyword Search

Thuy Ngoc Le, Huayu Wu, Tok Wang Ling and Luochen Li

March 2012

T e c h n i c a l R e p o r t

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

OOI Beng Chin
Dean of School

Problems of LCA and Impact of ORA-semantics in XML Keyword Search

Thuy Ngoc Le [#], Huayu Wu [*], Tok Wang Ling [#], and Luochen Li [#]

[#]School of Computing, National University of Singapore
{ltngoc,luochen,lingtw}@comp.nus.edu.sg
[*] Institute for Infocomm Research, Singapore
huwu@i2r.a-star.edu.sg

Abstract. Most keyword search approaches for data-centric XML documents are based on the computation of Lowest Common Ancestors (LCA). However, LCA-based search methods depend much on hierarchical structures of XML data. Therefore it may not be able to find desired answers for many keyword queries since a relationship among objects in XML data can be represented in different hierarchical structures. In this paper, we first point out serious problems of the LCA-based approach, due to its unawareness of semantics of object, relationship and attribute, referred to as ORA-semantics. Through detailed analysis of these problems, we show the impact of ORA-semantics in XML keyword search. We then propose an ORA-semantics based approach with rules to infer expected answers for XML keyword queries. Experimental results show that our ORA-semantics based approach can resolve the problems of the LCA-based approach, and thus can be a promising research direction for XML keyword search.

1 Introduction

As XML is becoming a standard format for data representation and exchange, keyword search in XML data has attracted a lot of research interest in recent years. Given a keyword query to a data-centric XML document, XML keyword search aims to find the most relevant information in the corresponding XML document. Inspired by the hierarchical structure of XML data, Guo et al. [5] and Schmidt et al. [13] proposed an LCA-based (Lowest Common Ancestor) search model to perform XML keyword search. Many subsequent works either enhance the efficiency of LCA computation [3] or improve the quality of LCA-based search [14, 10, 11]. However, if a keyword query involves relationships, these LCA-based methods do not always give appropriate answers.

For clarity, we motivate our work with an example. Consider the scenario in which a university needs to maintain an XML database for course registration. To clearly explain the data, we use an ER (Entity-Relationship) diagram to model the information, as shown in Fig. 1, where there are two many-to-many relationship types, between *Student* and *Course* and between *Course* and *Textbook*. Fig. 2 shows two corresponding possible XML designs. For simplicity, we only draw object class nodes and relationship attribute nodes in these schema diagrams. The two instance diagrams of these two schemas are the SC-XMLDB in Fig. 3(a), in which *Student* nodes are ancestors of *Course* nodes, and the CS-XMLDB in Fig. 3(b), in which *Course* nodes are ancestors of *Student* nodes.

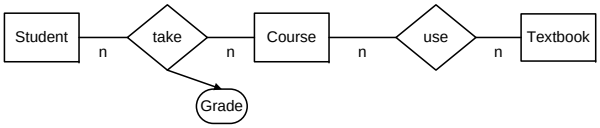
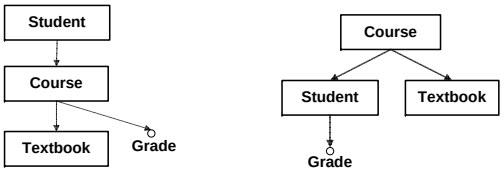


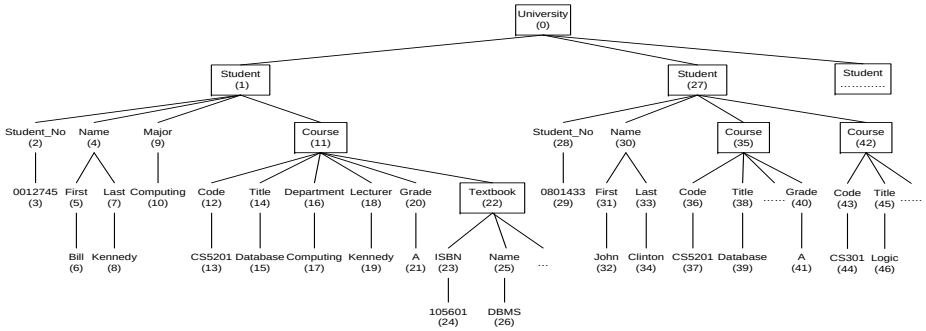
Fig. 1. ER diagram of the University database



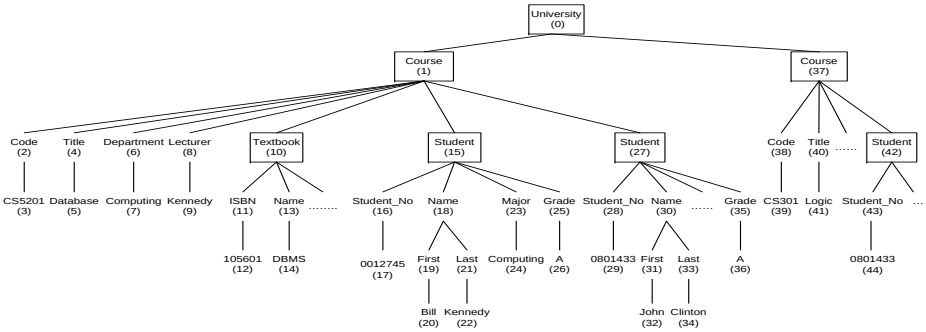
(a) SC-XMLDB schema (b) CS-XMLDB schema

Fig. 2. Two possible XML schemas of the University database

Suppose there are two XML keyword queries, $Q = \{Bill, John\}$ and $Q' = \{Database, Logic\}$ where *Bill* and *John* are names of two students and *Database* and *Logic* are parts of the title of two courses. With the SC-XMLDB in Fig. 3(a), the LCA-based approach returns the document root as the answer for Q, because only the root is the LCA of the query keywords. This answer is meaningless. The LCA-based approach faces this problem because it cannot discover the relationship type between *Course* and *Student* in order to find the common courses that both students *Bill* and *John* take. It is even worse that under the LCA search model, a user cannot find any keyword query to fulfil the requirement of Q. With the CS-XMLDB in Fig. 3(b), the LCA-based approach can answer Q. However, with the CS-XMLDB, it cannot find correct results for



(a) The SC-XMLDB



(b) The CS-XMLDB

Fig. 3. Two different instance diagrams of the same University database

Q' and the user cannot find any keyword query to meet the search intention of Q' either.

The above example shows that once relationships among objects are involved, the LCA-based approach cannot find correct answers for many keyword queries. This is a significant issue because relationships do frequently exist in practical databases. Generally, any two objects connected by an edge in an XML data tree may be involved in some relationship, with or without relationship attributes under the lower object participating in the relationship. For any XML document with a certain depth in its structure, there may be many relationships among objects.

We will discuss more problems of the LCA-based XML keyword search approach in Section 2, which are caused by its unawareness of semantics of object, relationship and attribute, referred to as ORA-semantics. By analysing these problems, we show the importance of ORA-semantics to the correctness of XML keyword search and propose an ORA-semantics based approach in Section 3. In brief, the contributions of our work are as follows.

- We discover and illustrate limitations of the existing LCA-based XML keyword search approaches. This finding is important since most current XML keyword search approaches are LCA-based. To the best of our knowledge, this is the first work pointing out problems of the LCA-based methods.
- We show that ORA-semantics plays important roles in XML keyword search. Thus, we propose to use ORA-semantics to process XML keyword queries. In particular, we propose several rules to identify expected answers of an XML keyword query.
- Experimental results show that our approach can return correct and meaningful answers for queries involving relationship, which cannot be correctly answered by the LCA-based approach.

2 Problems of the LCA-based approach for XML keyword search

In an XML instance diagram, a node may represent different things such as an object, an object attribute, an object attribute value, a relationship attribute or a relationship attribute value. For example, in the SC-XMLDB, objects are *Student_(1)*, *Course_(11)*, *Textbook_(22)*, etc. *Major* is an attribute of the object class *Student*; *Computing_(10)* is a value of object attribute *Major*; *Grade* is a relationship attribute of the relationship type between object classes *Student* and *Course* and *A_(21)* is one of its value.

Most practical XML keyword queries involve objects and/or relationships among objects. We classify queries into three cases based on how query keywords match objects and relationships: (1) query keywords match only one object; (2) query keywords match more than one object but no relationship attribute value; and (3) some query keywords match relationship attribute values. In each case, we point out the problems of the LCA-based XML keyword search approach by showing the differences between the LCA result and the expected result. We use the SC-XMLDB and the CS-XMLDB instance diagrams in Fig. 3(a) and Fig. 3(b) respectively for illustration. The label of each node in these diagrams is only for description purpose.

2.1 Case 1: Keywords matching only one object

Example. $Q_1 = \{Bill\}$

LCA-based answer. The LCA-based approach identifies node *Bill_(6)* in the SC-XMLDB or *Bill_(20)* in the CS-XMLDB as an answer. However, returning this node is not useful since it does not provide any supplemental information about *Bill*.

Expected answer. The expected answer should be the object matching keyword *Bill*, i.e., *Student_(1)* in the SC-XMLDB, or *Student_(15)* in the CS-XMLDB since they contain useful complementary information related to *Bill*.

2.2 Case2: Keywords matching multiple objects, but no relationship attribute value

We further divide queries of this case into two subcases according to the way the objects referred by a query are related to each other, i.e., whether they have any direct relationship or not.

Case 2(A): Objects having direct relationships

Example. $Q_2 = \{Bill,\ Database\}$

LCA-based answer. With the SC-XMLDB, the LCA-based approach returns object *Student_(1)*. The subtree rooted at node *Student_(1)* contains too much irrelevant information, e.g., all other courses he takes and textbooks used in these courses. With the CS-XMLDB, the subtree rooted at *Course_(1)* is returned. This subtree also contains extraneous information, e.g., all other students taking this course. In both cases, the irrelevant information makes users difficult to identify essential answers. Filtering such irrelevant information is not trivial. Some works such as [6] returning path can filter irrelevant information but they also filter relationship attributes which users wish to know. Moreover, the corresponding results for different schemas, the SC-XMLDB schema and the CS-XMLDB schema, are different though these schema designs contain exactly the same information and we are dealing with the same query.

Expected answer. The expected answer should be student *Bill* and course *Database* along with all accompanied information about the relationship between them such as the relationship attribute *Grade*, but should not contain any irrelevant information.

Case 2(B): Objects having no direct relationship

Example. $Q_3 = \{Bill,\ John\}$

LCA-based answer. With the SC-XMLDB, the LCA-based approach returns the document root which is definitely not meaningful. Because the LCA can never discover that, despite of appearing as different nodes, *Course_(11)* and *Course_(35)* refer to the same object *Course CS5201*, the common course taken by the two students. If the CS-XMLDB is used, the LCA-based approach can return *Course_(1)* as the LCA node. However, if another query involving two courses, e.g., {*Database*, *Logic*} is issued to the CS-XMLDB, the LCA-based approach will also return the document root since the common student taking both courses, *Student 0801433*, cannot be discovered.

Expected answer. The expected answer should contain student *Bill* and student *John* with the object connecting them, i.e., the common course, *Course CS5201*, taken by these two students.

2.3 Case 3: Some keywords matching relationship attribute values

Example. $Q_4 = \{Database,\ A\}$

LCA-based answer. With the SC-XMLDB, the LCA-based approach returns *Course_(11)* and *Course_(35)* as answers. Although these two nodes appear at different places in the XML data tree, they refer to exactly the same object, the course with *Code CS5201*. The above *duplicate* answers cannot provide the students getting *'A'* grade for the course *Database*. They are incorrect because *Grade* is an attribute of the relationship type between *Student* and *Course*, rather than an attribute of object *Course*. The LCA-based approach cannot distinguish between an *object attribute* and a *relationship attribute* under an object node. With the CS-XMLDB, the problem seems to be solved since the LCA-based approach returns the *Course_(1)* whose subtree contains the information of students taking this course and getting *'A'*. However, it also contains a lot of irrelevant information, such as all other students taking this course. Moreover, if a user issues a query {*Bill*, *A*} to the CS-XMLDB, the LCA-based approach faces the same problem, i.e., returning *Student_(15)* without providing in which course student *Bill* gets *'A'*.

Expected answer. The proper answer should be all students taking the *Database* course and getting an *'A'* grade, as well as the relationship between each student and the course *Database*, including all relationship attribute values (if any) such as *Grade*.

Summary. Although we only analyze binary relationships, the discussed problems also generalize to n-ary and recursive relationships. The main reason of these problems is that the LCA-based approach heavily relies on the hierarchical structure of an XML data while *ignores the real semantics* of objects, relationships and their attributes.

3 ORA-semantics based XML keyword search approach

In this section, we first show the importance of ORA-semantics in XML keyword processing. We then propose to use ORA-semantics to process XML keyword queries and we define rules to provide a conceptual-level understanding of our ORA-semantics based XML keyword search approach. Finally, we present our implementation to illustrate the proposed approach.

3.1 Impact of ORA-semantics in XML keyword search

The term *semantics* has different interpretations. In this paper, we define the concept of ORA-semantics as the semantics of *objects*, *relationships* and *attributes*. Specifically, in an XML schema tree, ORA-semantics includes the identification of object class, object identifier (OID), object attribute, and explicit/implicit relationship type with or without relationship attributes. These concepts are similar to those in ER model.

Recall the query $Q_3 = \{Bill, John\}$ to the SC-XMLDB studied in Section 2.2, the LCA-based approach returns the document root, a meaningless answer, since it cannot find the common courses taken by both students *Bill* and *John*. One possible structured XQuery query that correctly searches for desired answers for Q_3 is as follows.

For *$s1=doc(SC-XMLDB.xml)//Student[Name/First=*Bill*]*
For *$s2=doc(SC-XMLDB.xml)//Student[Name/First=*John*]*
Where *$s1/Course/Code=$s2/Course/Code*
Return *$s1/Course*

In this XQuery query, we need to join two paths *$s1/course/code* and *$s2/course/code*, based on the same *Code* value. In order to compose this query, we need to know the *semantics* that *Course* and *Student* are *object class*, *Code* is the *object ID* of *Course* and there is a *relationship type* between *Student* and *Course*. Without such ORA-semantics, a user cannot issue the above XQuery query. A user composes an XQuery expression based on his search purpose, which is actually built on top of ORA-semantics. For example, an XQuery expression joins objects based on object identifications, and returns interesting objects and/or relationships or their attributes. Then the search engine just systematically executes the query. However, in XML keyword search, a keyword query conveys little semantics. Thus, it is desirable that the engine exploits ORA-semantics to find more accurate answers. Otherwise, the result may be unsatisfactory, as illustrated above. In brief, no matter what forms of queries, ORA-semantics is necessary to guarantee the correctness of XML query processing.

3.2 ORA-semantics based approach with rules

Our approach works at object and relationship level which means that each query keyword utilizes the ORA-semantics of the corresponding objects and/or relationships. In contrast to the LCA-based approach, in which the results depend on the XML hierarchical structure, our approach is independent of the XML hierarchical structure, i.e., it returns the same results no matter which schema design of the data is used.

The process of our approach is shown in Fig. 4 and can be briefly described as follows. We first match each query keyword to a set of corresponding objects and/or relationships. We then exploit ORA-semantics to find meaningful connections (if any) between them. In particular, for each object matching some query keywords, we find the relationships in which this object participates and find the objects having relationships with this object. Based on these meaningful connections, we finally apply rules to identify expected answers. For clarity, we summarize the concepts and notations used in this paper in Table 1 and Table 2 respectively.

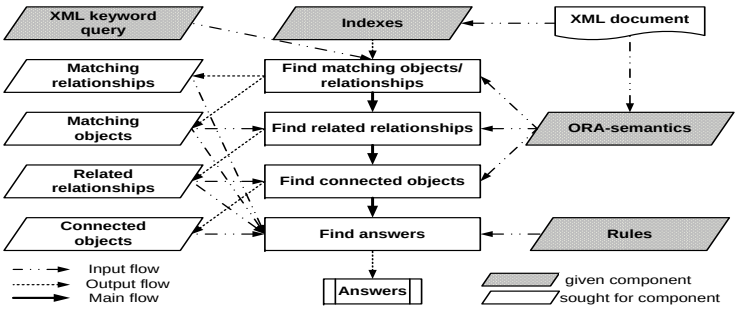


Fig. 4. Process of our ORA-semamtics based approach with rules

To determine the expected results, we propose several rules. In these rules, returning an object means returning all attribute values of that object; while returning a relationship means returning all attribute values of that relationship as well as all objects participating in that relationship. To illustrate our rules to show that our approach can handle the discussed problems of the LCA-based approach, we re-use the examples in Section 2. Although only the SC-XMLDB instance diagram is used for illustration, these rules also enable us to return the same answers for any schema designs, including the CS-XMLDB. In other words, unlike LCA-based approach, our rules are independent of the hierarchical structure of XML data.

Before presenting the details of our rules, we use a decision tree in Fig 5 to show the generality and completeness of our rules. As can be seen, we can apply one of six rules to a issued query. Thus, our rule-based approach can cover all cases of practical queries. For ambiguous queries in which some keywords may have several semantic meanings, our approach will apply different rules correspondingly and return diverse answers.

Assume that in the following rules, a query $Q = \{k_1, \ldots, k_n\}$ with n keywords is issued to a data-centric XML document D.

Table 1. Concepts

Concept	Description
Keyword k matching object o (or object o matching keyword k)	Keyword k is contained by the class name, any attribute or any attribute value of object o.
Keyword k matching a relationship r (or relationship r matching keyword k)	k is contained by any attribute of relationship type of r or any attribute value of relationship r.
Object o participating in relationship r	r is a relationship among object o and other object(s).
Keyword k matching objects of relationship r (keyword k involving r)	Keyword k matches at least one object which participates in relationship r.
Object o_i having a relationship with object o_j (or objects o_i and o_j have a relationship)	There exists a relationship among objects o_i, o_j and other objects (if any).
Keyword k connecting object o (or object o connecting keyword k)	Keyword k matches at least one object which has a relationship with object o.

Table 2. Notations

Notation	Description
$Res(Q, D)$	The result of query Q to a document D
$R(o_i, o_j)$	Objects o_i and o_j have a relationship of relationship type R
$r(o_1, \ldots, o_m)$	Relationship among objects $o_1, \ldots, o_m, m \geq 2$
$Class(R)$	The classes involving in relationship type R
$Obj(k)$	The set of objects matching keyword k
$Obj(Q)$	The set of objects matching all keywords in query Q. If $Q = \{k_1, \ldots, k_n\}$, then $Obj(Q) = \bigcap Obj(k_i)$, $i = 1..n$
$Obj(r)$	The set of objects joining in relationship r
$Rel(o)$	The set of relationships in which object o participates in
$Rel(k)$	The set of relationships which matches keyword k
$Rel(Q)$	Set of relationships matching all keywords in query Q. If $Q = \{k_1, \ldots, k_n\}$, then $Rel(Q) = \bigcap Rel(k_i)$, $i = 1..n$

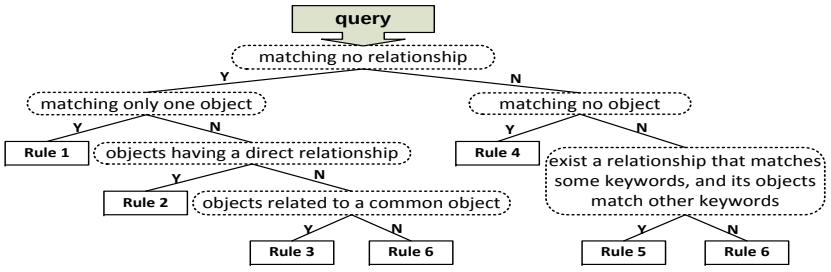


Fig. 5. Decision tree for semantic rules

Rule 1 *If* $o \in Obj(k_i)$, $\forall i = 1..n$, *then* $o \in Res(Q, D)$.

Rule 1 states that if an object matches all query keywords, this object is an answer.

Example 1 *Consider a query* $Q_1 = \{\text{Bill}\}$ *which matches object* Student_(1). *By Rule 1,* Student_(1) *is an answer. The LCA-based approach only returns node* Bill *while our approach returns object* Student_(1) *which provides more useful information about* Bill.

Rule 2 *Let* $o_i \in Obj(k_i), i = 1..n$ *and* r *be a relationship. If* $(|\bigcup\{o_i\}| \geq 2)$ *and* $(\bigcup\{o_i\} \subseteq obj(r))$, *then* $r \in Res(Q, D)$.

Rule 2 states that if all query keywords match objects of a relationship, then this relationship is an answer. The condition $(|\bigcup\{o_i\}| \geq 2)$ ensures at least two objects participating in that relationship.

Example 2 *Consider a query* $Q_2 = \{\text{Bill, Database}\}$. *Keywords* Bill *and* Database *match object* Student_(1) *and* Course_(11) *respectively. Since there is a relationship between these two objects, by Rule 2, that relationship, i.e.,* Bill *takes* Database *and gets an* A, *is an answer.*

Rule 3 *Let* $o_i \in Obj(k_i), i = 1..n$ *and* o *be an object. If* $|\bigcup\{o_i\}| \geq 2$ *and* $R_i(o, o_i)$, *i.e.* $\exists r$, $r(o, o_i)$, *then* $(o, o_1, \ldots, o_n) \in Res(Q, D)$.

Rule 3 states that if a common object connects all query keywords, then this common object together with all matching objects are returned as an answer.

There may be more than one relationship type between two object classes. In this case, if two object o_q and o_p belong to the same object class, then the relationship between the common objects o and each of them may belong to the same relationship type or different relationship types. In both cases, our approach will return the common object o. We assign a higher rank for the common object which involves in the same relationship type with both o_q and o_p, and this case is illustrated in Example 4. The case where there is only one relationship type is illustrated in Example 3.

Example 3 *Consider a query* $Q_3 = \{\text{Bill, John}\}$. *Keywords* Bill *and* John *match object* Student_(1) *and* Student_(27) *respectively. There is an object Course* CS5201 *having relationships with both of them. Although this object appears as two nodes,* Course_(11) *and* Course_(35), *these nodes refer to the same object since their ID attribute values are the same, the code* CS5201. *Therefore, based on Rule 3,* Course CS5201, *together with students* Bill *and* John *are returned as an answer.*

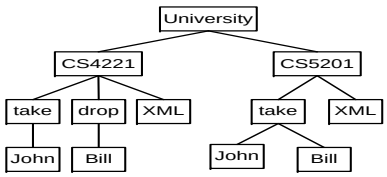


Fig. 6. Two relationship types between the same set of object classes

Example 4 *Consider a scenario where there are two relationship types, namely* take *and* drop*, between two object classes* Student *and* Course*, which is shown in an instance diagram in Fig. 6. Student* Bill *drops course* CS4221*, student* John *takes course* CS4221 *and both of them take course* CS5201*. For explanation purpose, this instance diagram is at object level.*

With a query Q_{3b} = {John, Bill}*, there are two common courses, namely* CS5201 *and* CS4221*, having relationships with both of them. According to Rule 3, both of course* CS5201 *and course* CS4221 *are returned as answers. However, course* CS5201 *should have a higher ranking than course* CS4221 *in the results, because both* John *and* Bill *involve in the same relationship type with course* CS5201*, while* John *and* Bill *involve in different relationship types with course* CS4221*.*

Rule 4 *If* $r \in Rel(k_i)$, $\forall i = 1..n$, *then* $r \in Res(Q, D)$.

Rule 4 states that if a relationship matches all query keywords, then that relationship is an answer. This kind of query is rather rare in practice, as most relationships are issued in queries together with some objects.

Rule 5 Q *is partitioned into two non-empty sets* Q_1 *and* Q_2*, i.e.,* $Q_1 \bigcap Q_2 = \emptyset$ *and* $Q_1 \bigcup Q_2 = Q$*. If* $r \in Rel(Q_1)$ *and* $r \in Rel(Obj(Q_2))$*, then* $r \in Res(Q, D)$.

Rule 5 states that if a relationship r matches some query keywords and all other query keywords match some objects of r, i.e., each keyword either matches an attribute value of that relationship or matches its participating object(s), then r is an answer.

Example 5 *Consider a query* Q_4= {Database, A}. 'A' *is the value of relationship attribute* Grade *which belongs to a relationship between some* Student *object and some* Course *object.* Database *matches a* Course *object. Then relationships between course* Database *and students taking* Database *course and getting* 'A' *are returned. The result contains information about all students taking* Database *and getting an* 'A'*, as well as other information (if any) of the relationship type such as midterm grade.*

The above rules handle popular cases of practical queries where matching objects and/or relationships are close together. The case where keywords are less closely related, i.e., the case does not fit in any previous rules, is seldom issued in practice. We still handle this case in Rule 6.

If a query refers to multiple relationships and multiple objects, we have to find a minimal tree T of a connected, undirected graph which covers all objects and relationships of the considered database. T must be the minimum tree among tress which connect all keywords as described in Rule 6.

Rule 6 *Given a query $Q = \{k_1, \ldots, k_n\}$ to a document D. Let G be a connected, undirected graph covering all objects and relationships of the considered database. In particular, each node of G is either an object or a relationship and each edge connects an object and one relationship in which this object participates. $\forall$ tree T such that T satisfies following conditions, then $T \in Res(Q, D)$.*

- *T is a tree of G which contains at least one matching object or relationship of each keyword.*
- *If any node is omitted out of T, then T does not satisfy the above condition.*

Rule 6 implies that if there exist intermediate objects and relationships which form a minimal tree connecting at least one object or relationship matching each query keyword, then the expected result should include that tree.

3.3 Implementation of our approach

To implement our ORA-semantics based approach for XML keyword search studied in Section 3.2, we use object-relational (OR) tables to capture ORA-semantics. However, our approach is different from existing keyword search over relational database (RDB). Although many relational databases are designed based on Entity-Relationship (ER) model, in the physical design of RDB tables, ORA-semantics can hardly be captured. Let's consider the following scenarios. Two object classes with a one-to-many relationship in an ER diagram will be translated into only two tables, one of which represents both an object class and the relationship type. Moreover, an object with multi-valued attributes is usually stored in several tables and primary keys of a table are not always object IDs. Additionally, many tables may be vertically and/or horizontally partitioned for performance concern. These scenarios show that, in most RDB, tables actually do not have much ORA-semantics. Therefore, current keyword search over RDB does not (or even cannot) use ORA-semantics to improve search quality. It only uses key references to join tuples. However, due to lack of ORA-semantics, the answers returned by tuples joined by key references could be meaningless. The

key technology of our approach is to propose a search model based on ORA-semantics. It is noteworthy that we use OR tables only for our search purpose. The whole approach, which is based on ORA-semantics, is different from keyword search over RDB.

Semantics discovery, data storage and indexing

We have designed algorithms to automatically discover ORA-semantics from XML schema and data, by using some heuristic rules together with statistic information, and the discovered ORA-semantics has high accuracy (more than 90%) [9].

We store ORA-semantics in OR tables. Each object table or relationship table captures semantic meanings of an object class or a relationship type, in which each tuple corresponds to an object or a relationship instance. Moreover, there are two metadata tables, one for storing all object classes with their attributes; and the other for storing all relationship types along with their participating object classes.

We use B^+ tree to index all attribute values and tag names in an XML document so that it can efficiently retrieve the set of objects and relationships matching a keyword during query processing.

Query processing. The process of our approach is shown in Algorithm 1, which finds the result $Res(Q, D)$ of a given keyword query Q to an XML document D by using rules in Section 3.2. We find all answers by applying all rules consecutively instead of applying rules for each query interpretation. This makes the process more efficient because the number of query interpretations increases exponentially with the number of matching objects and relationships.

Algorithm 1: Query processing

Input: Query Q, document D
Output: Result $Res(Q, D)$

```
1  Res(Q, D) = ∅
2  matchingObj = matchingObjects(Q, D) //find objects matching all keywords
3  Res(Q, D) = Res(Q, D) ⋃ matchingObj //Rule 1
4  //find relationships which match and/or involve all keywords
5  relatedRel = relatedRelationships(Q, D)
6  Res(Q, D) = Res(Q, D) ⋃ relatedRel //Apply Rule 2, 4, 5
7  connectingObj = connectingObjects(Q, D) //find common objects connecting all keywords
8  Res(Q, D) = Res(Q, D) ⋃ connectingObj //Rule 3
9  if Res(Q, D) = ∅ then
10     Res(Q, D) = find minimal Trees(Q, D)//find minimal trees connecting all keywords
11     Res(Q, D) = Res(Q, D) ⋃ subTrees //Apply Rule 6
```

In Algorithm 1, we use rules consecutively to identify expected answers. In particular, Rule 1 is applied first to find all common objects which match all

keywords. Then, Rule 2, 4, 5 are applied to get all relationships which match and/or involve all keywords. Next, Rule 3 is used to find the common objects having relationships with all keywords. The final result is the set of answers which is the union of answers from all rules. Finally, if applying all above rules gives no answer, we will find minimal trees which connects all keywords. Details of above functions are given in Algorithm 2-8.

Algorithm 2: matchingObjects

Input: Query $Q = \{k_1, \ldots, k_n\}$, document D
Output: Set of objects $matchingObj$ matching all keywords

foreach *keyword* k_i **do**
  $Obj(k_i)$ = get objects matching k_i `//retrieve` B^+ `tree`
`//find objects matching all keywords`
$matchingObj = \bigcap(Obj(k_i)), i = 1..n$

Algorithm 2 finds objects which match all keywords which are answers from Rule 1. Implementation of Rule 1 is straightforward. For each keyword, the system retrieves B^+ tree to get the set of matching objects. The intersection of such sets for all keywords is the set of objects matching all keywords.

Algorithm 3: relatedRelationships

Input: Query $Q = \{k_1, \ldots, k_n\}$, document D
Output: Set of relationships $relatedRel$

$relatedRel = \emptyset$
foreach *keyword* k_i **do**
  $matchingRel(k_i)$ = get relationships matching k_i `//retrieve` B^+ `tree`
  `//get relationships whose objects matching keyword` k_i
  $involvingRel(k_i)$ = getInvolvingRelationships($Obj(k_i)$)
  $referredRel(k_i)$ = get relationships matching and/or involving keywords k_i
`//find relationships matching all keywords`
$commonRel = \bigcap(matchingRel(k_i)), i = 1..n$
$relatedRel = relatedRel \bigcup commonRel$ `//Rule 4`
`//find relationships involving all keywords`
$involvingRel = \bigcap(involvingRel(k_i)), i = 1..n$
$relatedRel = relatedRel \bigcup involvingRel$ `//Rule 2`
`//find relationships matching or involving all keywords`
$referredRel = \bigcap(referredRel(k_i)), i = 1..n$
$relatedRel = relatedRel \bigcup referredRel$ `//Rule 5`

Algorithm 3 find relationships matching and/or involving all keywords. Implementation of Rule 4 is straightforward and similar to that of Rule 1. For each keyword, the system retrieves B^+ tree to get the set of matching relationships.

The intersection of such sets for all keywords is the set of relationships matching all keywords.

Rule 2 finds the set of relationships which involve all keywords. This set is the intersection of sets s of relationships which involve each keyword k. To find s, the system first gets object ID of each object matching k and then finds relationships which contain this ID. Those relationships involve keyword k since they contain object ID of some object matching k. Details of finding those relationships is given in Algorithm 4.

Algorithm 4: getInvolvingRelationships

Input: Keyword k
Output: Involving relationships of keyword k $involvingRel(k)$
1 $involvingRel(k) = \emptyset$
2 $Obj(k)$ = Getting matching objects of k
3 **foreach** *matching object in* $Obj(k)$ **do**
4 Getting object ID
5 $tempInvolvingRel$ = Getting relationships containing Object ID
6 $involvingRel(k) = involvingRel(k) \bigcup tempInvolvingRel$
7 Removing duplicate elements in $involvingRel(k)$

Implementation of Rule 5 is a combination of those of Rule 2 and 3 with one additional constraint to make sure these three rules are not overlapped.

Algorithm 5: connectingObjects

Input: Query $Q = \{k_1, \ldots, k_n\}$, document D
Output: Connecting objects $connectingObj$
1 **foreach** *keyword* k_i **do**
2 `//getting objects connecting keyword` k_i`, i.e., objects having relationships with objects in` $Obj(k_i)$
3 $connectingObj(k_i)$ = getConnectingObjects(k_i)
4 `//finding common objects connecting all keywords`
5 $connectingObj = \bigcap connectingObj(k_i), i = 1..n$

Algorithm 5 finds objects connecting all keywords which is answers from Rule 3. The implementation of Rule 3 is the extension of that of Rule 2. After getting relationships r which involve keyword k from Rule 2, the system finds the rest of participating objects of r, i.e., excluding the object matching k. These rest objects connect to k. Details of finding these rest objects is given in Algorithm 6.

Algorithm 6: getConnectingObjects

```
Input: Keyword k
Output: Connecting objects of keyword k connectingObj(k)
connectingObj(k) = ∅
Obj(k) = Getting matching objects of k
foreach matching object o in Obj(k) do
    Getting object ID
    tempInvolvingRel = Getting relationships containing Object ID
    foreach relationship r in tempInvolvingRel do
        tempConnectingObj = finding the rest of objects, except object o involving r
        connectingObj(k) = connectingObj(k) ⋃ tempConnectingObj
Removing duplicate elements in connectingObj(k)
```

Let G be the connected, undirected graph covering all objects and relationships of D. To find the minimal tree connecting all keywords, the system first finds sets of objects o which has the biggest number of connecting keywords and creates some trees T connecting o and these keywords. The system then recursive creates trees $tempT$ to connect as many of the rest keywords as possible. $tempT$ is merged to T and the system checks whether T satisfies Rule 6. If not, it keeps creating $tempT$ until there is no keyword left.

Algorithm 7: Find minimal trees

```
Input: Query Q, document D
Output: Result Res(Q, D)
Res(Q, D) = ∅
Tree T is empty
Recursively create trees (Q, T, Res(Q, D))
```

Details of recursively create trees is shown in Algorithm 8.

Ranking. We rank answers based on how close the keywords are. Particularly, the priority of answers is ordered as follows: (1) single object matching all keywords, i.e., answers from Rule 1; (2) relationships in which all keywords involving, i.e., answers from Rule 2, Rule 4 or Rule 5; (3) Objects connecting all keywords, i.e., answers from Rule 3 Answers in this category may still have different ranks, and we rank the common object connecting all keywords with a same relationship type higher than the common object connecting all keywords with different relationship types. (4) Minimal trees connecting all keywords, i.e., answers from Rule 6.

Algorithm 8: Recursively create trees

```
Input: Set of keywords Q
Output: Tree T, Result Res(Q, D)
importantObj = find sets of objects which has the biggest number of connecting keywords
foreach object o ∈ importantObj do
    Create tree tempT containing o
    Kw(o) = set of keywords connecting o
    notyetKw = Q - Kw(o)
    foreach keyword k ∈ Kw(o) do
        Add o_i to tempT. o_i has relationships with o and matches k
    T = merge tempT into T
    if T satisfies conditions in Rule 6 then
        Res(Q, D) = Res(Q, D) ⋃ T
    else
        if notyetKw = ∅ then
            Return;
        else
            Find minimal trees (notyetKw, T, D)
```

4 Experiments

We evaluate the search quality and efficiency of our approach, by comparing it with SLCA, a representative LCA-based approach. We implement the XKSearch [14] to perform SLCA computation. Comparing with XKSearch is enough to show the advantage of our approach as other LCA-based systems suffer the same problems as XKSearch. To study how the problems of the LCA-based approach affect search quality, we make the statistics about the percentage of queries falling into each case discussed in Section 2 and the satisfaction of users for answers of those queries. Two real data sets used in experiments are Basketball_Player[1] (45.2MB) and eBay[2] (0.36MB).

4.1 Search Results

We use some queries to illustrate the differences in search results of our approach and SLCA, as shown in Table 3. Due to space constraint, Table 3 only shows results for a subset of tested queries. Results of 100 other tested queries reveal the same conclusion. Problems in SLCA results for these 100 queries are summarized in Section 4.2.

Since one query may have more than one interpretation, Table 3 shows answers corresponding to each interpretation and problems of these answers. An answer *Ans* is classified into four categories: (1) *Correct*: *Ans* is the expected

[1] http://www.databasebasketball.com/

[2] www.cs.washington.edu/research/xmldatasets/data/auctions/ebay.xml

Table 3. Comparison on search results of our approach and SLCA

	Query	**Matched objects/ relationships**	**Result of our approach (Top 10)**	**problem of our approach**	**SLCA result**	**SLCA Problem**
Q_1	*Celtics*	*Celtics* (136): 1 *Team* object	1 answer: team Celtics	No	136 nodes with their value being *Celtics*.	Incomplete Duplicate
Q_2	*Mark, Steve*	*Mark* (11): 11 *Player* objects; *Steve* (9): 9 *Player* object;	10 answers: each of which contains a *Team* object, together with a *Player* whose *firstName* is *Mark*, and another *Player* whose *firstName* is *Steve*.	No	The document root.	Incorrect
Q_3	*Michael, Smith*	*Michael* (256): 13 *Player* objects.	1 answer: a *Player* object with his *firstName* *Michael* and *lastName* *Smith*.	No	1 subtree rooted at a *Player* whose name is *Michael Smith*, including all *Teams* he has worked for, and all the *Coachs* worked for these teams.	Irrelevant
		Smith (20): 15 *Player* object;	8 answers, containing a *Team* object, together with a *Player* object whose *lastName* is *Smith*, and another *Player* object whose *firstName* is *Michael*.	No	The document root.	Incorrect
		Michael (256): 2 *Coach* objects; *Smith* (20): 15 *Player* object;	1 answer containing *Team* object, together with a *Player* object whose *lastName* is *Smith*, and a *Coach* object whose *firstName* is *Michael*.	No	1 subtree rooted at object *Player* whose *lastName* is *Smith*, including all *Teams* he has worked for, and all *Coachs* of these teams.	Irrelevant
Q_4	*Bill, 1991*	*Bill* (1219): 3 *Player* object; *1991* (5052):129 relationships w.r.t *Player* and *Team*.	3 answer, containing a *Player* whose *firstName* is *Bill* and the *Team* in which he played in *1991*, together with all relationship attributes of the relationship among them.	No	577 subtrees rooted at *Player*, whose *firstName* is *Bill*, including all *Teams* for which he has played, together with all *Coachs* working for these teams.	Irrelevant
		Bill (1219): 9 *Coach* object; *1991* (5052): 28 relationships w.r.t *Coach* and *Team*.	7 answers, containing a *Coach* whose *firstName* is *Bill* and the *Team* in which he worked in *1991*, together with all relationship attributes of the relationship among them.	No	259 subtrees rooted at *Coach* whose *firstName* is *Bill*, together with all other attributes under this *Coach* object.	Incomplete Duplicate

answer; (2) *Irrelevant*: *Ans* contains the expected answer, but also contains a lot of irrelevant information; (3) *Incomplete*: *Ans* only contains part of the expected answer; and (4) *Incorrect*: *Ans* is totally different with the expected answer, or there is a missing answer. Moreover, for each case, answers can be *duplicate*.

Discussion. The answer for Q_1 of SLCA is not only incomplete, but also meaningless as it contains only one node with no other useful information. The irrelevant answers for Q_2, Q_3, and Q_4 of SLCA do overwhelm users with a large amount of information. Q_4 involves relationship attribute value, SLCA returns subtree rooted at either higher object (with a lot of irrelevant information) or lower object (without other participating objects of the relationship, answers are incomplete). Moreover, SLCA returns many duplicate answers (Q_1, Q_4) which are not easy to be filtered. It can be seen that SLCA may return duplicate, irrelevant, incomplete, incorrect answers, while our approach can avoid all of them and return expected answers as discussed in Section 2.

4.2 Problems of LCA-based Approach

We collected 100 keyword queries for both Basketball_Player and eBay data sets from PhD students in computer science. We analysed these keyword queries and classified them into four different cases mentioned in Section 2: Case 1, Case 2A, Case 2B and Case 3. For each case, Fig. 7 shows the percentage of queries, for which answers returned by SLCA fall into four categories *Correct*, *Irrelevant*, *Incomplete* and *Incorrect* as defined as in Section 4.1.

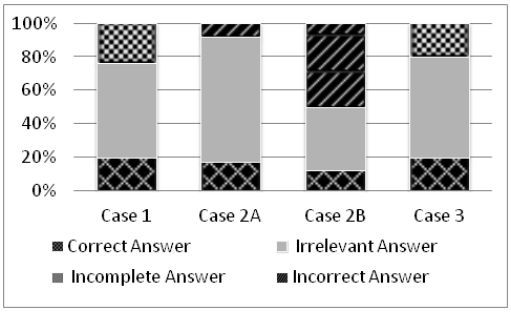


Fig. 7. Problems of LCA

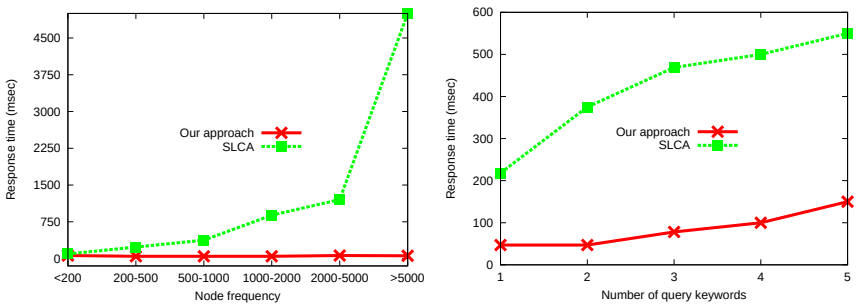
Among these 100 collected queries, 50 queries fall into case 1; 24 queries fall into case 2A, 16 queries fall into case 2B; and 10 queries fall into case 3. It infers that users most frequently ask for information about one object, then ask about several objects having direct relationships. For each of these 100 queries, we give

the answers returned by SLCA to query designers and let them decide which category these answers should be classified. The correctness of answers for 100 collected queries is given in Fig. 7.

Discussion. For all cases, the percentages of correct answers are very low (less than 20%). Most of answers returned by SLCA are irrelevant, especially answers for queries of Case 2A where an object at high level is returned at an SLCA node. SLCA also returns incomplete answers for queries of Case 1 and Case 3. For Case 1, queries containing only one keyword cannot return the whole object. For Case 3, queries containing relationship attribute values may not return all the participating objects of that relationship. Most of incorrect answers are for queries of Case 2B where matching objects do not have direct relationship.

4.3 Efficiency evaluation

We evaluate the efficiency of SLCA and our approach by varying the number of query keywords and node frequency of keyword, i.e., the frequency of a keyword in the XML data. We run data set *Basketball_Player* and use the average time of 10 runs for each query. The result is shown in Fig. 8.



(a) Varying node frequency (b) Varying No. of keywords

Fig. 8. Response time of our approach and SLCA

The response time of varying node frequencies of keywords is plotted in Fig. 8(a). A set of queries with two keywords are randomly chosen. The response time of our approach depends on the frequency of matching objects and relationships rather than the node frequency since it works at object and relationship level instead of node level as SLCA. Thus, our approach runs stably while SLCA response time increases very fast with the node frequency. Moreover, working at object and relationship level enables our approach to run faster than SLCA

since the number of matching objects and relationships needed for processing is less than the number of nodes required by SLCA.

The processing time of varying the number of query keywords from 1 to 5 is given in Fig. 8(b). A set of queries whose keywords with medium node frequency, i.e., from 1000 to 2500, are randomly chosen. The response time of both our approach and SLCA increases almost linearly with the number of keywords. However, our approach runs faster than SLCA since our approach works at object and relationship level as explained.

5 Related work

LCA-based XML keyword search. Most existing XML keyword search methods are LCA-based and depend on hierachical structure of the data. XRANK [5] proposes a stack based algorithm to efficiently compute LCAs, and also presents a ranking method to rank subtrees rooted at LCAs. [15] proposes an Index Stack algorithm to find LCAs more efficiently. XKSearch [14] defines Smallest LCAs (SLCAs) to be the LCAs that do not contain other LCAs, and proposes efficient algorithms to compute SLCAs. Meaningful LCA (MLCA) [10] incorporates SLCA into XQuery. MCTs [6] introduces minimum connecting trees to exclude the subtrees rooted at the LCAs that do not cover query keywords. VLCA [8] introduces the concept of valuable LCA to improve the effectiveness of SLCA. Chen et al. [2] propose join-based algorithms to combine both semantic pruning of SLCA and ELCA to get top-k answers more efficiently. Although researchers have put efforts on improving LCA-based effectiveness, their works are still based on the hierarchical structure and thus face problems as pointed out in Section 2.

Semantics in XML keyword search. XSeek [11] and MaxMatch [12] infer semantics from keyword query. They can only infer semantics of object since it is impossible to infer any semantics of object ID, relationship and relationship attribute from a keyword query. XReal [1] performs XML keyword search on object level, and uses statistics information from XML data to identify *search for node*, which is similar to the object concept in our approach. However, XReal only considers the object in XML data but not object ID and relationship. XKeyword [7] exploits semantics from the XML schema. However, XML schema does not fully contain semantics as most XML schemas do not capture semantics of object ID and relationship. Some works focus on adding semantics into query such as XSEarch [4] but the added semantics is for distinguishing a tag name and a value keyword only. Since the semantics inferred, exploited or added in

above approaches is limited, they cannot answer all queries correctly, especially those related to relationships and relationship attribute values.

6 Conclusion

In this paper, we have systematically illustrated problems of the existing LCA-based XML keyword search approaches when they handle XML data and queries involving relationships. We also showed the impact of ORA-semantics in XML keyword search and proposed to use ORA-semantics together with rules to process XML keyword queries. Our experiments showed that the proposed approach returns correct and meaningful answers for queries involving relationship, which cannot be correctly answered by the LCA-based approach.

References

1. Z. Bao, T. W. Ling, B. Chen, and J. Lu. Efficient XML keyword search with relevance oriented ranking. In *ICDE*, 2009.
2. L. J. Chen and Y. Papakonstantinou. Supporting top-k keyword search in xml databases. In *ICDE*, 2010.
3. S. Chong, C. Chee-Yong, and G. A. K. Multiway SLCA-based keyword search in XML data. In *WWW*, pages 1043–1052, 2007.
4. S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A semantic search engine for XML. In *VLDB*, 2003.
5. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD Conference*, 2003.
6. V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava. Keyword proximity search in XML trees. *IEEE Trans. Knowl. Data Eng.*, 2006.
7. V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *ICDE*, 2003.
8. G. Li, J. Feng, J. Wang, and L. Zhou. Effective keyword search for valuable LCAs over XML documents. In *CIKM*, 2007.
9. L. Li, T. N. Le, T. W. Ling, H. Wu, and S. Bressan. Discovering semantics from XML. *TRA3/12, 2012, technical report, School of Computing, NUS.*
10. Y. Li, C. Yu, and H. V. Jagadish. Schema-free XQuery. In *VLDB*, 2004.
11. Z. Liu and Y. Chen. Identifying meaningful return information for XML keyword search. In *SIGMOD Conference*, 2007.
12. Z. Liu and Y. Chen. Reasoning and identifying relevant matches for XML keyword search. *PVLDB*, 1(1), 2008.
13. A. Schmidt, M. L. Kersten, and M. Windhouwer. Querying xml documents made easy: Nearest concept queries. In *ICDE*, pages 321–329, 2001.
14. Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD*, 2005.
15. Y. Xu and Y. Papakonstantinou. Efficient LCA based keyword search in XML data. In *EDBT*, 2008.