From Revisiting the LCA-based Approach to a New Semantics-based Approach for XML Keyword Search

Thuy Ngoc Le, Huayu Wu, Tok Wang Ling, and Luochen Li
School of Computing
National University of Singapore
{*ltngoc, wuhuayu, lingtw, luochen*}*@comp.nus.edu.sg*

Abstract

Most keyword search approaches for data-centric XML documents are based on the computation of Lowest Common Ancestors (LCA), such as SLCA and MLCA. In this paper, we show that the LCA is not always a correct search model for processing keyword queries over general XML data. In particular, when an XML database contains relationships among objects, which is quite common in practical data, LCA-based search may not be able to find desired answers for many keyword queries. We propose to use semantics instead of the structure of XML data to perform keyword search, and show that the semantics-based search can solve the problems of the LCA-based approach. To the best of our knowledge, this is the first work to point out serious problems of the LCA-based XML keyword search approach, and propose an approach to perform XML keyword search based on semantics rather than the hierarchical structure of XML data to address those problems.

1 Introduction

XML keyword search has been studied for several years. Inspired by the hierarchical structure of XML data, most efforts in keyword search over data-centric XML documents focus on exploring an XML tree to discover the useful information covered by query keywords. The LCA-based approach is a typical XML keyword search approach, which is based on the hierarchical XML structure. It searches for every subtree containing all query keywords and less irrelevant information. Many LCA-based search models have been proposed, such as SLCA [22] and MLCA [17]. However, the LCA-based approach is only correct for a part of practical XML databases and queries. If an XML data contains relationships among objects, the LCA-based approach may be improper for processing many keyword queries.

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, the *take* between *Student* and *Course* and the *use* between *Course* and *Textbook*. Generally, there are several XML designs for this database. In these designs, *Student* and *Course* are usually put into the hierarchical structure with an edge to reflect the relationship between them. Fig. 2 shows two possible designs for this XML data. For simplicity, we only draw object class nodes and relationship attribute nodes in these schema diagrams. The 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.

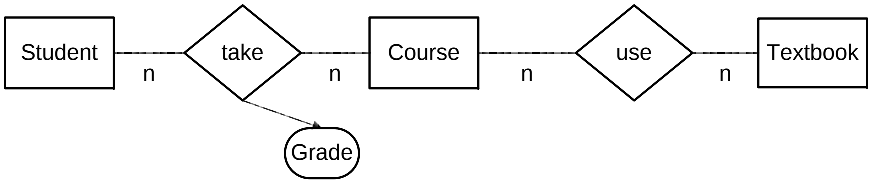
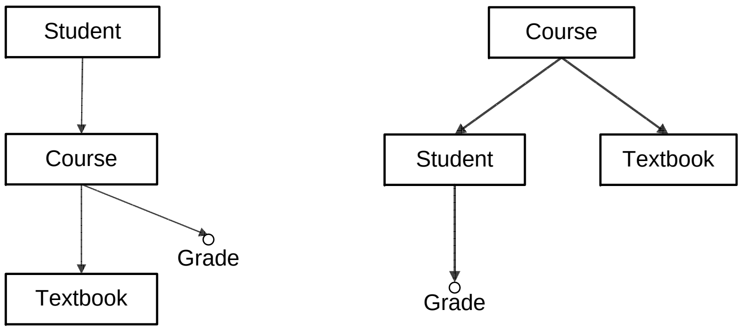


Figure 1: The University database in ER diagram.

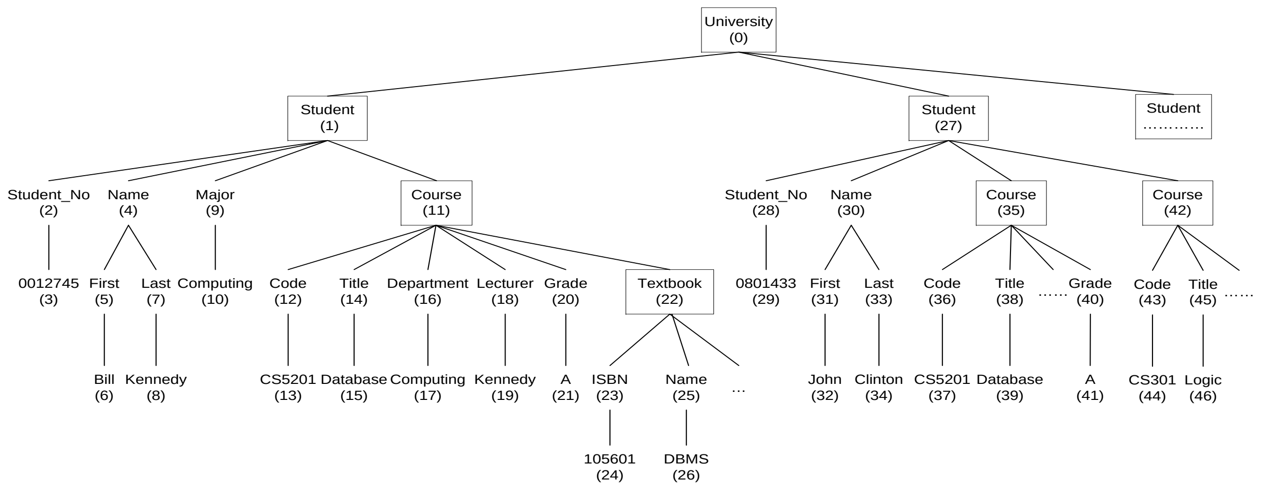


(a) SC-XMLDB schema (b) CS-XMLDB schema

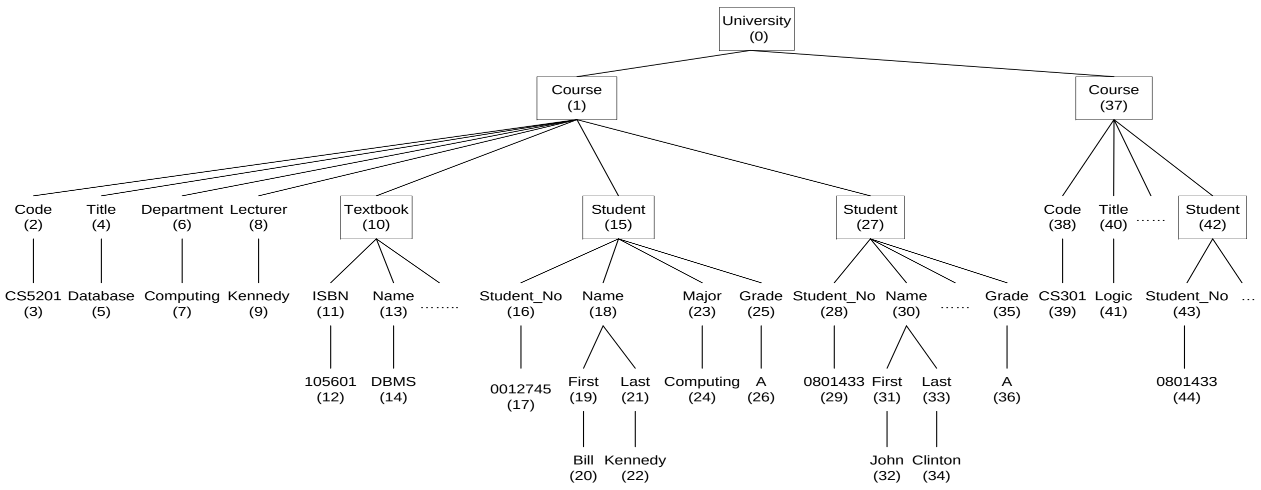
Figure 2: Two possible XML schemas of the University database

Suppose that there are two XML keyword queries, Q_a and Q_b. $Q_a = \{Bill, John\}$ asks for the common information of two students *Bill* and *John*, i.e., the common courses taken by these two students in this document. $Q_b = \{Database, Logic\}$ asks for the common information of two courses *Database* and *Logic*, i.e., the common students taking these two courses. With the SC-XMLDB, the LCA-based approach returns the document root as the answer to Q_a, because only the root is the LCA of the query keywords. Obviously, this answer is not meaningful at all. The LCA-based approach faces this problem because it cannot discover the real relationships between *Course* objects and *Student* objects to find the common courses between two students. It is even worse that under the LCA search model, a user cannot find any keyword query to fulfil the purpose of Q_a. With the CS-XMLDB, the LCA-based approach can answer Q_a. However, it cannot find correct results for Q_b and the user cannot find any query to meet the search intention of Q_b either. More cases where the LCA-based approach fails to work will be discussed in Section 2. It shows that when relationships among objects are considered, the LCA-based approach cannot find correct answers for many keyword queries.

Unfortunately, relationships do frequently exist in practical databases. If we consider any relational database designed based on ER model, there will be many relationship tables, each of which can correspond to a relationship type among several object classes. Generally, in XML data, any two objects connected by an edge may be considered in a certain relationship, with or without relationship attributes under the lowest object of the relationship. For any XML document with a certain depth in structure, there may be many relationships among objects. Furthermore, in recent years, to be exchanged on the Internet, relational data needs to be mapped to XML data [7, 21] which definitely contains relationships between objects in their XML trees.



(a) An instance diagram of the SC-XMLDB



(b) The same data of the CS-XMLDB

Figure 3: The University database

To meet the purpose of Q_a, i.e., finding the common courses taken by two students in the SC-XMLDB, where *Course* is a descendent of *Student* in the XML schema, a user can use XQuery 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 expression, we need to join two paths *\$s1/course/code* and *\$s2/course/code*, based on the same *Code* value. The rationale behind this XQuery is that we have to know the semantics that every course and every student is an object, *Code* is the object ID (identifier) of *Course* and there is a relationship between *Student* and *Course*, in order to compose that XQuery query.

More generally, in XML query processing, we need such semantics to answer query Q_a correctly regardless of query types, e.g., XQuery query or keyword query. It means that either the search engine or the query expression must contain the semantics. In XML keyword search, a keyword query itself does not have the semantics. An end user does not need to know the semantics to issue an XML keyword query. Even if he knows the semantics, he is not able to express it in any XML keyword query. Therefore, in XML keyword search, to answer Q_a correctly, a search engine, whichever approach it applies, must use the semantics of the XML data. This semantics can be provided by the XML data designer or the search engine administrator. Otherwise, i.e., neither the query nor the search engine has the semantics, the Q_a can never be answered correctly since the common courses taken by the two students cannot be identified without knowing the ID attribute of course.

Despite the important role of semantics for XML keyword search, very few systems pay attention to using semantics for XML query processing. The only work we identified that may be considered as a semantics-based approach is XKeyword [12]. It exploits the XML schema to improve search quality. However, most XML schemas cannot fully capture semantics of XML data, especially for relationships and relationship attributes. As a result, it cannot discover real relationships among objects and cannot distinguish relationship attributes and object attributes. For example, in the SC-XMLDB, most schemas cannot tell that *Grade* is an attribute of the relationship between *Student* and *Course*. Thus, when XKeyword processes the query {*Database, A*} to the SC-XMLDB, it only returns the object *Course_(11)*[1], which is not quite meaningful to that query.

In this paper, we propose a new semantics-based approach that uses the semantics of objects, object IDs, relationships among objects and attributes of objects and relationships to process XML keyword queries. In particular, we propose rules and guidelines to identify answers of those queries. To avoid returning meaningless answers, our approach handles problems at the object and relationship level in which each query keyword is associated with an object or a relationship rather than with an attribute in XML data.

In summary, the contributions of our work are as follows.

- We illustrate serious limitations of the existing LCA-based XML keyword search approaches when they handle XML data accompanied with relationships among objects. This finding is important since most current XML keyword search approaches are still LCA-based. To the best of our knowledge, this is the first work pointing out this problem, and showing that the LCA-based search is not a correct search model for XML query processing.
- We propose to use semantics of XML data to process XML keyword queries. In particular, we propose several rules and guidelines to identify the expected answers.
- We design XSem, a semantics-based XML keyword search engine, based on the rules and guidelines we propose, and show the effectiveness of XSem experimentally.

The rest of this paper is organized as follows. In Section 2, we show problems of the current LCA-based approach. To avoid these problems, we propose a new semantics-based approach in

[1] *Course_(11)* refers to the node whose name is *Course* and label is 11. It is similar to other nodes.

Section 3. Section 4 presents XSem, a search engine of our semantics-based XML keyword search approach. Our experimental results are shown in Section 5. Related work is reviewed in Section 6. Finally, conclusion and future work are given in Section 7.

2 Problems of the LCA-based XML keyword search approaches

In an XML instance diagram, a node may correspond to 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)*, *Student_(27)*, *Course_(11)*, *Textbook_(22)* etc. *Major* is an attribute of the object class *Student* and *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 a relationship attribute value.

Most practical XML keyword queries involve objects and/or relationships among objects, and aim to find the common information of those objects and/or relationships. We classify keyword queries into different cases based on how the query keywords match objects and relationships. Particularly, we divide queries into three cases: (1) query keywords matching only one object; (2) query keywords matching more than one object but no relationship attribute value; and (3) some query keywords matching relationship attribute values. In each case, we point out the problems of the LCA-based XML keyword search approach by showing differences between its search result and the expected result for each example query. Although we consider the scenario that each query contains at most two keywords to illustrate the problems, such problems also occur for queries containing more than two keywords. We use the University database introduced in Section 1, i.e., the SC-XMLDB and the CS-XMLDB instance diagrams shown 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

We further classify queries of this case into two subcases. In the first subcase, a query contains only one keyword. In the second subcase, a query contains multiple keywords, but all these keywords center at one object.

2.1.1 Case 1(A): Query containing only one keyword

Example 2.1 *A user wants to find the information of a student whose name is* Bill *by issuing a query* {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, as this node is the LCA of itself. However, returning this node is not useful at all since it does not provide any useful information about *Bill*. Only with more accurate complementary returned information inferences such as in [19, 20], the LCA-based approach may return the correct object to answer the query.

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*.

Conclusion. If a query contains only one keyword, the LCA-based approach returns the node matching this single keyword. This answer is meaningless since it does not provide any other useful information.

2.1.2 Case 1(B): Query containing multiple keywords

Example 2.2 *Consider the following illustrative query* {Kennedy*,* Computing}*.*

LCA-based answer. With the SC-XMLDB, most LCA-based approaches, e.g., SLCA [22] and MLCA [17], return node *Course_(11)*. Although node *Student_(1)* is also an LCA of *Kennedy_(8)* and *Computing_(10)*, it is not returned by SLCA and MLCA, since its descendant node *Course_(11)* makes it unable to be an SLCA of the two keywords. Similarly, with the CS-XMLDB, SLCA and MLCA only return *Student_(15)*, and ignore *Course_(1)*.

Expected answer. For this query, both *Student 0012745* and *Course CS5201* are meaningful answers. In particular, this ambiguous query may search for students named *Kennedy* and majored in *Computing*, or course taught by *Kennedy* and offered by the department of *Computing*.

Conclusion. Most LCA-based extension, e.g., SLCA and MLCA, put constraints to filter LCA answers to make the query processing more effective. However, these constraints may filter out many meaningful answers as well.

2.2 Case 2: Keywords matching more than one object, but no relationship attribute value

We further classify queries of this case into two subcases according to how the objects referred by a query are related to each other. In the first subcase, these objects are directly related to each other through a relationship; while in the second subcase, these objects are not directly related.

2.2.1 Case 2(A): Objects having direct relationships

Example 2.3 *Suppose that there is a query* {Bill*,* Da-tabase} *asking for the relationship between student* Bill *and course* Database*.*

LCA-based answer. With the SC-XMLDB, the LCA-based approach returns object *Student_(1)*. The subtree of 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 of *Course_(1)* is returned. This subtree also contains a large amount of irrelevant information, e.g., all other students taking this course. The irrelevant information makes users difficult to identify essential answers. Filtering such irrelevant information is not trivial. No matter how the database is designed, the LCA-based approach only returns some subtrees containing the two objects, instead of really finding the relationship between them. Moreover, the corresponding results for different schemas, the SC-XMLDB schema and the CS-XMLDB schema, are completely different though these schema designs contain exactly the same information and we process the same query.

Expected answer. The expected answer to this query should be the relationship between student *Bill* and course *Database*. From the ER diagram in Fig. 1, the relationship between these two objects should be student *Bill* taking the course *Database* and getting a grade of *A*. Note that *Grade* is a relationship attribute of the relationship type between *Student* and *Course*.

Conclusion. When a query contains two objects with a direct relationship, the relationship (with relationship attribute values) should be returned as an answer. However, the LCA-based approach cannot return such desired answers precisely since they cannot discover this relationship. Moreover, they heavily depend on the XML hierarchical structure, i.e., returning different answers for different hierarchical designs of XML data.

2.2.2 Case 2(B): Objects having indirect relationships

Some queries refer to objects with no direct relationship but may be related indirectly by some intermediate objects connecting them through more than one relationship. There are two subcases: one refers to objects of the same class and the other refers to objects of different classes.

Case 2(B1): objects belonging to the same class

Example 2.4 *Recall the scenario in Section 1. Consider a query that finds the common information of two students, i.e.,* {Bill, John}.

LCA-based answer. With the SC-XMLDB, the LCA-based approach returns the document root. Definitely it is not meaningful. If the CS-XMLDB is used, the LCA-based approach can find the common courses (if any) taken by these two students. However, if another query finds the common information of two courses, e.g., {*Database*, *Logic*}, the LCA-based approach will also return the document root and the common students taking both courses cannot be found.

Expected answer. The common information of the two students should be the common courses (if any) taken by them, in our data set. No matter how the XML data is designed, the returned answer should always be this common information.

Conclusion. When a query contains two objects from the same class, ideally we need to find the common information of these two objects. However, the correctness of the LCA-based approach to solve this type of query highly depends on the design of XML data and particular queries.

Case 2(B2): objects belonging to different classes

Example 2.5 *The query* {Bill, DBMS} *asks for common information of student* Bill *and textbook* DBMS.

LCA-based answer. With the SC-XMLDB, node *Student_(1)* is returned. The subtree rooted at this node contains a lot of extraneous information such as all courses *Bill* takes and other textbooks. With the CS-XMLDB, the *Course_(1)* is returned. The subtree rooted at this node also contains a lot of unnecessary information, such as all other students taking this course and other textbooks used in this course. Similar to case 2(A), such irrelevant information makes users difficult to determine expected answers and filtering this information is not easy. Besides, it shows again that different schema designs have different answers for the same query, though all these designs contain the same information.

Expected answer. When a user issues such a query, his search target is all common information between student *Bill* and textbook *DBMS*. The expected answer should be object Course *CS5201* which is taken by student *Bill* and uses textbook *DBMS*, i.e., this object has relationships with both student *Bill* and textbook *DBMS*.

Conclusion. Similar to Case 2(B1), for this kind of queries, the common information of the two objects, i.e., the common objects which have relationships with both of the two objects, should be an answer. However, the correctness of the LCA-based approach to process keyword queries in this case depends on the XML hierarchical structure and the queries.

2.3 Case 3: Some keywords matching relationship attribute values

Example 2.6 *Consider a query* {Database*,* A} *asking for students getting* A *grade of course* Database*.*

LCA-based answer. With the SC-XMLDB, the LCA-based approach returns *Course_(11)* and *Course_(35)* to answer this query. Although these two nodes appear at different places in the XML tree, they refer to exactly the same object, which is the course whose Code is *CS5201*. The above answers cannot provide the students getting *A* grade for the course *Database*. Obviously, they are incorrect, because *Grade* is the attribute of the relationship between *Student* and *Course*, rather than the attribute of *Course*. The existing LCA-based approach cannot distinguish between an object attribute and a relationship attribute since they do not capture the semantic meaning of relationships. With the CS-XMLDB, the LCA-based approach also returns the *Course_(1)*. Though the subtree of this node contains the information of students taking this course and getting *A*, it also contains a lot of irrelevant information, such as all other students taking this course. Moreover, if users issue a query {*Bill*, *A*}, then using the CS-XMLDB schema makes the LCA-based approach also face the same problems as the ones it has when using the SC-XMLDB.

Expected answer. The proper answer to this query should be all students taking the *Database* course and getting an *A* grade, as well as the relationship between them and that course. This relationship contains other information (if any), such as their midterm grades.

Conclusion. If a query involves relationship attribute values, the LCA-based approach does not return correct answers since they cannot distinguish whether an attribute value belongs to an object or a relationship. Moreover, their answers highly depend on the hierarchical structure of the underlying database as in above cases.

2.4 Summary

We pointed out serious problems of the existing LCA-based approach by various examples. Although we only analyze binary relationships, in general, the discussed problems also occur in n-ary relationships and recursive relationships. The main reason of these problems is that the LCA-based approach highly relies on the hierarchical structure of an XML data and ignores real semantics of objects and relationships between objects. Therefore, with the same query, they normally return different answers for different schema designs of the same data source. Moreover, in many cases when a query contains several objects involving certain relationships, the LCA-based search cannot find meaningful answers by only considering the hierarchical structure of data. Additionally, the LCA-based approach cannot identify duplicate answers and remove them out of the results. The above observations motivate us to propose a new semantic approach for XML keyword search which is independent of the hierarchical structure of XML data and exploits real semantics of XML data to find answers.

3 Semantics-based approach

We provide an overview of our semantics-based XML keyword search approach in Section 3.1, followed by technique details of identifying the expected answers in Section 3.2 and Section 3.3.

3.1 Overview of our approach

To correctly process an XML keyword query, we are interested in exploiting the semantics of objects, object IDs, relationships among objects, and attributes of those objects and relationships in an XML data. Such semantics can be provided by database designers or administrators. Also there are semantically rich models such as ORA-SS [18], to capture and express semantic information for XML data. As discussed in Section 1, semantics is very important to XML keyword search. Without it, many queries can never be answered correctly.

Our approach works at object and relationship level which means that each query keyword is associated to the corresponding objects and/or relationships. Specifically, a keyword matching an attribute value is considered as matching the objects and/or the relationships to which this attribute value belongs. Similarly, a keyword matching an attribute is considered as matching the object class and/or the relationship type to which this attribute belongs. Additionally, working at object level does not allow duplicate objects, i.e., one object appears as many different nodes. Two different nodes are considered as matching the same object if they belong to the same object class and have the same value of Object ID. Handling problems at object and relationship level has the following advantages. First, it can avoid returning meaningless answers such as a subtree containing only one value node or nodes which are not objects. Second, it can filter duplicate information when the same object appears at different places in XML data. Third, the number of object combinations to be considered is much less than the number of node combinations, when each object appears as different nodes.

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 for any schema designs of the same data. In addition, it can also handle the case where there is more than one relationship type between the same set of object classes. The most important advantage is that our approach can handle all the problems of the LCA-based approach studied in

Section 2. Particularly, it returns the expected answers and does not return meaningless answers.

The process of our semantics-based approach can be briefly described as follows. We first match each query keyword to a set of corresponding objects and/or relationships. We then exploit semantics to find meaningful connections between those objects and/or relationships (if any). In particular, for each object matching some query keywords, we find the relationships (if any) in which this object participates and find the objects (if any) having relationships with this object. Based on these meaningful connections, we finally apply rules and guidelines to identify expected results. Specifically, we classify query keywords into two types (1) keywords as values in leaf nodes, and (2) keywords as tag names. We refer them as value keywords and tag name keywords respectively. Section 3.2 introduces rules to determine expected results. Section 3.3 introduces guidelines to improve efficiency in the case where queries contain tag name keywords.

For ease of presentation, we summarize the concepts and notations used in this paper in Table 1 and Table 2 respectively.

Table 1: Concepts

Concept	**Description**
Keyword k matching object o (or object o matching keyword k)	Keyword k is contained by the object class name, any attribute name or any attribute value of object o.
Keyword k matching a relationship r (or relationship r matching keyword k)	Keyword k is contained by any attribute of relationship type of r or any attribute value of relationship r.
Object o participating in relationship r (or object o being a participating object of relationship r)	r is a relationship among object o and other object(s).
Keyword k matching objects of relationship r (keyword k involving relationship 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.

3.2 Rules to identify answers

In this section, we propose several rules to determine the expected results. 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. We re-use the examples in Section 2 to illustrate our rules to show that our approach can handle the discussed problems of the LCA-based approach. Although only the SC-XMLDB instance diagram is used for illustration, these rules enable us to return the same answers for any schema designs, including the CS-XMLDB. In other words, our rules are independent of the hierarchical structure of XML data.

Table 2: Notations

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

Rule 1 *Given a query $Q = \{k_1, \ldots, k_n\}$ to a document D. 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, then this object is an answer.

Example 3.1 *Consider a query Q_1 =*{Bill}*. This keyword matches object* Student_(1)*. By Rule 1,* {Student_(1)} *is an answer.*

The LCA-based approach returns only node *Bill* while our approach returns object *Student_(1)*. This object provides more useful information about *Bill*. Therefore, it can be seen that returning objects, instead of returning subtree rooted at LCA nodes, can avoid meaningless answers.

Rule 2 *Given a query $Q = \{k_1, \ldots, k_n\}$ to a document D. 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 3.2 *Consider a query Q_2 =* {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.*

The following ER diagram represents a ternary relationship.

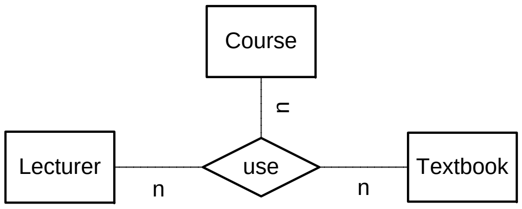


Figure 4: A ternary relationship type in ER diagram.

Example 3.3 *Consider a ternary relationship type shown in Fig. 4 among three classes* Lecturer*,* Course *and* Textbook*. Intuitively, for the same course, different lecturers may use different textbooks. Suppose there is a relationship* r *among* John*,* Database*,* XML *which belong to the classes* Lecturer*,* Course *and* Textbook *respectively. In the query* Q_{2a} = {John, Database, XML} *or* Q_{2b}={John, Database}*, all objects matching the query keywords participate in* r*. Therefore, by Rule 2, relationship* r *is an answer.*

Rule 3 *Given a query* $Q = \{k_1, \ldots, k_n\}$ *to a document* D*. Let* $o_i \in Obj(k_i), i = 1..n$ *and* o *be an object. If* $|\bigcup\{o_i\}| \geq 2$ *and* $R_i(o, o_i)$ *and* $(Class(o_q) = Class(o_p) \implies R_q = R_p, p, q \in [1, n])$*, then* $o \in Res(Q, D)$*.*

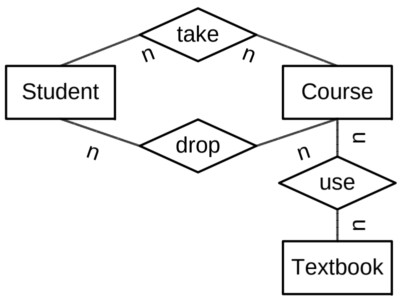
Rule 3 states that if a common object connects all query keywords, then this common object is an answer. Recall that an object o connecting a keyword k means that keyword k matches at least one object which has a relationship with object o. This rule is similar to the LCA-based computation. However, this rule can find the common object o no matter that object is the ancestor or the descendent of objects o_i; while LCA-based approach can only find answers when the common object o are the ancestor of objects o_i.

The condition $(Class(o_q) = Class(o_p) \implies R_q = R_p, p, q \in [1, n])$ is necessary when there is more than one relationship type between two object classes. In this case, if two objects o_q and o_p belong to the same class, then the relationships between the common object o and each of them must belong to the same relationship type. Example 3.6 will illustrate the necessity of this condition. The case where there is only one relationship type between two object classes is illustrated in Example 3.4 where all objects o_i are in the same object class, and in Example 3.5 where objects o_i are in different classes.

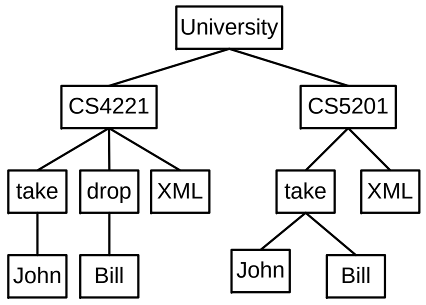
Example 3.4 *Consider a query* Q_{3a} = {Bill, John}*. Keywords* Bill *and* John *match object* Student_(1) *and* Student_(27) *respectively in the SC-XMLDB data. There is an object Course* CS5201 *having relationships of the same type with both of them. Note that this object* Course *appears as two nodes,* Course_(11) *and* Course_(35)*. Though they appear in different places, 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 *is an answer.*

Different from the LCA-based approach, Rule 3 can find the common object *Course CS5201* which has relationships with those two students, even though the former object is the descendent of latter ones. In addition, by returning only object *Course CS5201*, it can filter duplicate information contained by both *node (11)* and *node (35)*.

Example 3.5 *Consider a query* Q_{3b} = {Bill, DBMS}. *Keywords* Bill *and* DBMS *match object* Student_(1) *and* Textbook_(22) *in Fig. 3(a) respectively. There is an object* Course_(11) *having relationships with both of them. Therefore, based on Rule 3,* Course_(11) *is an answer.*



(a) ER diagram



(b) An instance diagram

```
- <University>
  - <Course>
      <Code>CS4221</Code>
    - <Take>
      - <Student>
          <Name>John</Name>
        </Student>
      </Take>
    - <Drop>
      - <Student>
          <Name>Bill</Name>
        </Student>
      </Drop>
    - <Textbook>
        <Name>XML</Name>
      </Textbook>
    </Course>
  - <Course>
      <Code>CS5201</Code>
    - <Take>
      - <Student>
          <Name>John</Name>
        </Student>
      - <Student>
          <Name>Bill</Name>
        </Student>
      </Take>
    - <Textbook>
        <Name>XML</Name>
      </Textbook>
    </Course>
  </University>
```

(c) XML document

Figure 5: Two relationship types between the same set of object classes

Example 3.6 *Consider a scenario where there are two relationship types, namely* take *and* drop*, between two object classes* Student *and* Course *in Fig. 5(a). Suppose there is an example instance diagram in Fig. 5(b) where student* Bill *drops course* CS5201*, student* John *takes course* CS5201 *and both of them take course* CS4221*. For explanation purpose, this instance diagram is at object level. The corresponding XML document is in Fig. 5(c).*

With a query $Q_{3c} = \{$John, Bill$\}$*, there are two common courses, namely* CS5201 *and* CS4221*, having relationships with both of them. However, according to Rule 3, only* $\{$Course CS5201$\}$ *is an answer since* CS4221 *has relationships belong to different relationship types,* take *and* drop*, with* Bill *and* John *respectively. Intuitively, this query searches for the common courses that those two students take or drop together, instead of the courses that one of them takes and the other drops.*

Rule 4 *Given a query* $Q = \{k_1, \ldots, k_n\}$ *to a document* D*. If* $r \in Rel(k_i)$*,* $\forall i = 1..n$*, then* $r \in Res(Q, D)$*.*

Rule 4 states that if attribute values of a relationship matches all query keywords, then that relationship is an answer. Recall that, a relationship matching a keyword means that this keyword matches any attribute value of this relationship. This kind of query is rather rare in practice, as most relationships are issued in queries together with some objects.

Rule 5 *Given a query* Q *to a document* D*.* 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 relationship r, then that relationship is an answer. It means that each keyword either matches an attribute value of that relationship or matches its participating object(s).

Example 3.7 *Consider a query* $Q_5 = \{$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 between the student and the course such as midterm grade.*

The above rules handle general cases of practical queries. 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.*

- $\forall o_i, o_j \in Obj(Q)$, $i \neq j$, *if* $Class(o_i) = Class(o_j)$, *then* $\forall o$ *such that* $R_i(o, o_i)$ *and* $R_j(o, o_j)$, $R_i = R_j$.
- *If any node is omitted out of* T, *then* T *does not satisfy two above conditions.*

Rule 6 implies that if there exist intermediate objects which form a minimal tree connecting all the objects and relationships matching query keywords, then the expected result should include that tree. Note that if there exist two objects o_i, o_j in the same object class, then any object having a relationship with o_i must have a relationship of the same type with o_j.

Fig. 6 and Fig. 7 show the corresponding graph of the University database in Fig. 3, and the data in Fig. 4 respectively.

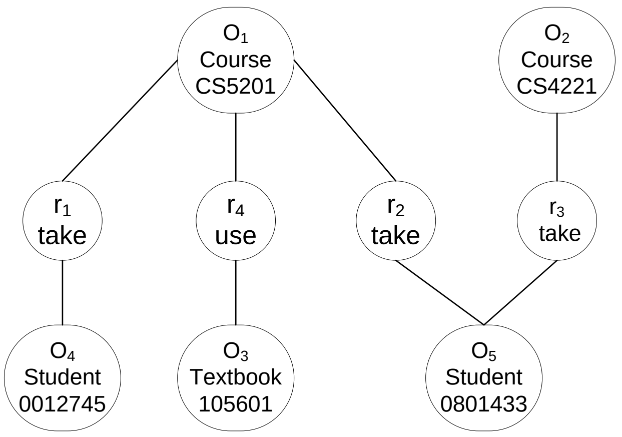


Figure 6: A corresponding graph (partial) of the University database

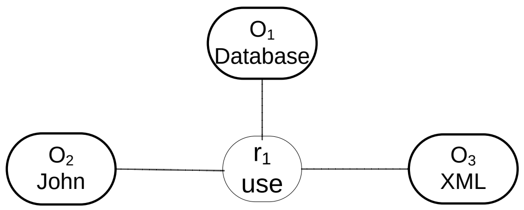


Figure 7: A partial graph with a ternary relationship

3.3 Improving efficiency by handling tag name keywords

Many queries aim to find the object matched by a tag name, instead of values, such as queries {*John*, *student*} and {*John*, *course*} aim to search for student *John* and the courses taken by *John* respectively. Generally, the meaning of a tag name keyword is either a predicate/description name or an output name. Though we can associate a tag name to all objects belonging to the corresponding object class to process a keyword query, we propose two guidelines to handle tag name keywords separately to identify the outputs efficiently.

Section 3.3.1 and Section 3.3.2 consider the case in which a query contains only one tag name keyword and each query keyword can be either a tag name keyword or a value keyword, but not both. Specifically, Section 3.3.1 and Section 3.3.2 handle the case where each value keyword matches at most one object and the case where some value keywords match multiple objects respectively. We then extend the method to handle queries containing multiple tag name keywords or a query keyword can be both a tag name keyword and a value keyword in Section 3.3.3 and 3.3.4 respectively.

Since the query keywords are not order sensitive, for the ease of explanation, we assume that in a keyword query $Q = \{k_1, k_2, \ldots, k_n\}$, k_1 is the tag name keyword with $Obj(k_1) = \{o_1, o_2, \ldots, o_m\}$, and the rest of the keywords $k_i, i = 2..n$ are value keywords. We still use the SC-XMLDB in examples in this section.

3.3.1 Value keywords matching at most one object

Guideline 1 *If some value keyword k_i, $i = 2..n$ matches an object $o \in Obj(k_1)$, then rules in Section 3.2 are applied to the rest value keywords since k_1 can be considered as a predicate or description of k_i.*

Example 3.8 *Consider a query Q =*{student, John} *to the SC-XMLDB. Value keyword* John *matches object* Student_(27)*. Keyword* student *is a tag name keyword, matching all student objects including* Student_(27)*. Therefore, we only match* John *with* Student *object instead of any other object class and get the result* {Student_(27)}*.*

Guideline 2 *If no value keyword k_i, $i = 2..n$ matches any object $o \in Obj(k_1)$, then we imply that k_1 is the output. Therefore, the result should include objects of $Obj(k_1)$ that (1) have direct or indirect relationships with all other objects referred to by value keywords, and (2) participate in all relationships matching value keywords.*

Similar to Rule 3, the following condition is necessary if there is more than one relationship type between $Class(o)$ and $Class(o_i)$, o_i is some object matching some value keyword k_i (i=2..n). If there exists object o_j such that $Class(o_j) = Class(o_i)$, then the relationship between object o and object o_i must belong to the same relationship type with the relationship between object o and object o_j.

Example 3.9 *Consider a query Q =*{Database, student} *to the SC-XMLDB. The value keyword* Database *matches object* Course_(11) *in the SC-XMLDB, and keyword* student *has the same meaning in Example 3.8. Therefore, we can imply that tag name keyword* student *is the output and users aim to search for all students who enroll course* Database*. Since* Student_(1) *and Student (11) has a relationship with Course (11), they are answers.*

3.3.2 Value keywords matching multiple objects

Suppose that $Obj(k_2) = \{o_1^2, o_2^2, \ldots, o_l^2\}, \ldots,$ $Obj(k_n) = \{o_1^n, o_2^n, \ldots, o_m^n\}$. Recall that k_1 is a tag name keyword and $k_i, \forall i = 2..n$ is a value keyword. Let c_p is an object combination of value keywords, $c_p = \left(o^2_{p_2}, \ldots, o^n_{p_n}\right)$ in which $o^i_{p_i} \in Obj(k_i)$, $i = 2..n$, i.e., c_p picks each object $o^i_{p_i} \in Obj(k_i)$. We apply guidelines in Section 3.3.1 for each object combination c_p to get the corresponding result $Res_p(Q, D)$. Then, the final result will be $Res(Q, D) = \bigcup Res_p(Q, D)$.

Example 3.10 *Consider a query Q =*{Kennedy*,* student} *to the SC-XMLDB in Fig. 3(a). Keyword* student *is a tag name keyword, matching all student objects including* Student_(1)*. The ambiguous value keyword* Kennedy *matches two objects* Student_(1) *and* Course_(11) *in the SC-XMLDB.*

When value keyword Kennedy *matches object* Student_(1)*, we can say that tag name keyword* student *is just a description of value keywords* Kennedy*. Then, we apply Guideline 1 and get the result $Res_1(Q, D) =$* {Student_(1)}*. It is aimed to search for a student whose name is* Kennedy*.*

When Kennedy *matches object* Course_(11)*, it does not match any student object. Therefore, by applying Guideline 2, we can imply that the user aims to search for all students taught by* Kennedy*. Then, $Res_2(Q, D) =$* {Course_(11)}*. Finally, $Res(Q, D) = Res_1(Q, D) \bigcup Res_2(Q, D) =$* {Student_(1)*,* Course_(11)}*.*

3.3.3 Queries containing multiple tag name keywords

In this section, we consider the case where a query contains keywords matching multiple tag names in the document. First, Guideline 1 is applied to those tag name keywords, to filter the keywords that are for description purpose. Next, for each remaining tag name keyword, we recursively check whether it satisfies Guideline 2. If so, the object it matches will be considered as a part of returned information.

Example 3.11 *Consider a keyword query Q =*{Bill*,* John*,* student*,* course*,* textbook} *to the SC-XMLDB. Value keywords* Bill *and* John *match two objects of Student and no value keyword matches tag name keyword* course *or* textbook}*. Therefore, by Guideline 1, tag name keyword* student *is a description and by Guideline 2, tag name keywords* course *and* textbook} *are outputs. In particular, this query aims to search for the common courses taken by both students* Bill *and* John *and all the textbooks used in those courses.*

3.3.4 Keywords which are both tag name keywords and value keywords

Suppose that $Obj(k_2) = \{o^2_1, o^2_2, \ldots, o^2_l\}, \ldots, Obj(k_n) = \{o^n_1, o^n_2, \ldots, o^n_m\}$, and $Obj(k_1) = Val\_Obj(k_1) \bigcup Tag\_Obj(k_1)$ where $Val\_Obj(k_1)$ and $Tag\_Obj(k_1)$ are the sets of objects match keyword k_1 when k_1 is a value keyword and is a tag name keyword respectively. Firstly, to each combination of $Val\_Obj(k_1)$, $Obj(k_2), \ldots,$ $Obj(k_n)$, the rules as described in Section 3.2 are applied to get some answers $Res_a(Q, D)$. After handling $Val\_Obj(k_1)$, k_1 can be considered as a tag name keyword only. Thus, guidelines in Section 3.3.1, 3.3.2, 3.3.3 will be applied for $Obj(k_2), \ldots,$ $Obj(k_n)$ to identify some answers $Res_b(Q, D)$. Finally, $Res(Q, D) = Res_a(Q, D) \bigcup Res_b(Q, D)$.

Example 3.12 *Consider a keyword query $Q = \{k_1, k_2\} =$* {Bill*,* John} *to an XML fragment in Fig. 8, in which keyword k_1 =* Bill *can be both a tag name keyword and a value keyword. Then, $Val\_Obj(k1) =$* {Student_(1)} *and $Tag\_Obj(k1) =$* {Bill_(3) *and* Bill_(4)}*, $Obj(k_2) = Obj($*John*$) =$*

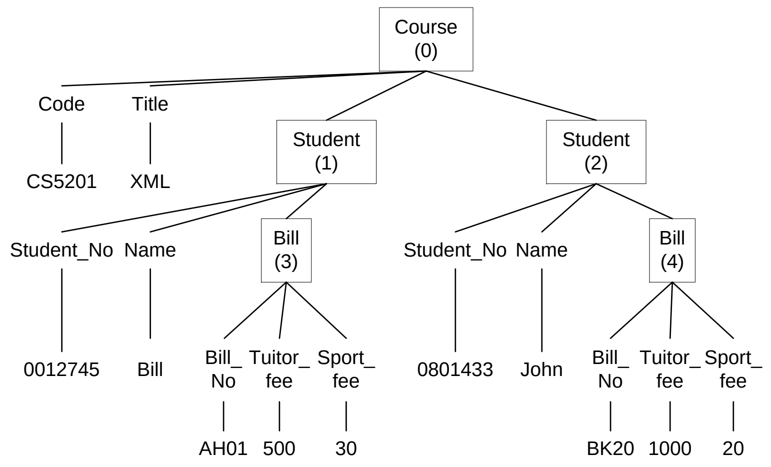


Figure 8: An XML fragment with ambiguous labels.

{Student_(2)}. *When* Bill *is a value keyword, applying Rule 3 in Section 3.2 to* $Val\_Obj(k1)$ *and* $Obj(k_2)$*, the corresponding result is* $Res_a(Q, D)$ = {Course_(0)}. *Intuitively, this query searches for common courses taken by both* Bill *and* John. *When* Bill *is a tag name keyword, on applying Guideline 2 in Section 3.3.1,* Bill *is the output and the corresponding result is* $Res_b(Q, D)$ = {Bill_(4)}. *Intuitively, the bill of student* John *is returned as an answer. Thus, the final result is* {Course_(0), Bill_(4)}.

4 XSem search engine

This section presents XSem, a search engine to implement our semantics-based approach for XML keyword search studied in Section 3.

Using Dewey labels for a XML tree can find answers for some queries but cannot find answers for some other queries. Recall the scenario in Section 1 where there are symmetric queries Q_a and Q_b, with the SC-XMLDB, using Dewey labels, a system for XML keyword search can answer Q_a but not Q_b. Vice versa, with the CS-XMLDB, it can answer Q_b but not Q_a. Thus, XSem does not use Dewey labels. Generally, XSem does not rely on the hierarchical structure of an XML data to process keyword queries. As a result, it makes keyword search independent of XML designs, i.e., having the same result for the same query with any hierarchical structures.

XSem translates an XML document into object relational (OR) tables, and then indexes them. It does not store data in relational tables because relational database has less semantics. In relational database, it is difficult to capture objects and relationships having multi-valued attributes. An object with multi-valued attributes is usually stored in several tables and primary keys are not always object IDs. These may make the searching by joining relational tables based on primary-foreign key constraints more complex and difficult or give meaningless answers.

4.1 Data storage and indexing

XSem captures semantic information, i.e., objects, object IDs, relationships among objects, object attributes and relationship attributes of an XML document, then stores them into OR tables. Specifically, 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.

For example, in the University database in Fig. 3, there are three object classes *Student*, *Course*, *Textbook* and two relationship types, the *take* between *Student* and *Course* with relationship attribute *Grade*, and the *use* between *Textbook* and *Course*. Suppose that there is a multi-valued attribute *Author* of object class *Textbook*. Those objects classes and relationship types are stored in the object tables and relationship tables whose schemas are as follows.

Student(Student_No, First-Name, Last_Name, Major)
Course(Code, Title, Department, Lecturer)
Textbook(ISBN, Name, {Author})
take(Student_No, Code, Grade)
use(Code, ISBN)

It shows that although *Textbook* has a multi-valued attribute *Author*, it can be stored in a single object table whose primary key *ISBN* is also its object ID. Relational database are not able to capture such semantics in one object table. The example object and relationship tables are shown in Table 3-7.

Table 3: Object table *Student*

Student_No	**First_Name**	**Last_Name**	**Major**
0012745	Bill	Kennedy	Computing
0801433	John	Clinton	....
....	....	....	....

Table 4: Object table *Course*

Code	**Title**	**Department**	**Lecturer**
CS5201	Database	Computing	Kennedy
....	....	....	....

Table 5: Object table *Textbook*

ISBN	**Name**	**{Author}**
105601	DBMS	Standley White Richard Smith
....	....	....

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

Table 6: Relationship type *take*

Student_No	Code	Grade
0012745	CS5201	A
0801433	CS5201	A
....	....	....

Table 7: Relationship type *use*

Code	ISBN
CS5201	105601
....	....

4.2 Query processing

This section introduces how XSem processes an XML keyword query. This process 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 and guidelines in Section 3. XSem finds all answers on applying all rules consecutively instead of applying rules for each object combination. This makes the process more efficient, especially when the number of objects matching each query keyword is large.

Algorithm 1: XSem query processing

```
Input: Query $Q$, document $D$
Output: Result $Res(Q, D)$
$Res(Q, D) = \emptyset$
$tagList$ = get tag name keywords in $Q$
if $tagList \neq \emptyset$ then
    Handle tag name keywords ($Q$, $D$, $tagList$) //Algo.2
else
    //find objects matching all keywords
    $matchingObj$ = matchingObjects($Q, D$)  //Algo.3
    $Res(Q, D) = Res(Q, D) \bigcup commonObj$ //Rule 1
    //find relationships which match and/or involve all keywords
    $relatedRel$ = relatedRelationships($Q, D$) //Algo.4, 5
    $Res(Q, D) = Res(Q, D) \bigcup relatedRel$ //Apply Rule 2, 4, 5
    //find common objects connecting all keywords
    $connectingObj$ = connectingObjects($Q, D$) //Algo.6, 7
    $Res(Q, D) = Res(Q, D) \bigcup connectingObj$ //Rule 3
    if $Res(Q, D) = \emptyset$ then
        //find minimal trees connecting all keywords
        $Res(Q, D)$ = find minimal Trees($Q, D$) //Algo.8, 9
        $Res(Q, D) = Res(Q, D) \bigcup subTrees$ //Apply Rule 6
```

In Algorithm 1, XSem first uses guidelines to handle tag name keywords (if any). If there is no tag name keyword, it uses 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, XSem will find minimal trees which connects all keywords. Details of above functions are given in Algorithm 2-9.

```
Algorithm 2: handling tag name keyword
   Input: Query $Q = \{k_1, \ldots, k_n\}$, document $D$,
          tag name keyword list $tagList$
   Output: Result $Res(Q, D)$
1  $Res(Q, D) = \emptyset$
2  $allComb$ = get all object combinations ($Q$)
3  foreach object combination $comb \in allComb$ do
4      $predicate$ = get predicate tag name keywords of $tagList$
5      $output$ = get output tag name keywords of $tagList$
6      $Q = Q - predicate$
7      if $output = \emptyset$ then
8          $tempRes$ = Apply rules to $Q$ //Guideline 1
9          $Res(Q, D) = Res(Q, D) \bigcup tempRes$
10     else
11         $tempRes$ = Apply Rule 3
12         $tempRes$ = filter answers ($tempRes$) //Guideline 2
13         $Res(Q, D) = Res(Q, D) \bigcup tempRes$
```

Algorithm 2 processes an XML keyword query containing tag name keywords. XSem will process each object combination of the query. Each of tag name keywords is checked to see whether it is a predicate/description or an output tag name keyword. XSem then filters answers which do not match any output tag name keywords. Only Rule 3 is applied since the answers must be objects which can connect all the value keywords.

```
Algorithm 3: matchingObjects
   Input: Query $Q = \{k_1, \ldots, k_n\}$, document $D$
   Output: Set of objects $matchingObj$ matching all keywords
1  foreach keyword $k_i$ do
2      $Obj(k_i)$ = get objects matching $k_i$ //retrieve $B^+$ tree
3  //find objects matching all keywords
4  $matchingObj = \bigcap(Obj(k_i)), i = 1..n$
```

Algorithm 3 finds objects which match all keywords which are answers from Rule 1. Implementation of Rule 1 is straightforward. For each keyword, XSem 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 4 find relationships matching and/or involving all keywords. Implementation of Rule 4 is straightforward and similar to that of Rule 1. For each keyword, XSem 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, XSem 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

Algorithm 4: relatedRelationships

```
Input: Query $Q = \{k_1, \ldots, k_n\}$, document $D$
Output: Set of relationships $relatedRel$
1  $relatedRel = \emptyset$
2  foreach $keyword\ k_i$ do
3      $matchingRel(k_i)$ = get relationships matching $k_i$ //retrieve $B^+$ tree
4      //get relationships whose objects matching keyword $k_i$
5      $involvingRel(k_i)$ = getInvolvingRelationships($Obj(k_i)$)
6      $referredRel(k_i)$ = get relationships matching and/or involving keywords $k_i$
7  //find relationships matching all keywords
8  $commonRel = \bigcap(matchingRel(k_i)), i = 1..n$
9  $relatedRel = relatedRel \bigcup commonRel$ //Rule 4
10 //find relationships involving all keywords
11 $involvingRel = \bigcap(involvingRel(k_i)), i = 1..n$
12 $relatedRel = relatedRel \bigcup involvingRel$ //Rule 2
13 //find relationships matching or involving all keywords
14 $referredRel = \bigcap(referredRel(k_i)), i = 1..n$
15 $relatedRel = relatedRel \bigcup referredRel$ //Rule 5
```

given in Algorithm 5.

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: 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 $involvingRel(k)$
```

Algorithm 6: 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 6 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, XSem 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 7.

Algorithm 7: getConnectingObjects

```
Input: Keyword $k$
Output: Connecting objects of keyword k $connectingObj(k)$
1 $connectingObj(k) = \emptyset$
2 $Obj(k)$ = Getting matching objects of $k$
3 foreach matching object $o$ in $Obj(k)$ do
4     Getting object ID
5     $tempInvolvingRel$ = Getting relationships containing Object ID
6     foreach relationship $r$ in $tempInvolvingRel$ do
7         $tempConnectingObj$ = finding the rest of objects, except object $o$ involving $r$
8         $connectingObj(k) = connectingObj(k) \bigcup tempConnectingObj$
9 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, XSem first finds sets of objects o which has the biggest number of connecting keywords and creates some trees T connecting o and these keywords. XSem then recursive creates trees *tempT* to connect as many of the rest keywords as possible. *tempT* is merged to T and XSem checks whether T satisfies Rule 6. If not, XSem keeps creating *tempT* until there is no keyword left.

Algorithm 8: Find minimal trees

```
Input: Query $Q$, document $D$
Output: Result $Res(Q, D)$
1 $Res(Q, D) = \emptyset$
2 Tree $T$ is empty
3 Recursively create trees ($Q$, $T$, $Res(Q, D)$)
```

Algorithm 9: Recursively create trees

```
Input: Set of keywords $Q$
Output: Tree $T$, Result $Res(Q, D)$
1 $importantObj$ = find sets of objects which has the biggest number of connecting keywords
2 foreach object $o \in importantObj$ do
3     Create tree $tempT$ containing $o$
4     $Kw(o)$ = set of keywords connecting $o$
5     $notyetKw = Q - Kw(o)$
6     foreach keyword $k \in Kw(o)$ do
7         Add $o_i$ to $tempT$. $o_i$ has relationships with $o$ and matches $k$
8     $T$ = merge $tempT$ into $T$
9     if $T$ satisfies conditions in Rule 6 then
10        $Res(Q, D) = Res(Q, D) \bigcup T$
11    else
12        if $notyetKw = \emptyset$ then
13            Return;
14        else
15            Find minimal trees ($notyetKw$, $T$, $D$)
```

Ranking. Since the results may contain a lot of answers, XSem ranks these answers based on the similarity between the query keywords. The ranking in XSem is similar to XSEarch [6]. In particular, the priority of results is ordered as follows: (1) objects matching all keywords, i.e., answers of Rule 1; (2) relationships matching and/or involving in all keywords, i.e., answers of Rule 4, Rule 2 and Rule 5; (3) objects connecting all keywords, i.e., answers of Rule 3; and (4) minimal trees connecting all keywords, i.e., answers from Rule 6. XSem does not focus on result ranking currently. We leave effective ranking methods as our future work.

5 Experiments

In this section, we evaluate the quality of results and running time of our XSem, by comparing with the LCA-based approach. We use the SLCA, the most popular LCA-based approach as a representative of LCA-based approach, and take the XKSearch [22] as the implementation. SLCA returns subtrees rooted at some node while XSem returns an object or a relationship. Recall that in XSem, returning an object means returning all attribute values of that object, and returning a relationship means returning all attribute values of that relationship as well as all participating objects.

We use two real data sets: Basketball_Player[2] and eBay[3]. In the Basketball_Player data set, we filter out old information with the year earlier than 1990. A 28 MB Basketball_Player XML document used in our experiment is available at [1]. The experiments were performed on a Intel(R) Core(TM)2 Duo CPU 2.33GHz with 3.25GB of RAM. A set of queries covering all cases discussed in Section 3 are tested for each data set. We present the result of running queries for the Basketball_Player data set and the eBay dat aset in Table 8 and Table 9 respectively.

5.1 Effectiveness evaluation

5.1.1 Results for Basketball_Player data set

To make readers well understand the result, we show the schema diagram of the Basketball_Player data in Fig. 9. From this schema, we can easily figure out the following semantics of the data. There are three object classes *Player*, *Team* and *Course* with *ilkID*, *team* and *coachID* as their ID attributes respectively. There are two many-to-many relationship types. The one between *Player* and *Team* has relationship attributes *year*, *gp*, etc. The other between *Team* and *Course* has relationship attributes *year*, *year_order*, etc.

For this data set, we compare the results of different queries covering all cases mentioned in Section 1 between XSem and SLCA, as shown in Table 8. *Matching objects or relationship* column shows the matching objects and/or relationships of each query keywords. *Node occurrences* column shows the number of nodes matching each query keywords. It can be seen that the number of matching objects and/or relationships is much less than number of matching node to be processed. Moreover, SLCA contains duplicate answers Q_1, Q_4, Q'_4, Q_5), irrelevant answers (Q_3, Q'_3, Q_4, Q'_4) and meaningless answers (Q_1, Q_2, Q'_2, Q''_2, Q_3, Q'_3). XSem can avoid all of them. Most important thing is XSem can return expected answers as discussed in Section 2 while SLCA sometimes cannot.

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

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

Table 8: Results for Basketball_Player data set

	Query	**Matching objects or relationships**	**XSem result**	**Node occurences**	**SLCA-based approach result**	**Discussion**
Q_1	Celtics	Celtics: 1 team (Celtics matches 1 team object)	1 answer: team Celtics	Celtics: 136	136 answers: 136 nodes with value Celtics (team Celtics appreas as 136 nodes corresponding to 136 players working for team Celtics)	SLCA-based approach returns a meaningless answer, i.e., the node with value *Celtics* with no other information, while XSem returns a team object which provides more useful relevant information about *Celtics*. Moreover, SLCA-based approach has many duplicated answers while XSem can filter them.
Q_2	Michael Thomas Brown	Michael: 2 coaches and 13 players;	17 teams hiring players whose name are Michael, Thomas and Brown	Michael: 265;		Matching objects of these queries are in the same class, i.e., class *Player*. XSem returns common objects, i.e., teams related to all of them;
Q'_2	Michael Thomas	Thomas: 15 players; Brown: 4 coaches, 17 players	20 teams hiring players whose name are Michael and Thomas	Thomas: 15; Brown: 2900	The document root	while SLCA-based approach gives a meaningless answer, the root, since the common ancestor of two players is only the root.
Q_3	Johnson Edwards	Johnson: 5 coaches, 14 players; Edwards: 4 players	2 answers: team Hawks and Pacers (common teams of two players Johnson and Edwards)	Johnson: 19; Edwards: 4	4 answers: players (whose common name is Edwards, working with the coaches whose name is Johnson). The root is answer for combinations of 2 players	If combination of objects in the same class, SLCA-based approach faces the same problem with Q_2, Q'_2. If they are in different classes, XSem can find common objects having relationships with both of them,
Q'_3	Brown Thomas	Brown: 4 coaches, 17 players; Thomas: 15 players	22 teams, 2 teams (Sun and Knicks) between a coach and a player, 20 other teams between 2 players	Brown: 2900; Thomas: 15	45 answers: 45 players (whose name is Thomas and work with the coach whose name is Brown). The root is answer for combinations of 2 players.	i.e., team; while SLCA-based approach returns either object, the one in higher level in the XML tree. This makes answers contain a lot of irrelevant information.
Q_4	Bickerstaff 1996	Bickerstaff: 1 coach; 1996: year, attribute of 378 relationships	2 answers: 2 teams Nuggets and Bullets (teams trained by coach Bickerstaff in 1996)	Bickerstaff: 830; 1996: 5750	700 answers: 700 coach nodes (a lot of duplicate information)	Q_4 and Q'_4 are about one object and one relationship attribute, XSem return the relationship which all objects and relationships attribute involve in; while
Q'_4	Buechler 1991	Buechler: 1 player; 1991: year, attribute of 157 relationships	3 answers: 3 teams Nuggets, Bullets, and Bickersta (player Buechler working for these teams in 1996)	Buechler: 1; 1991: 5052	1 answers: player Buechler (including a mount of irrelevant information)	SLCA-based approach cannot identify such relationship correctly and only return objects with contain a lot of irrelevant and/or duplicate information.
Q_5	Celtics team	Celtics: 1 team; team: tag name keyword	1 answer: team Celtics	Celtics: 136; team: 3676	136 answers: 136 team node containing team name Celtics. (duplicate information)	XSem and SLCA-based approach approach give the same results for this query but SLCA-based approach contains duplicate information.

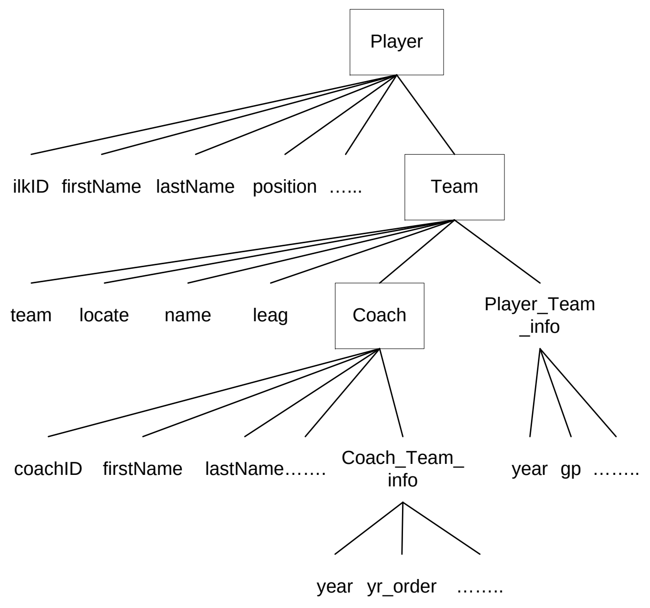


Figure 9: The schema of Basketball dataset.

5.1.2 Results for eBay data set

For explanation purpose, the matching tag name (attribute) for value keywords are shown in column *Matching attributes* instead of matching objects or relationships as when discussing results for the Basketball_Player dataset.

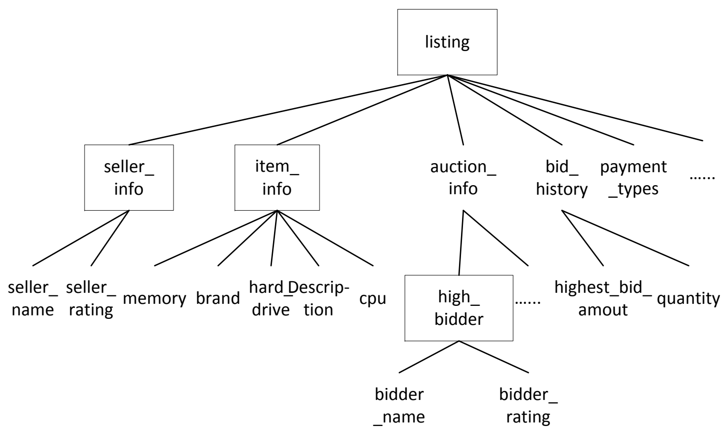


Figure 10: The schema of eBay dataset.

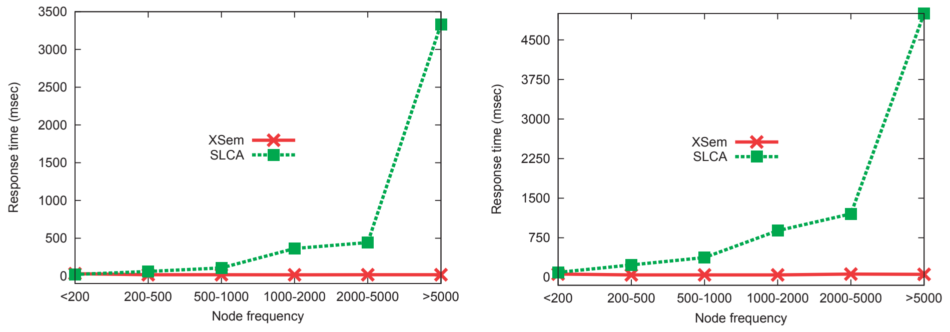
The semantic information supposed to be available for this data set is as follows. There are three object classes, *seller_info*, *high_bidder* and *item_info* whose ID attributes are *seller_name*, *bidder_name* and *itemID* respectively. There is a ternary relationship type *listing* among three object classes *seller_ info*, *high_ bidder* and *item_info*. This relationship type has several attributes, including the *highest_bid_amout* which is the highest price given by a bidder for a particular item in an auction.

Table 9: Results for the eBay data set

	Query	**Matching attributes**	**XSem result**	**Node occurences**	**SLCA result**	**Discussion**
Q_1	ct-inc	ct-inc: seller_name	All information about the object whose seller_name is *ct-inc*.	ct-inc: 1	Just a node whose value is *ct-inc*	SLCA only returns one node with value *ct-inc* with no other useful information while Sem-XSearch can provide all object information of this seller.
Q_2	petitjc, hsclm9	petitjc: bidder_name; hsclm9: bidder_name	The seller ct-inc (the common seller who selling items for both bidders)	petitjc: 1; hsclm9: 1	The document root	Q_2 contains the names of two bidders, XSem returns the common seller who sells items to both of them; while SLCA returns the document root which is meaningless since they cannot find any common information between these bidders.
Q_3	cubsfantony, gosha555	cubsfantony: seller_name; gosha555: bidder_name	The item *I01* (sold from the seller *cubsfantony* to the buyer *gosha555*), and all attribute values of the relationship among them, such as highest_bid_amount	cubsfantony: 1; gosha555: 1	One *listing* node (besides information of XSem, it also includes unnecessary information about the seller, bidder and item)	Q_3 is about two objects, a seller and a bidder, having a relationship, XSem returns the rest participating objects, the item, as well as the relationship between these objects. SLCA can also return the relationship since it is putted as a sub-root.
Q_4	wizbang4, \$610.00	wizbang4: bidder_name; \$610.00: highest_bid_amount	The seller *ct-inc* and the item *I04* (item *I04* are sold from the seller *ct-inc* to the buyer *wizbang4* with the price \$610.00) as well as other information of the relationship among them	wizbang4: 1; \$610.00: 1	One *listing* node (information of the relationship between the bidder wizbang4, the seller ct-inc and the item I04, and information about all these nodes)	Q_4 is related to one object, the bidder, and one relationship attribute, the price, XSem can return the rest objects participating in the relationship, the seller and the item, as well as attributes of relationships these objects participates in. SLCA can also return the relationship since it is put at the higher level of participating objects.
Q_5	highest-bid_amout, \$680.00	highest-bid_amout: tag name; \$680.00: highest-bid_amout	Information of the relationship among seller *ct-inc*, buyer *petitjc@yahoo.com* and item *I02* (its attribute *highest_bid_amount* has value \$680.00)	highest-bid_amout: 1; \$680.00:1	An internal node *highest_bid_amout*, no any extra information besides the keyword	Q_5 is about a value keyword and a predicate tag name keyword. This value keyword refers to a relationship. XSem can return this relationship while SLCA return only one node matches with this value with no other useful information.
Q_6	256MB PC-133 SDRAM, seller	256MB PC-133 SDRAM: memory; seller:tag name	The sellers whose name is *cubsfantony*, *ct-inc* (sellers who sells memory 256MB PC-133)	256MB PC-133 SDRAM: 1; seller:1	One *listing* node (relationship contains this kind of memory, including other unnecessary attributes, e.g., bidder information and auction information.)	Q_6 is related to a output tag name and a value keyword. After using Guideline 2 to handle this tag name, XSem can return the expected answers which are sellers who sell that kind of memory; while SLCA approach returns relationship *listing*.

5.2 Efficiency evaluation

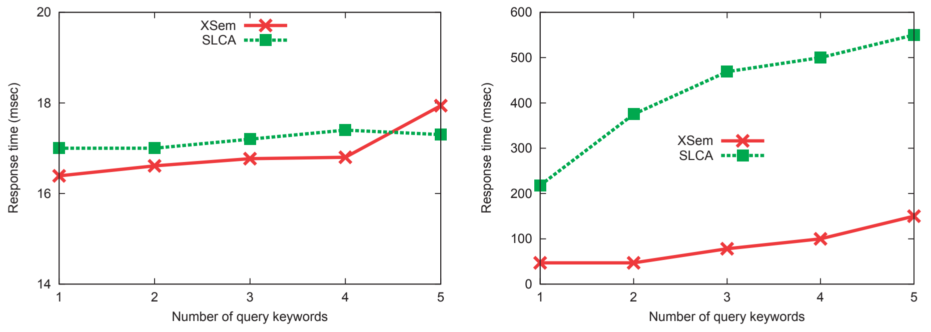
We evaluate efficiency of SLCA and XSem by varying the number of query keywords and node frequency of keywords. We run data set *Basketball_Player* on both hot cache and cold cache and use the average time of 10 runs for each query. The response time of varying node frequencies of query keywords and varying the number of query keywords are shown in Table 11 and 12 respectively.



(a) hot cache (b) Cold cache

Figure 11: Varying the node frequency of keywords

The response time of varying node frequencies of query keywords is given in Fig. 11. A set of queries with two keywords are randomly chosen. The response time of XSem 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. Therefore, XSem runs stable while SLCA response time increases very fast as the node frequency increases. Moreover, working at object and relationship level makes XSem run much faster than SLCA because the number of matching objects and relationships to be processed is much less than the number of nodes to be processed. Additionally, handling tag name keywords separately makes XSem process queries involving tag names much more efficient since most tag name keywords have very high frequency, especially those appearing at low levels.



(a) hot cache (b) Cold cache

Figure 12: Varying the number of keywords

The processing time of varying the number of query keywords from 1 to 5 is given in Fig. 12. A set of queries whose keywords with medium node frequency, i.e., from 1000 to 2500, are randomly chosen. With cold cache, response time of both XSem and SLCA increases almost linear with the increasing of the number of keywords. However, XSem run faster than SLCA since XSem works at object and relationship level as stated.

6 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 [8] proposes a stack based algorithm to efficiently compute LCAs, and also presents a ranking method to rank subtrees rooted at LCAs. XKSearch [22] defines Smallest LCAs (SLCAs) to be the LCAs that do not contain other LCAs, and proposes efficient algorithms to compute SLCAs, such as indexed lookup and scan eager. Meaningful LCA (MLCA) [17] incorporates SLCA into XQuery. MCTs [10] introduces MCTs (minimum connecting trees) to exclude the subtrees rooted at the LCAs that do not cover query keywords. VLCA [15] introduces the concept of valuable LCA to improve the effectiveness of SLCA.

XSeek [19] and MaxMatch [20] infer semantics from the query. They may only infer semantics of objects since it is impossible to infer any semantics of object ID, relationship and relationship attribute from a keyword query. XSEarch [6] adds more information into query by some format such as *l:k*, *l:*, *:k* where *l* is a label and *k* is a keyword. This additional information does not contain any concepts of object, object ID, relationship and attribute to handle queries of all cases correctly. Although researchers have put efforts on improving LCA-based effectiveness, their works are still based on the hierarchical structure and face serious problems and faults as pointed out in Section 2.

Relation-based XML keyword search. The most related work to XSem is XKeyword [12] in which an XML document is stored in a relational database. XKeyword views an XML document as a graph of nodes. The result of a keyword query is the minimal total target object networks which are the minimal graphs involving all query keywords and in which each node is a target object. It exploits the properties of the schema of the database to facilitate the result presentation, to find target objects and to optimize the performance of the search system, e.g., reducing search space. By using XML schema, it can avoid some (yet not all) problems of the LCA-based approach. It can filter duplicate answers and does not return meaningless answers by using target objects as nodes of the returned networks. The most significant difference between XSem and XKeyword is that although XKeyword has connection relationships, it does not discover real relationships among objects and does not distinguish relationship attributes and object attributes as XSem does as discussed in Section 1.

Graph-based XML keyword search. There are several other works on keyword search in XML graph databases. A bi-directional expansion heuristic algorithms to search as small portion of graph as possible is discussed in Bidirectional [13]. BLINKS [9] suggests a bi-level index to prune and increase speed of searching for top-k results. EASE [16] introduces a unified graph index to handle keyword search on heterogeneous data. More recent work has focused on efficiency issues [4, 14]. [14] also focuses on ranking answers of keyword search in databases over a graph, in which a database can be an XML document or a relational database. Some of these work consider the schematic information to reduce search space in graph, but they still did not use real semantics, i.e., object, object ID, relationship, object attribute and relationship attribute, to infer search result. Thus, they may not have enough capability to capture the semantic information and may produce less relevant results.

Keyword search over relational database. Many works have focused on keyword search over relational database such as DBXplorer [2], DISCOVER [11], BANKS [5]. DBXplorer generates trees of tuples that contain all query keywords and are connected through primary key-foreign key relationship. DISCOVER handles problem of keyword proximity search. BANKS uses an approximation of the Steiner tree problem to identify connected trees in a labeled graph. Our approach is different from keyword search over relational database especially when the database contains multi-valued attributes as discussed in Section 4.

Handling tag name keywords. Identifying meaningful return nodes are introduced in XReal [3] and XSeek [19]. XReal focuses on interpreting the search intentions of a keyword query according to the statistics in XML database and the pattern of keyword co-occurrence in a query. XSeek first applies the SLCA algorithm to find master entities, i.e., SLCA nodes, then infers the query interpretations from the input keywords to predict which parts of the subtrees rooted at the master entities should be selected. This happens in the post-processing phrase, i.e., after getting the SLCA nodes, while XSem determines the user's search intention in processing phrase (XSem uses guidelines for tag name keywords to identify the expected answers effectively).

7 Conclusion and Future work

We systematically illustrated several serious problems and faults of the existing LCA-based XML keyword search approaches, when they handle XML data and queries involving relationships among objects. Whether the LCA-based approach can find correct answers depends on how relationship types are represented by some hierarchical structure in XML. We conclude that the LCA-based search model is not a correct model for XML keyword search in general cases. This finding is significant since most current XML keyword search approaches are LCA-based. As a result, we proposed to use semantics to process XML keyword queries. In particular, we proposed rules and guidelines to infer answers for general XML keyword queries. Based on these rules and guidelines, we designed XSem, a search engine for semantics-based XML keyword search. Our experiments show that XSem returns more accurate answers for queries involving relationship than the LCA-based approach. As part of our future work, we will design more efficient algorithms for semantics-based XML keyword search. Another future work direction is to use tag name for improving ranking and output information.

References

[1] http://www.comp.nus.edu.sg/∼ltngoc/basketballplayer.xml.

[2] S. Agrawal, S. Chaudhuri, and G. Das. DBXPlorer: A system for keyword-based search over relation databases. In *ICDE*, 2002.

[3] Z. Bao, T. W. Ling, B. Chen, and J. Lu. Efficient XML keyword search with relevance oriented ranking. In *ICDE*, 2009.

[4] K. Benny and S. Yehoshua. Finding and approximating top-k answers in keyword proximity search. PODS, 2006.

[5] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.

[6] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A semantic search engine for XML. In *VLDB*, 2003.

[7] M. F. Fernandez, A. Morishima, D. Suciu, and W. C. Tan. Publishing relational data in XML: the silkroute approach.

[8] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD Conference*, 2003.

[9] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD Conference*, 2007.

[10] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava. Keyword proximity search in XML trees. *IEEE Trans. Knowl. Data Eng.*, 2006.

[11] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *PVLDB*, 2002.

[12] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *ICDE*, 2003.

[13] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, and R. D. Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.

[14] G. Konstantin, K. Benny, and S. Yehoshua. Keyword proximity search in complex data graphs. SIGMOD, 2008.

[15] G. Li, J. Feng, J. Wang, and L. Zhou. Effective keyword search for valuable LCAs over XML documents. In *CIKM*, 2007.

[16] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: Efficient and adaptive keyword search on unstructured, semi-structured and structured data. In *SIGMOD*, 2008.

[17] Y. Li, C. Yu, and H. V. Jagadish. Schema-free XQuery. In *VLDB*, 2004.

[18] T. W. Ling, M. L. Lee, and G. Dobbie. *Semistructured Database Design.* Springer-Verlag, 2004.

[19] Z. Liu and Y. Chen. Identifying meaningful return information for XML keyword search. In *SIGMOD Conference*, 2007.

[20] Z. Liu and Y. Chen. Reasoning and identifying relevant matches for XML keyword search. *PVLDB*, 1(1), 2008.

[21] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently publishing relational data as XML documents. *The VLDB Journal*, 2001.

[22] Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD*, 2005.