Exploiting Semantics for XML Keyword Search

Thuy Ngoc Le[1] #, Zhifeng Bao[2] *, and Tok Wang Ling[3] #

#National University of Singapore
[1] *ltngoc@u.nus.edu*
[3] *lingtw@comp.nus.edu.sg*
*RMIT University, Australia
[2] *zhifeng.bao@rmit.edu.au*

Abstract. XML keyword search has attracted a lot of interests with typical search based on lowest common ancestor (LCA). However, in this paper, we show several problems of the LCA-based approaches, including meaningless answers, incomplete answers, duplicated answers, missing answers, and schema-dependent answers. To handle these problems, we exploit the semantics of object, object identifier, relationship, and attribute (referred to as the ORA-semantics). Based on the ORA-semantics, we introduce new ways of labeling and matching. More importantly, we propose a new semantics, called CR (Common Relative) for XML keyword search, which can return answers independent from schema designs. To find answers based on the CR semantics, we discover properties of common relative and propose an efficient algorithms. Experimental results show the seriousness of the problems of the LCA-based approaches. They also show that the CR semantics possesses the properties of completeness, soundness and independence while the response time of our approach is faster than the LCA-based approaches thanks to our techniques.

1 Introduction

Since XML has become a standard for information exchange over the Internet, more and more data are represented as XML. Therefore, XML has wide applications such as electronic business[1], science[2], text databases[3], digital libraries[4], healthcare[5], finance[6], and even in the cloud [3]. As a result, XML has attracted a huge of interests in both research and industry with a wide range of topics such as XML storage, twig pattern query processing, query optimization, XML view, and XML keyword search. There have been several XML database systems such as Timber [10], Oracle XML DB[7], MarkLogic

[1] http://www.ebxml.org
[2] http://www.biodas.org/documents/spec-1.53.html
[3] http://www-connex.lip6.fr/~denoyer/wikipediaXML/
[4] http://www.loc.gov/standards/mods/presentations/mets-mods-morgan-ala07/
[5] http://www.ncbi.nlm.nih.gov/pubmed/11066651
[6] http://schemas.liquid-technologies.com/Category/Financial
[7] http://www.oracle.com/technetwork/database-features/xmldb/overview/index.html

Server[8], and the Toronto XML Engine[9]. XML keyword search has also been studied extensively based on lowest common ancestors such as SLCA [24], VLCA [16], MLCA [19] and ELCA [26].

Keyword search is a user-friendly way so that users can issue keyword queries without or with little knowledge about the schema of the underlying data. However, they often know what the data is about. Therefore, when they issue a query, they often have some expectations about the answers in mind. Since they may not know which schema is being used, their expectations are independent from schema designs. If they already got some answers for this schema, it could be surprised if different answers are returned when they try another schema which represents the same data content. Thus, different schemas of the same data content should provide them the same answers. However, this is not the case for the existing LCA-based approaches as shown in Example 1.

Running database: Consider the database with the ER diagram in Figure 1. There are many ways to represent this database in XML. Figure 2 shows five possible XML schema designs for this database. For simplicity, we do not show attributes and values in these schemas. Each edge in the schemas corresponds to a *many-to-many* relationship types between the two object classes.

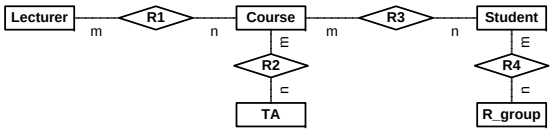


Figure 1: ER diagram of a database

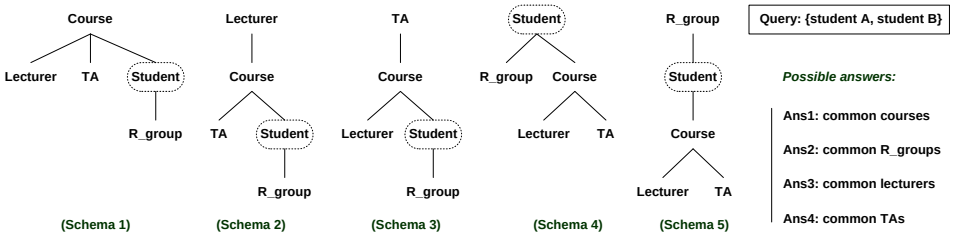


Figure 2: Equivalent XML schemas of the database in Figure 1

[8] http://www.marklogic.com/
[9] http://www.cs.toronto.edu/tox/

Example 1 (Schema dependence) *Users may know a university database about courses, lecturers, teaching assistants (TAs), students, and research groups (R_group)*[10]*, but they do not know what the schema looks like, i.e., which of the five schema designs in Figure 2 is used. When they ask for two* students *(e.g., Q = {StudentA, StudentB}), beside information about the two students, they may want to know some of the below:*

- *Ans1: the common* courses *that they both take,*
- *Ans2: the common* research groups (R_groups) *that they both belong to,*
- *Ans3: the common* lecturers *who teach both of them,*
- *Ans4: the common* teaching assistants (TAs) *who teach and mark both of them.*

They are common ancestors in some schema(s): Ans1 in Schema 1, Schema 2 and Schema 3; Ans2 in Schema 5; Ans3 in Schema 2; and Ans4 in Schema 3. Therefore, they are all meaningful answers (probably with different ranking scores). Different users may have different expectations. However, expectations of a user should be independent from schema designs because he does not know which schema is used. However, all five different schema designs provide five different sets of answers by the LCA semantics. Particularly:

- *for Schema 1: only Ans1 could be returned;*
- *for Schema 2: Ans1 and Ans3 could be returned;*
- *for Schema 3: Ans1 and Ans4 could be returned;*
- *for Schema 4: no answer;*
- *for Schema 5: only Ans2 could be returned.*

The above example provides a strong evidence for our two following arguments:

Firstly, meaningful answers can be found beyond common ancestors because all kinds of answers Ans1, Ans2, Ans3 and Ans4 are meaningful. However, if relying only on the common ancestor techniques, none of the five schemas can provide all the above meaningful answers. For some schema, answers from common ancestors may be better than the others, but returning more meaningful answers would be better than missing meaningful ones.

A final answer obtained by LCA-based approaches includes two parts: a returned node (LCA node) and a presentation of the answer, e.g., a subtree or paths. Arguably, the presentation of an answer as a subtree may contain other answers. For instance, for Schema 1, the subtree rooted at the common courses (Ans1) that both students take may contain other kinds of answers (Ans2, Ans3, Ans4). However, the LCA-based approaches do not explicitly identify them and it may be hard for users to identify them because this presentation contains a great deal of irrelevant information. Thus, it is necessary to identify and separate them clearly.

Secondly, answers of XML keyword search should be independent from the schema designs, e.g., Ans1, Ans2, Ans3 and Ans4 should be returned regardless

[10] R_group can be an object class with attributes: name, topics, leader, etc.

which schema is used to capture data. However, as can be seen, the LCA-based approaches return different answer sets for different schema designs in Figure 2.

In practice, many real XML datasets have different schema designs such as IMDb[11] and NBA[12]. In IMDb, there are many ways to capture relationships among actors, actresses, movies, and companies. In NBA, relationships among coaches, teams, and players can also be captured in different ways. Moreover, due to the flexibility and exchangeability of XML, many relational datasets can be transformed to XML [12], and each relational database can correspond to several XML schemas by picking up different entities as the root for the resulting XML document.

Therefore, it necessitates to consider the above two arguments when processing XML keyword search. However, to the best of our knowledge, no current system satisfies the above two arguments, including keyword search over XML graph.

Challenges. To determine what should be returned beside common ancestors is a great challenge. First, the new answers must be reasonably meaningful. That they must also cover possible answers returned by other alternative schemas is even harder. After such kinds of answers are defined, another challenge is how to construct an efficient index and how to find answers efficiently. Finding common ancestors is efficient because the computation can be based on node labels. However, this technique cannot be easily applied for finding other types of answers.

Other problems of the LCA-based approaches. Let us call the discussed problem where answers depend on the schema used to represent data content as the problem of schema-dependent answers. This problem occurs when multiple XML documents share the same data content. We find that even when we only consider one XML document but not other equivalent XML documents, the LCA-based approaches also have other problems. Particularly, they may also suffer from the following problems.

- (P1) Meaningless answers: they may return answers without any other information beside the input query keywords.
- (P2) Incomplete answers: they may return answers which do not contain enough information about all objects related to a relationship attribute.
- (P3) Duplicated answers: they may return answers which provide the same information due to duplicated objects.
- (P4) Missing answers: due to only search up from matching nodes, they may miss meaningful answers appearing as descendant of those nodes.

We will discuss these problems in details in Section 3. In summary, the problems of the LCA-based approaches we would like to solve in this paper can be summarized in Figure 3. We study from the case where data content is captured in only one XML document to the case where multiple XML

[11] http://www.imdb.com/interfaces
[12] http://www.nba.com

documents can share the same content by representing the content in different ways. For the former, we handle family of problems of the existing XML keyword search, including meaningless answers, missing answers, duplicated answers, and incomplete answers. For the latter, we deal with the problem of schema-dependent answers, which is the most challenging problem.

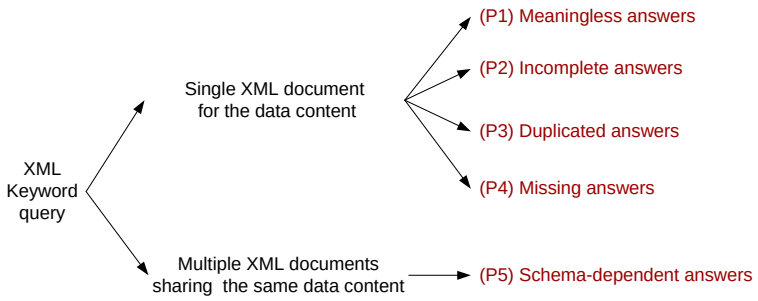


Figure 3: Problems of the LCA-based approaches to be solved

Our approach and contributions. We make the following contributions.

- *Problems of the LCA-based approaches.* We show that answers of the LCA-based approaches depend on the schema design. In addition, we find that the LCA-based approaches may miss meaningful answers, return meaningless answers, incomplete answers and duplicated answers (Section 3).
- *New semantics.* We propose a new semantics for XML keyword search, called CR (Common Relative), which provides common relatives as answers. A common relative corresponds to a common ancestor in some equivalent schema(s). The CR semantics not only improves the effectiveness by providing more meaningful answers beyond common ancestors, but also returns the same answer set regardless of different schemas backing the same data content. So it is more reliable and stable to users. The CR semantics can be applied for both XML documents with and with no IDREFs. It helps solve the problems of dependent answers and missing answers of the LCA-based approaches (Section 4).
- *Labeling and matching.* To solve the problems of meaningless answers and incomplete answers, we introduce a new scheme of labeling an XML document and a new scheme of matching a keyword to nodes in XML data (Section 5).
- *Indexing techniques.* Unlike conventional inverted index where each keyword has a set of matching nodes, to find common relatives efficiently, we need to maintain a set of relatives for each keyword, which is much more difficult

to construct. To accomplish this index, we propose some properties and an algorithm to identify relatives of a node effectively and efficiently (Section 5).
– *Processing techniques.* Unlike a common ancestor which appears at only one node, a common relative may be referred by multiple nodes. Therefore, we model data as a so-called *XML IDREF graph* by using *virtual IDREF* mechanism, in which we assign a *virtual object node* to connect all instances of the same object. We also discover the hierarchical structure of the XML IDREF graph and exploit it to find common relatives efficiently. In addition, our post-process helps filter out duplicated answers (Section 5).
– *Experiment.* The experimental results show the completeness, the soundness, and the independence from schema designs of our CR semantics. They also show the seriousness of problems of the LCA-based approaches. In addition, they show our approach can find answers based on the CR semantics efficiently (Section 6).

2 Background and Preliminary

2.1 LCA-based approaches

When XML documents do not contain IDREF, they can be modeled as trees. Approaches to handle such documents are called tree-based approaches because they are based on tree model. Inspired by the hierarchical structure of the tree model, most of existing tree-based approaches are based on the LCA (Lowest Common Ancestor) semantics, which was first proposed in XRANK [7]. By the LCA semantics, for a set of matching nodes, each of which contains at least one query keyword and each query keyword matches at least one node in this set, the lowest common ancestor (LCA) of this set is a returned node. An answer is a subtree rooted as a returned node (i.e., an LCA) or a path from the returned node to matching nodes.

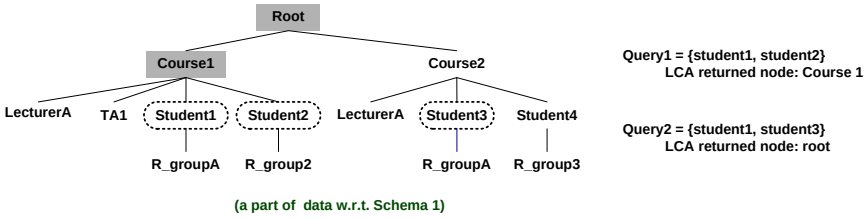


Figure 4: Illustration for the LCA semantics

For example, with the data in Figure 4, for query {`Student1, Student2`}, the returned node is `Course1`; and for for query {`Student1, Student3`}, the returned node is the root of the data.

2.2 Object-Relationship-Attribute (ORA)-semantics

The term *semantics* has different interpretations. We defined the ORA-semantics as the identifications of nodes in XML data and schema. The following are some concepts of the ORA-semantics used in this paper. Readers can find more information about the ORA-semantics in our previous works [18, 14, 13].

At schema level. An *object class* is an internal node representing a real world entity. An object class has a set of *object attributes* to describe its properties. Each object class has an *object identifier (OID)* to uniquely identify its instances. Several object classes may be connected through a *relationship type*, which may have a set of *relationship attributes*. For example, the ORA-semantics of the schemas in Figure 2 includes:

- `Lecturer, Course, TA, Student` and `R_group` are object classes.
- There are four many-to-many relationship types between object classes: each corresponds to each edge connecting object classes.

At data level. *Object*, *relationship*, *OID value* and *attribute value* at data level belong to object class, relationship type, OID and attribute at schema level respectively. In an XML document, an object can have multiple instances, each of which is represented by a group of nodes, starting at a tag w.r.t. object class, followed by a set of attributes and their associated values. Among the nodes describing an object instance, the one that belongs to an object class is called an *object node* and all remaining nodes are called *non-object nodes*. An object node is considered as the representative of an object and non-object nodes are associated with the corresponding object node. Thus, in unambiguous contexts, for simplicity, we use the object node to refer to the whole object instance. For example, in the data in Figure 4, all nodes are object nodes because we already associate all attributes to object nodes.

ORA-semantic vs. XML schemas. The ORA-semantics cannot be fully captured in XML schemas such as XML Schema or DTD. In XML schemas, relationships among objects cannot be captured, thus relationship attributes and object attributes cannot be distinguished. In addition, OID of objects cannot be fully captured either. XML schemas can only capture OID if it is defined as ID. However, for the child objects of many-to-many relationships without using ID/IDREF, OID cannot be expressed as ID because the child objects are duplicated. Furthermore, these schemas cannot distinguish between object classes and multi-valued attributes because they are both represented as star (*) nodes.

ORA-semantic discovery. Discovering the ORA-semantics involves determining the identification of nodes in XML data and schema. If different matching nodes of a keyword have different identifications, then that keyword corresponds to different concepts of the ORA-semantics. The recall and

precision of discovery of the ORA-semantics in XML data and schema is high (greater than 94.5% for both recall and precision) in our previous work [18]. Thus, for this paper, we assume the task of discovering the ORA-semantics has been done. In this paper, we focus on exploiting the ORA-semantics to handle problems of the LCA-based approaches, especially the problem of schema-dependence.

2.3 Other concepts

A *reasonable schema* is a schema in which an implicit relationship type must be represented by adjacent object classes, i.e., there is nothing between object classes of a relationship type. The same data content can have different reasonable *schema designs* (or *schemas* in short). For example, to transform from a relational database to XML, there are different schema designs, each of which corresponds to a way that XML organizes the data. These schemas are *equivalent* in the sense that they capture the same information in different ways. We call databases corresponding to these equivalent schemas and represent the same data content as *equivalent databases*.

In an XML document with IDREFs, an object node which is referred by some other object node(s) by IDREFs is called a *referred object node*. In other words, a referred object node is an object node having IDREFs as its incoming edges.

In an XML data tree, the path of a node u, denoted as $path(u)$, is the path from the root to u. All nodes having the same path belong to the same object class.

3 Other problems of the LCA-based approaches

Beside the serious schema-dependence problem of the LCA-based approaches discussed in Section 1, those approaches suffers from the following problems:

- (P1) meaningless answers
- (P2) incomplete answers
- (P3) duplicated answers
- (P4) missing answers

In this section, we systematically point out these problems of the LCA semantics by comparing answers returned by the LCA semantics and answers that are probably expected by users. We will also discuss the reason behind each problem.

We will illustrate these problems by example at data level. We take the data w.r.t. Schema 2 for illustration, though the data of other schemas also provides the same problems. For simplicity, we only use part of the data containing information about lecturers, courses and students as in Figure 5. A more detailed schema which contains OID and attributes of object classes is also provided. In this schema, `grade` is an attribute of the relationship between `Course` and `Student` rather than an attribute of `Student`.

Since we use an object node as the representative for a whole object including attributes and values, we denote an object node using `Object class (Dewey label)` (e.g., `Lecturer(1.1)`). Since an object can be identified by its object class and OID, we denote it `<object class: OID>` (e.g., `<Lecturer:L1>`).

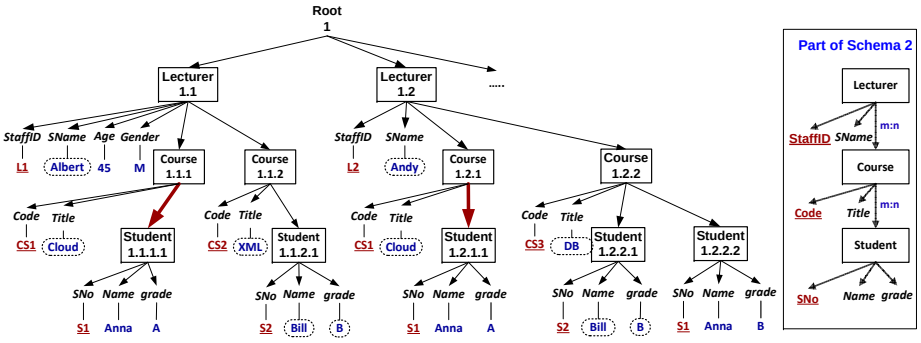


Figure 5: Part of data w.r.t. Schema 2 in Figure 2

3.1 (P1) Meaningless answer

Consider query {`Albert`}. The LCA-based approaches return value node `Albert` (under `Lecturer(1.1)`), or some approaches return attribute `Sname` and value node `Albert`. However, such answers are not useful since they do not provide any supplementary information about `Albert`. This happens when a returned node is a non-object node, e.g., an attribute or a value.

The LCA-based approach cannot differentiate object and non-object nodes. Returning non-object node is not meaningful whereas returning object node is meaningful because it contains other associated attributes and values. Therefore, the expected answer should be forced up to `Lecturer(1.1)`, the object containing `Albert` since it contains supplementary information related to `Albert` such as `StaffID, Age` and `Gender`.

Several works such as XSeek [20], XReal[1], and [23] have attempted to solve the problem of meaningless answers by identifying entity (object), and they can obtain more meaningful answers in several cases. However, these works do not use OIDs as ours, and thus they do not always distinguish an object from an aggregation node, a composite attribute, and a multi-valued attribute. As a result, they cannot avoid meaningless answers for many scenarios.

3.2 (P2) Incomplete answer

Suppose that in Figure 5, the relationship between `Course` and `Student` has an attribute `grade` to describe grade of a student for a particular course. Consider

query {Bill, B} where Bill matches student(1.1.2.1) and student(1.2.2.1); and B matches value of relationship attribute grade under those two students. Note that two matching nodes of Bill are duplicated because they both refer to <Student:S2>.

The LCA-based approaches return Student(1.1.2.1) and student(1.2.2.1) (and attributes under them) as answers. There are two problems with these answers. First, these answers are not correct because 'B' grade is not an attribute of student, but it is grade of a student taking the course instead, i.e., Grade is an attribute of the relationship between Student and Course. With these answers, users do not know which course the students got grade 'B'.

Second, these answers seem the same but in fact 'B' grade under Student(1.1.2.1) is the grade of student named Bill (<Student:S2>) taking course XML, while 'B' grade under Student(1.2.2.1) is for the same student (<Student:S2>) taking course DB. Thus, despite of looking exactly the same, these answers are not duplicated.

The LCA-based approaches face these problems because they do not consider the ORA-semantics. Thus, they cannot distinguish between an object attribute and a relationship attribute. Once we know that grade is an attribute of the relationship between Student and Course, the proper answers should be moved up to include courses taken by the queried students.

3.3 (P3) Duplicated answer

A common problem with XML is that objects can be duplicated in multiple places to achieve a tree structure when there exist *many-to-many* or *many-to-one* relationships. Queries which match each of these duplicate instances, as will be discussed in this section, often return overwhelmingly large result sets. Suppose a course is taken by 300 students in an XML document where student is parent of course (as Schema 4 and Schema 5 in Figure 2), then when a query about this course is issued, users get 300 duplicated answers about the same course. Therefore, it is necessary to filter out duplicated answers, which are answers provide the same information due to duplicated objects.

The duplication of objects is due to many-to-many ($m : n$) or many-to-one ($m : 1$) relationships because in such relationship, the child object is duplicated each time it occurs in the relationship. For example, in the XML data in Figure 5, since the relationship between lecturer and course is $m : n$, a course can be taught many lecturers such as <Course:CS1> about Cloud is taught by both <Lecturer:L1> and <Lecturer:L2>. Thus, the child object (<Course: CS1>) is duplicated as two object nodes Course(1.1.1) and Course(1.2.1).

Now we consider examples of duplicated answers due to duplicated objects. Formal definition of duplicated answers will be introduced in Section 5.6. Consider query {Course, Cloud}. The LCA-based approaches will return Course(1.1.1) and Course(1.2.1) as two answers. However, we can see that these answers refer to the same object because they belong to the same object class Course and have the same object ID value CS1. As such, it is enough to

return this object just once. The LCA-based approaches do not discover such duplicated answers because they cannot detect the duplication of objects having multiple occurrences.

Importance of detecting duplicated answers. Detecting duplicated answers eliminates users' irritation due to overwhelming answers. More importantly, detecting duplicated answers (or detecting duplicated objects in particular), is very necessary in other applications. For instance, the computation for group-by and aggregate functions, which is discussed in [13], cannot be correct without considering duplication.

For example, to *count the number of students taught by lecturer Albert*, a user can issue a query {`Andy, count student`} to the data in Figure 5. Without considering duplicated objects, the number of students is three. However, object node `Student (1.2.1.1)` and object node `Student (1.2.2.2)` refer to the same object `<Student:S1>`. Hence, only two students are taught by lecturer `Andy`.

3.4 (P4) Missing answer

Consider query {`XML, DB`} where the query keywords match two courses, `Course(1.1.2)` and `Course(1.2.2)` respectively. The LCA-based approaches return the document root for this query, which is definitely not meaningful for users. The LCA-based approaches only search up to find common ancestors, but never search down to find common information appearing as their descendants. In the considered data, `Student(1.1.2.1)` and `Student(1.2.2.1)` refer to the same student `<Student:S2>` though they appear in different places. Therefore, they should be returned as answers. Intuitively, this is the student taken both courses. Due to unawareness of semantics of object, the LCA-based approaches can never recognize such duplicated information and thus they miss this meaningful answer.

Once the problem of schema dependence is solved, the problem of missing answers is also solved. This is because an answer set which is independent schema designs provides answers beyond common ancestors, including missing common descendants.

4 The CR semantics

This section introduces our proposed semantics, called CR (Common Relative), which can return more meaningful answers beyond LCAs of matching nodes and the returned answer set is independent from schema designs. For ease of comprehension, we first present intuitive analysis about the CR semantics by example.

4.1 Intuitive analysis

We analyze the problem in Example 1 and discuss how to find all types of answers with only one particular schema. For simplicity, figures used for illustration in this section provide intuitive information and only contain object nodes, without attributes and values. For example, for the left most figure in Figure 6, `StudentA` means that this node together with the corresponding attributes and values represent information about `studentA`; or `common R_group` represents the `research group` that both `StudentA` and `StudentB` belong to.

Example 2 (Using one schema to find all types of answers) *Recall that in Example 1, there are four types of meaningful answers for a query about two students (e.g., `StudentA` and `StudentB`). Each type of answers can be returned by the LCA semantics for some schema(s) in Figure 2. They are: Ans1 (common `courses`) from Schema 1, Schema 2 and Schema 3, Ans2 (common `R_groups`) from schema 5, Ans3 (common `lecturers`) from Schema 2, and Ans4 (common `TAs`) from Schema 3. Now we discuss how a database w.r.t. a given schema can return all the above answers. We take the data of Schema 1 for illustration.*

For Ans1 (common `courses`): this is a common ancestor of the two students and Schema 1 can provide it.

For Ans2 (common `R_groups`): Schema 1 cannot provide it, but Schema 5 can provide it. Figure 6 shows that in Schema 1, common `R_groups` appear as descendants of the two students. If these descendants are connected by a referred object node via IDREFs, Ans2 can be found at that referred object node. We call that referred object node as a common descendant.

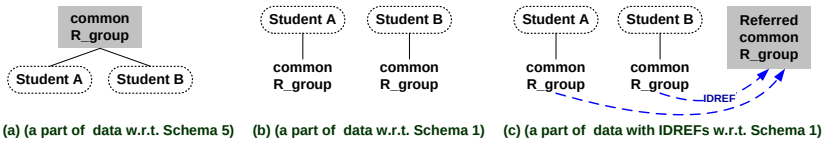


Figure 6: Illustration for Ans2 (common R_groups)

For Ans3 (common `lecturers`): Schema 1 cannot provide it, but Schema 2 can provide it. Figure 7 shows that in Schema 1, common `lecturers` appear as relatives of the two students (formal definition of relative is given in Section 4.2). If these relatives are connected by a referred object node via IDREFs, Ans3 can be found at that referred object node. We call that referred object node as a common relative.

Ans4 (common `TAs`) is similar to Ans3 (common `lecturers`).

As can be seen, all types of answers can be found at common ancestors, common descendants, or common relatives. Although we only take the data of Schema 1 for illustration, the data of other schemas have similar results when analyzed.

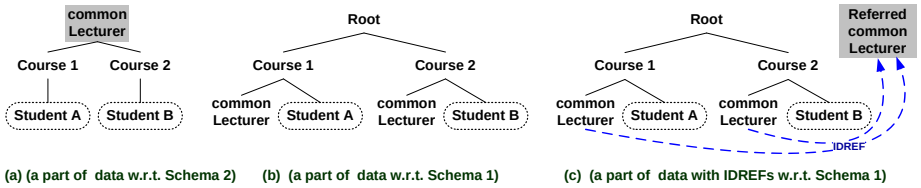


Figure 7: Illustration for Ans3 (common lecturers)

4.2 The CR semantics

Before introducing the new semantics, let us present some properties which makes the semantics meaningful. Consider a chain C: $< u_1, u_2, \ldots, u_n >$ of object nodes, where u_i and u_{i+1} have parent-child or child-parent relationship in an XML data D. We have the following properties related to C.

Property 1 *If C is a parent-child chain of object nodes, i.e., u_i is the parent of u_{i+1} $\forall i$, then all nodes on the chain C have different node paths.*

The above property is obvious. Recall that node path (or the path of a node) presented in Section 2 is the path from the root to that node. If an object class has multiple occurrences in XML schema, its instances may corresponds to different node paths.

Property 2 *The chain C has a corresponding chain C': $< u'_1, u'_2, \ldots, u'_n >$ of object nodes in a database D' equivalent to D, where u'_i refers to the same object with u_i.*

Property 2 can be illustrated in Figure 8, in which the data chain $< u_1, u_2, u_3, u_4 >$ (in the most left) has three corresponding chains $< u'_1, u'_2, u'_3, u'_4 >$ in its equivalent databases. Combining Property 1 and Property 2, we have Property 3.

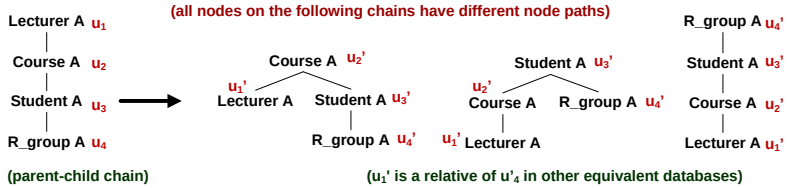


Figure 8: The "same" chain w.r.t. different equivalent databases

Property 3 *If C is a parent-child chain, then there always exists a corresponding chain C': $< u'_1, u'_2, \ldots, u'_n >$ of object nodes in another database D' equivalent to D, where u'_i refers to the same object with u_i $\forall i$, such that all object nodes u'_i's in the chain C' have different node paths.*

We call nodes u'_i's in the chain C' in Property 3 are *relatives* of each other. It has different meanings from relatives in family relationship and it is defined as follows.

Definition 1 (Relative) *In an XML data tree, an object node u is a* relative *of an object node v if there is a chain of object nodes from u to v where all object nodes on that chain (including u and v) have different node paths.*

By Definition 1, ancestors and descendants of a node u are also relatives of u. However, siblings of u may or may not be relatives of u, depending on the node path of u and that of its siblings. The following properties are inferred from Definition 1 and Property 3.

Property 4 *If u is a relative of v in an XML database D, then there exists some XML database D' equivalent to D such that u' is an ancestor of v', where u' and v' refer to the same object with u and v respectively.*

Property 5 *If w is a common relative of u and v in an XML database D, then there exists some XML database D' equivalent to D such that w' is a common ancestor of u' and v', where w', u' and v' refer to the same object with w, u and v respectively.*

The proof of Property 5 is given in Appendix. By Property 4, a relative corresponds to an ancestor in some equivalent database(s). More generally, a common relative corresponds to a common ancestor in some equivalent database(s) as stated in Property 5. Since a common ancestor can provide a meaningful answer, a common relative should correspond to an answer. Based on all discussions above, we propose the novel semantics for XML keyword search as follows.

Definition 2 (The CR (Common Relative) semantics) *Given a keyword query $Q = \{k_1, \ldots, k_n\}$ to an XML database, an answer to Q is a pair $\langle c, \mathbb{K} \rangle$ where:*

- $\mathbb{K} = \bigcup_1^n u_i$ *where object node u_i matches k_i.*
- *c is a common relative of $\mathbb{K}$.*

Although the number of relatives of an object node may be large, the number of relatives which is potential to be common relatives is much fewer as will be discussed in Property 7 in Section 5.3. We only index such potential relatives, not all relatives. This saves index space dramatically.

We consider all common ancestors (common ancestors are a part of common relatives) of matching nodes instead of filtering out common ancestors which are less relevant as the LCA semantics and its extensions such as SLCA, ELCA do. This is because in many cases, this filter loses many meaningful answers. For example, consider a query about two students. For Schema 2 in Figure 2, if two students take the same course, then the lecturer teaches that course cannot be returned as an answer. However, common lecturer of two students is meaningful to users.

4.3 The CR semantics for XML document with IDREFs

XML allows the notion of reference links from a node to other nodes not its children, using an IDREF (ID Reference) mechanism. In this section, we will investigate the case where an XML document contains IDREFs. Particularly, we will extend the CR semantics discussed in Section 4.2 for XML documents with IDREFs.

Definition 3 (Relative-extend) *In an XML data tree, a node u (an object node or a referred object node) is a* relative *of a node v (an object node or a referred object node) if there is a chain of nodes (object nodes or referred object nodes) from u to v such that for every two nodes on that chain, either (1) they have different node paths, or (2) they are referrer and referee of each other.*

Recall that a *referred object node* is an object node which is referred via IDREFs by some other object node(s). The latter is called its referrer(s). Intuitively, the condition in Definition 1 is extended: the referred object node is not considered if its referrer(s) are already considered. We also have the following property.

Property 6 *If an object node u is a relative of an object node v, then u is also a relative of the referred object node of v.*

With Definition 3 and Property 6, Property 4 and Property 5 keep valid. Therefore, Definition 2 is also valid for XML documents with IDREFs. Let us illustrate by the following example.

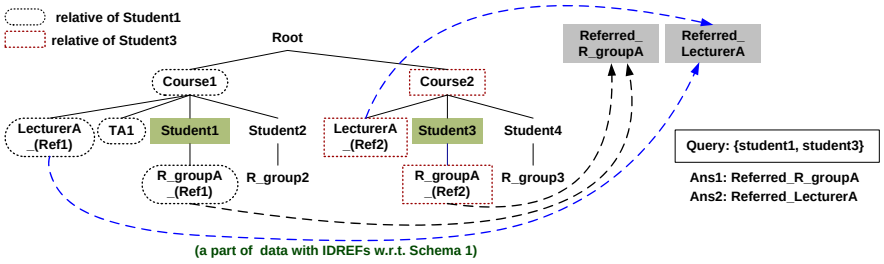


Figure 9: Illustration for query {`Student1, Student3`}

Example 3 *Consider the data in Figure 9. For simplicity, we only show object nodes in the data and all attributes and values are associated to the corresponding object nodes. In this data, we use ID/IDREFs to connect all instances of the same object. Particularly,* Referred_LecturerA *is referred by LecturerA_(Ref1) and LecturerA_(Ref2) by IDREFs; and* Referred_R_groupA *is referred by R_groupA_(Ref1) and R_groupA_(Ref2) by*

IDREFs. Noted that an XML document can contain both objects with duplication and objects with IDREFs. For query {Student1, Student3}, by Definition 3, we have:

- *Relatives of Student1: Student1, Course1, TA1, LecturerA_(Ref1),* Referred_LecturerA*, R_groupA_(Ref1), and* Referred R_groupA.
- *Relatives of Student3: Student3, Course2, LecturerA_(Ref2),* Referred_LecturerA*, R_groupA_(Ref2), and* Referred R_groupA.

Therefore, the common relatives of the two students are Referred_LecturerA*, and* Referred_R_groupA *which provide two answers for the query.*

We can also see that Referred_LecturerA is a relative of Student1 and Student3; and its referrers, LecturerA_(Ref1) and LecturerA_(Ref2), are also relatives of Student1 and Student3 respectively (Property 6). It is similar to Referred_R_groupA.

Later, in Example 4, we will show that by applying Property 7 (discussed in Section 5.3), the lists of relatives of keywords are much shorter. This helps save index space and improve efficiency of the search.

5 Our semantics-based approach

5.1 Overview of our approach

To handle five discussed problems of the LCA-based approaches, we propose a semantics-based approach, in which we exploit the ORA-semantics (introduced in Section 2.2) and apply the CR semantics. In particular:

(P1) meaningless answers and (P2) incomplete answers. To solve (P1) and (P2), we use all concepts of the ORA-semantics such as object class, object, non-object, object attribute and relationship attribute. We introduce the new way of labeling in which we assign all attributes and values the same label with the corresponding object, and the new way of matching for query keywords, especially for relationship attributes and their values. More details will be discussed in Section 5.4.

(P3) duplicated answers. To solve (P3), we also need the ORA-semantics to discover duplicated objects, which is defined based on object class and OID. This problem will be post-processed in Section 5.6.

(P4) missing answers. The CR semantics returns common ancestors, common descendants (special cases of common relatives) and common relatives as answers. Therefore, it already includes missing answers. Hence, the CR semantics also solves (P4).

(P5) schema-dependent answers. Among problems of the LCA-based approaches, (P5) is the most challenging problem. To solve (P5), we follow our

proposed CR semantics, which returns common relatives for a set of matching object nodes. Finding common relatives is much more challenging than finding common ancestors.

- First challenge: while the set of ancestors of a node can be easily identified based on the hierarchical structure of XML, the set of relatives of a node are difficult to identify.
- Second challenge: Given a set of matching nodes, unlike a common ancestor which appears as only one node, a common relative may be referred by many different nodes. Therefore, it requires more complex techniques for indexing and searching to find common relatives.

To address the first challenge, we discover some properties about the relationships of relatives. These properties enable us to introduce an effective algorithm to pre-compute all relatives of a node (Section 5.3).

To address the second challenges, we model an XML document as a so-called *XML IDREF graph*, in which all instances of the same object are connected via IDREFs by a referred object node. Thereby, all instances of a common relative are also connected by a referred object node (Section 5.2). Another difficulty appears when searching over an XML IDREF graph. Searching over graph-structured data has been known to be equivalent to the group Steiner tree problem, which is NP-Hard [6]. To solve this difficulty, we discover that XML IDREF graph is a special graph. Particularly, it is an XML tree (with parent-child (PC) edges) plus a portion of *reference edges*. A reference edge is an IDREF from a referring node to a referred node. Although these nodes refer to the same object, we can treat them as having a parent-child relationship, in which the parent is the referring node and the child is the referred node. This shows that XML IDREF graph still has hierarchy, which enables us to generalize efficient techniques of LCA-based approaches (based on the hierarchy) for searching over an XML IDREF graph. Particularly, we use ancestor-descendant relationships among nodes for indexing (Section 5.3 and Section 5.4). Thereby, we do not have to traverse the XML IDREF graph when processing a query (Section 5.5).

5.2 Data modeling

We propose virtual ID/IDREF mechanism, in which we assign a virtual referred object node as a hub to connect all instances of the same object by using virtual IDREF edges. The resulting model is called an XML IDREF graph. Intuitively, it is ID/IDREF mechanism, but we do not modify XML documents and IDREFs are virtually created just for finding common relatives. In the XML IDREF graph, there may co-exist both real and virtual IDREFs. For example, in the XML IDREF graph in Figure 9, LecturerA_(Ref1) and LecturerA_(Ref2) are instances of the same object and there are two virtual IDREFs to connect them with the virtual object node Referred_Lecturer_A.

To generate an XML IDREF graph from an XML document, we need to detect object instances of the same object. Since an object is identified by object class and OID, two object instances (object nodes as their representatives) are

considered as the same object if they belong to the same *object class* and have the same *OID* value. In many cases, object classes (e.g., Lecturer, Course, Student, TA and R_group in Figure 2) and OIDs are directly available, because XML was initially designed based on them. When this is not the case, these values can be discovered from XML by our previous work [18], which achieve high accuracy (greater than 98% for object classes and greater than 93% for OIDs). Thus, we assume object classes and OIDs are available.

5.3 Identifying relatives of a node

To facilitate the search, we identify the set of relatives of a node in advance and maintain an index for the set of relatives for each object node. To solve challenges of identifying such sets, we propose the following properties about the relationships of relatives. Note that, as discussed in Section 5.2, the data is modeled as an XML IDREF graph which still has hierarchy. Thus, it contains ancestor-descendant relationships among nodes.

Property 7 *Among relatives of an object node u, potential common relatives of u and other object node(s) can only be ancestors of u or relatives of u which are also referred object nodes, i.e., object nodes with IDREFs as incoming edges.*

We discover that not all relatives can become common relatives. A common relative of more than one node must be able to connect multiple nodes. Thus, it can only fall into cases in Figure 10. We can ignore Case 3 because u is already the common ancestor of u and v in this case. Therefore, to be a potential common relative, a relative of a matching object node u must be u, or an ancestor of u, or a relative of u which is also a referred object node. Thereby, this saves index space significantly and therefore improves the efficiency of the search as well.

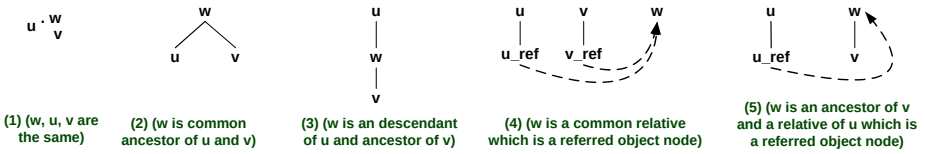


Figure 10: Cases which w is a common relative of u and v

Example 4 *Recall Example 3 with Figure 9 to find answers for query {Student1, Student3}. By Property 7, the set of relatives of the keywords which can be potential common relatives are:*

- *For Student1: Student1, Course1, Referred_LecturerA and Referred_R_groupA*
- *For Student3: Student3, Course2, Referred_LecturerA and Referred_R_groupA*

where Student1 and Student3 are matching object nodes; Course1 and Course2 are ancestors of matching nodes; and Referred_LecturerA and Referred_R_groupA are referred object nodes. The common relatives are Referred_LecturerA and Referred_R_groupA. As can be seen, we can get the same answers as in Example 3 while the sets of relatives of keywords is much fewer. TA1, LecturerA_(Ref1) and R_groupA_(Ref1) (relatives of Student1); and LecturerA_(Ref2) and R_groupA_(Ref2) (relatives of Student3) are not considered because they cannot be a common relative.

Property 8 *Consider two sets $\mathbb{S}_1$ and $\mathbb{S}_2$ where (1) each set contains all object nodes of the same node path, (2) the node paths w.r.t. these two sets are different, and (3) these sets do not contain referred object nodes and are sorted by document order. If $u_i \in \mathbb{S}_1$ is a relative of $v_{j-1} \in \mathbb{S}_2$, but not a relative of $v_j \in \mathbb{S}_2$, then u_i will not be a relative of $v_k \in \mathbb{S}_2$ $\forall k > j$.*

This is because node $u_i \in \mathbb{S}_1$ can have many relatives in $\mathbb{S}_2$, but these relatives are continuous in $\mathbb{S}_2$ because the sets are sorted by document order as illustrated in Figure 11. Thus, instead of checking all nodes in $\mathbb{S}_2$, we can proactively stop the checking soon thanks to Property 8.

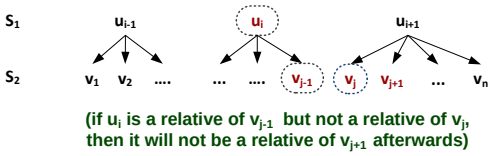


Figure 11: Illustration for Property 8

Property 9 *In XML data, two nodes u and v are relative if and only if the path of their LCA corresponds to the path of the LCA of their schema nodes in XML schema. In other words, we have:*
$path(LCA(u, v)) = path(LCA(schema(u), schema(v))) \leftrightarrow relative(u, v) = true$
where $schema(u)$ is the corresponding node in XML schema of node u.

For example, in Figure 9, we have: path(Course1) = path(Course2) = root/Course; and path(Student1) = root/Course/Student. Therefore, we have:

- path(LCA(Student1, Course 1)) = root/Course = path(LCA(schema(Student1), schema(Course 1))). Thus, Student1 and Course 1 are relatives.
- path(LCA(Student1, Course 2)) = root ≠ path(LCA(schema(Student1), schema(Course 2))) = root/Course. Thus, Student1 and Course 2 are not relatives.

Property 9 is used to construct the set of relatives of a node efficiently. We provide the proof of this property in Appendix.

Based on all the discussions above, we design an algorithm for constructing of the set of relatives of object nodes in Algorithm 1. For an object node u, we only consider nodes having different node path with $path(u)$ thank to Definition 1.

Algorithm 1: Find relatives of object nodes

```
   Input: All object nodes in an XML data
   Output: The set of relatives Rel(u) of each object node u
 1 for each object node u in the data do
 2     for each node path p ≠ path(u) (//Def.1) do
 3         v_first ← find the first relative of u
 4         for each node v after v_first having node path p do
 5             if path(LCA(u, v)) is path(LCA(schema(u), schema(v))) (//Prop
               9) then
 6                 flag = 1;
 7                 if v is an ancestor of u (//Prop 7) then
 8                     Add v to Rel(u)
 9                 else
10                     (//Prop 7)
11                     ref(v) ← object node referred by v
12                     if ref(v) is not in Rel(u) then
13                         Add ref(v) to Rel(u)
14             else
15                 if flag = 1 then
16                     flag = 0;
17                 break (for non-referred nodes) //Prop8
```

Space complexity. The space complexity for index is $N \times (H + R)$ where N is the number of *real object nodes* in the XML IDREF graph; H is the maximum number of *ancestors* of a real object nodes, which is equal to the height of the XML IDREF graph (XML IDREF graph still has hierarchy); and R is the maximum number of referred object nodes which are referred by the relatives of a real object node. N is much smaller than the number of nodes (including attributes and values) in an XML data. H is usually a very small number. Thus, the space for indexing is reasonable.

5.4 Labeling, matching and indexing

Labeling. We only label *object nodes*. All *non-object nodes* are assigned the same label with their corresponding object nodes. Thereby, the number of labels is largely reduced. We use *number* instead of Dewey for labeling because computation on number is faster than on Dewey since a Dewey label has

multiple components to be accessed and computed. Each virtual node is also assigned a label which succeeds labels of real nodes.

By assigning the same label for all nodes of an object, we associate all attributes and values to the corresponding object nodes. Thereby, only object nodes can be returned as CRs. Therefore, we can avoid *meaningless answers*.

Matching. To solve the problem of *incomplete answers* (when queries related to relationship attributes), we exploit the ORA-semantics to distinguish a relationship attribute and an object attribute. An object attribute belongs to an object while a relationship attribute or its value belongs to a relationship between/among objects, not just belong to the lowest object (of the relationship) where it appears as the child. Once relationship attributes are identified, we re-define the matching of query keywords as follows.

If a keyword k matches a relationship attribute value u, then k is treated as matching all objects participating in the relationship which u belongs to. Similarly, if a keyword k matches a relationship attribute name u, then k matches all object classes participating in the relationship type which u belongs to.

By this matching, when a query contains a relationship attribute (or its value), it is considered as related all objects involved in the relationship which the attribute belongs to. Therefore, we can avoid *incomplete answers* when handling relationship attributes because those objects are included in final answers.

For example, in query {`Bill, B`}, `'B'` has two matches under `student(1.1.2.1)` and `student(1.2.2.1)`. Since `'B'` is the value of a relationship attribute, we consider grade `'B'` under `student(1.1.2.1)` matches `course(1.1.2)` as well. It is similar to the other grade 'B' under `student(1.2.2.1)`. As a result, the two answers contains two courses as shown in Figure 12.

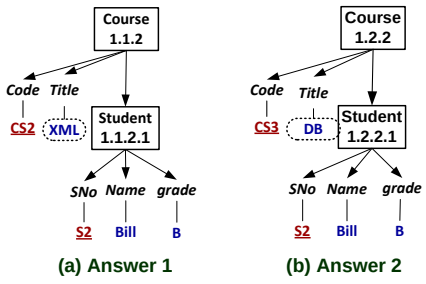


Figure 12: Answers for query {`Bill, B`}

Indexing. Each keyword k has a set $Rel(k)$ of relatives of *real object nodes* matching k. We have $Rel(k) = \bigcup Rel(u_i)$ where u_i is an object node matching

k and $Rel(u_i)$ is the set of relatives of u_i. u_i must be an object node because of our labeling scheme, which helps reduce the index size dramatically. u_i is a real node because virtual nodes, which are created only for connecting instances of the same object, do not contain contents. To identify $Rel(u_i)$, we follow the properties and algorithm introduced in Section 5.3.

5.5 Processing

Thanks to the index where we already have the set of relatives of each keyword, the processing of our approach is very efficient as follows. Consider a query $Q = \{k_1, \ldots, k_n\}$. Let $CR(Q)$ denote the set of common relatives of Q. We have $CR(Q) = \bigcap_1^n Rel(k_i)$, where $Rel(k_i)$ denotes the set of relatives of nodes matching keyword k_i. Therefore, to find $CR(Q)$, we compute the intersection of sets $Rel(k_i)$'s;

The computation for set intersection can leverage any existing fast set intersection algorithms. The computation of set intersection has been used to find SLCA and ELCA in [25] and has been shown to be more efficient than the traditional computation based on common prefix of labels when dealing with XML tree.

5.6 Post-processing: removing duplicated answers

Duplicated answers are filtered out at the post-processing step. By Definition 2, an answer for a query is defined as $\langle c, \mathbb{K} \rangle$ where $\mathbb{K}$ is a combination of matching object nodes and c is common relative of $\mathbb{K}$. Based on this definition, we define duplicated answers as follows.

Definition 4 (Duplicated answers) *Two answers $\langle c_1, \mathbb{K}_1 \rangle$ and $\langle c_2, \mathbb{K}_2 \rangle$ are duplicated if (1) c_1 and c_2 refer to the same object (i.e., they have the same OID); and (2) $\mathbb{K}_1$ and $\mathbb{K}_2$ refer to the same set of objects (i.e., for each object node u in $\mathbb{K}_1$, there exists an object node v in $\mathbb{K}_2$ such that u and v refer to the same object).*

Detecting duplication. If there exists a $m : n$ or $m : 1$ relationship type between object classes A and B, then for all object classes appearing as B or the descendants of B, the objects of those classes may have duplication. Otherwise, with no $m : n$ or $m : 1$ relationship type, duplication does not happen. Therefore, to detect duplication, we first identify the possibility of duplication by checking $m : n$ and $m : 1$ relationship types. If there is no $m : n$ and no $m : 1$ relationship type, we can determine quickly the objects which are not duplicated. Once we determine that an object is possibly duplicated, we determine whether two objects (of the same object class) are really duplicated by checking whether they have the same OID.

5.7 Output presentation

To avoid irrelevant information, we present an answer as a path from a returned CR node to matching object nodes. Since we use XML IDREF graph without users' awareness, we do not show IDREFs in answers. Thus, a return CR is converted to the set of its referrers if it is a virtual referred object node.

For example, for query {`Cloud, DB`} issued to the data in Figure 5, there are two answers. Answer1 corresponds to the common relative lecturer `Andy` (`<Lecture:L2`). Answer2 corresponds to student Anna `<Student:S1>`. Note that In Answer2, both `Student(1.2.1.1)` and `Student(1.2.2.2)` represent student Anna `<Student:S1>`. The outputs contain the paths from the returned nodes to the matching object nodes of the issued query as shown in Figure 13.

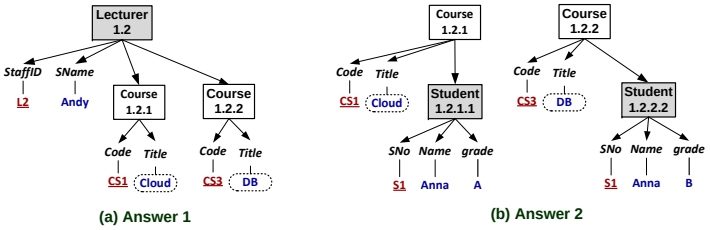


Figure 13: Output presentation of query {`Cloud, DB`}

If an answer is not related to all objects of a relationship, then the output will not contain any relationship attributes of that relationship. For example, for query {`Student Bill`}, the answer will not contain `grade` because this query does not related to course. Intuitively, grade is not attribute of student, it is attribute of the relationship between student and course. Therefore, if course is not related to the query, that relationship should not be shown in the answer as illustrated in Figure 14.

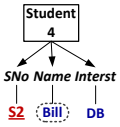


Figure 14: `grade` is not included in the output of query {`Student Bill`}

6 Experiment

In this section, we evaluate the completeness, the soundness, the independence from schemas of our proposed CR semantics. Before checking these properties of the CR semantics, duplicated answers are filtered out. We consider both schemas with no IDREF and with IDREFs. We also make a comparison between our semantics and common ancestors, SLCAs and ELCAs [25]. In addition, we study the percentage of queries suffering from each kind of problems of the LCA-based approaches. Finally, we compare the efficiency of our approach with an LCA-based approach. The experiments were performed on an Intel(R) Core(TM) i7 CPU 3.4GHz with 8GB of RAM.

6.1 Experimental setup

Dataset. We pre-processed two real datasets including **IMDb**[13], and **Basketball**[14]. We used the subsets with the sizes of 150MB and 86MB for IMDb and Basketball respectively. In IMDb, there are many ways to capture relationships between actors, actresses, movies, and companies. In Basketball, relationships between coaches, teams, and players also can be captured in different ways.

The ORA-semantics of dataset. We discovered the ORA-semantics for each dataset by using the algorithms in [18]. To make sure the ORA-semantics we have got is correct, we then manually checked and corrected the obtained ORA-semantics. For example, the ORA-semantics of Basketball dataset is given in Table 1. We have a similar table for IMDb dataset.

Table 1: The ORA-semantics of Basketball dataset

Object class	OID	Object attribute
Coach	CoachID	firstName, lastName
Team	team	name
Player	ilkID	firstName, lastName

Relationship type	Relationship attribute
between Coach and Team	Coach_Team_info (composite attribute), year, ….
between Player and Team	Player_Team_info (composite attribute), year, gp

Creating equivalent databases. For each dataset, we manually designed all possible schemas, both with no IDREF and with IDREFs. For example, there

[13] http://www.imdb.com/interfaces
[14] http://www.databasebasketball.com/stats_download.htm

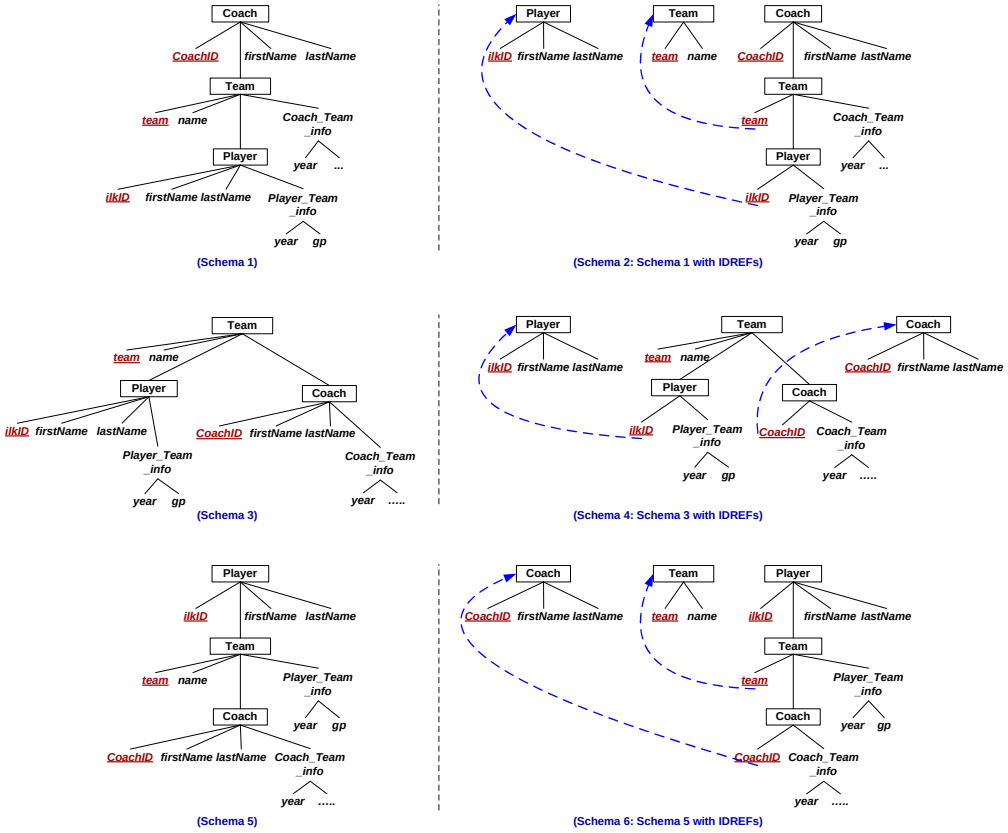


Figure 15: Six equivalent schema designs of Basketball dataset

are three equivalent schemas with no IDREF for Basketball, corresponding to picking up three different object classes (Coach, Team, Player) as the root of the schema. There are three more equivalent schemas containing IDREFs, corresponding to transforming from the three equivalent schemas with no IDREF. As a result, we have six equivalent schemas for Basketball dataset as shown in Figure 15. From the original databases, we automatically created the corresponding database for each schema of each dataset.

Query set. We randomly generated 50 queries from document keywords. To avoid meaningless queries, we filtered out generated queries which do not contain any value keyword, such as queries contains only tags, or prepositions, or articles, e.g., query {and, the, to}. 35 remaining queries include 20 and 15 queries for Basketball and IMDb datasets respectively.

Compared Algorithms. We compared our approach with an LCA-based approach to show the advantages of our approach over the LCA-based approaches. We chose Set-intersection [25] because it processes two popular semantics: SLCA and ELCA, because it is one of the most recent works, and because it outperforms other LCA-based approaches in term of efficiency.

6.2 Completeness

The completeness describes whether our semantics can return all common ancestors from all equivalent databases by using only one equivalent database. To study the completeness, for each query, we calculated the ratio of the number of CAs from all equivalent databases found in CRs from the original database over the total number of CAs from all equivalent databases, i.e., $\frac{A \cap B}{B}$, where A is the number of answers by our proposed CR semantics from only the original database, and B is the number of all common ancestors (CAs) for all equivalent databases. The checking has been done both automatically and with user study.

Automatically. We based on Definition 5 to check whether the two answers from equivalent databases are the same or not. We achieved the result of 100% for Basketball and 100% for IMDb. This is because based on the properties and definitions in Section 4.2, given a query Q to a database D, for any common ancestor of Q in some database equivalent to D, there always exists a common relative of Q in D.

Definition 5 (Answer-equivalent) *Given an n-keyword query Q, two answers of Q $a_1 = \langle c_1, \mathbb{K}_1 \rangle$ in schema $\mathcal{S}_1$ where $\mathbb{K}_1 = \{u_1, \ldots, u_n\}$, and $a_2 = \langle c_2, \mathbb{K}_2 \rangle$ in schema $\mathcal{S}_2$ where $\mathbb{K}_2 = \{v_1, \ldots, v_n\}$ are equivalent w.r.t. Q, denoted as $a_1 \equiv_Q a_2$ if*

- *c_1 and c_2 refer to the same object and*
- *u_i and v_i refer to the same object for all i.*

User study. We asked 15 students in major of computer science to compare answers from different equivalent databases. Although the information for these answers are exactly the same by our Definition 5, they are represented in different ways due to different schemas such as two answers in Figure 6(a) and 6(c) or two answers in Figure 7(a) and 7(c). Thus, some users might think they are different. Therefore, we would like to study how users think about them. Surprisingly, we got the results of 100% for Basketball and 100% for IMDb from users. This implies that users share the same opinions with us on the similarity of answers.

6.3 Soundness

The soundness describes whether all answers (CRs) returned from our semantics can be common ancestors in other equivalent database(s). To study

the soundness, for each query, we calculated the ratio of the number of CRs from the original database found in all CAs from all equivalent databases over the total number of CRs from the original database, i.e., $\frac{A \cap B}{A}$, where A and B have the same meanings in Section 6.2. The checking was also done both automatically and with user study. We compared the two answers in the same manner with the discussion in Section 6.2.

We got the result of 100% for both Basketball and IMDb for automatical checking. This is because based on the discussions in Section 4.2, for any common relative of a query Q to a database D, there exists a common ancestor of Q in some database equivalent to D. For user study, we also got the surprising results of 100% for both Basketball and IMDb. This implies the agreements of users on our theories.

6.4 Schema-independence

To study the independence of our CR semantics from schemas, we checked whether the answer sets returned by the CR semantics from all equivalent databases are the same or not. The result is the ratio of the number of answers returned from all equivalent databases over the total number of distinct answers from all equivalent databases. We also performed this checking both automatically and with user study.

We achieved the result of 100% for Basketball and 100% for IMDb. This can be explained because the completeness and the soundness of our semantics are both 100%. This implies that for a query Q and two equivalent databases D and D'. If Ans is an answer of Q in D, then there exists an answer Ans' for Q in D' such that $Ans' \equiv_Q Ans'$ and vice versa. For user study, once again we got the result of 100% for Basketball and 100% for IMDb.

6.5 Comparing with SLCA and ELCA

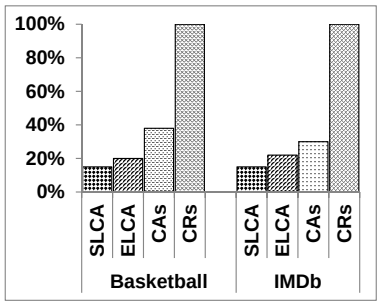


Figure 16: Percentages of CAs, ELCAs, SLCAs in CRs

For a given query Q, We have $CR(Q) \supseteq CA(Q) \supseteq ELCA(Q) \supseteq SLCA(Q)$. We ran our approach to find CAs and CRs while we ran Set-intersection [25] to

find SLCAs and ELCAs. Figure 16 shows the percentages of $CA(Q)$, $ELCA(Q)$ and $SLCA(Q)$ in $CR(Q)$ for the original databases. The results are similar for the two datasets. As can be seen, CAs is just around one third of CRs, and SLCAs and ELCAs are around 15% to 20% of CRs. This implies that our CR semantics improves the recall significantly by providing much more meaningful answers.

6.6 Statistics on problems of the LCA-based approaches

We investigated on the percentage of queries having each kind of problem of the LCA-based approaches. We classify 35 generated queries into five categories, including queries having (P1) meaningless answer, (P2) incomplete answers, (P3) duplicate answer, (P4) missing answer, and (P5) schema-dependent answers. Note that a query may suffer from more than one problem. For the problems (P1) - (P4), we only consider on the original schema (e.g., Schema 1 of Basketball).

The statistics on the number of queries having each kind of problem and the corresponding percentage are shown in Table 2 and Figure 17. Although Figure 17 can be inferred from Table 2, we show it for a more visual display.

Table 2: Statistics on the problems of the LCA-based approaches

	Basketball (20 queries)		IMDb (15 queries)	
Problems of the LCA-based approaches	Number of queries	Percentage	Number of queries	Percentage
P1. Meaningless answers	4	20%	3	20%
P2. Duplicated answers	12	60%	10	66.7%
P3. Incomplete answers	4	20%	2	13.3%
P4. Missing answers	10	50%	8	53.3%
P5. Schema-dependent answers	16	80%	12	80%

As we can see that each kind of problems has a numerous queries involved in, especially 80% queries suffer from the problem of schema-independent answers. All queries containing only one keyword provide meaningless answers. Queries containing relationship attribute (e.g., year in Basketball) return incomplete answers. Queries related to lower objects (e.g., team, player in Basketball) most likely will return duplicated answers, while queries related to higher objects (e.g., coach, team in Basketball) usually miss answers. Since both datasets have multiple designs, most of queries return different answers for different schema designs, except the queries matching only one value. However, such queries (matching only one value) return meaningless answers. Therefore, we find that almost all queries issued to the considered datasets suffer from at least one problem of the LCA-based approaches.

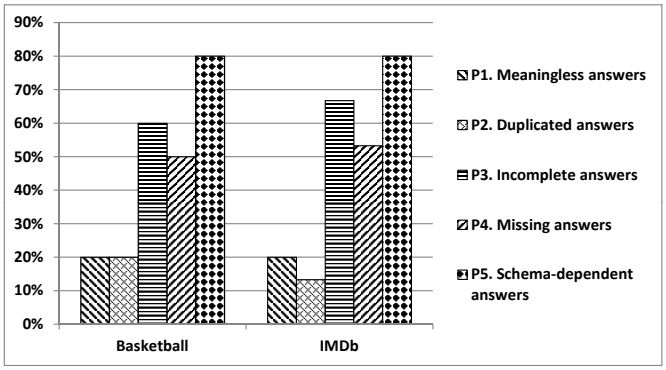


Figure 17: Statistics on the problems of the LCA-based approaches

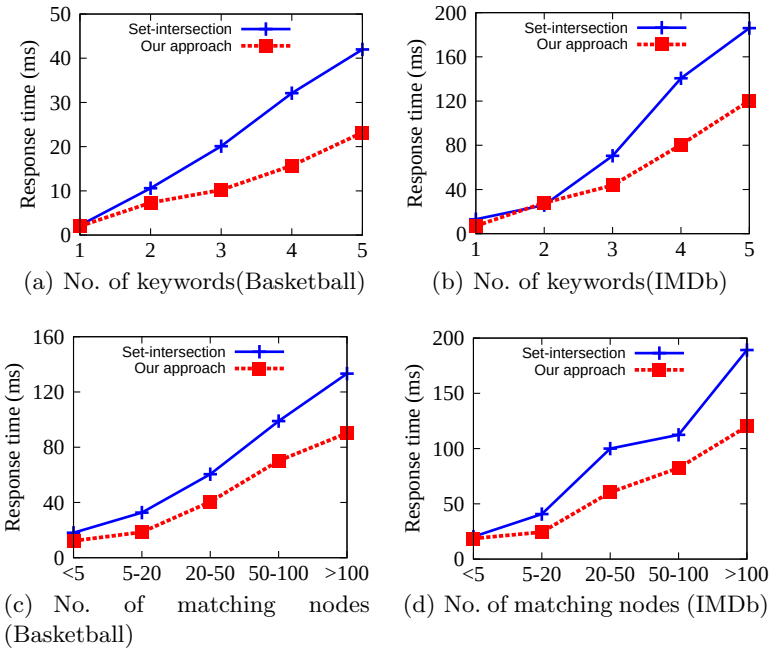
6.7 Efficiency evaluation

The response time of our approach and Set-intersection [25] is shown in Figure 18, in which we varied the number of query keywords and the number of matching nodes. Although our approach has to process more matching nodes because of the relatives, its response time is faster than the Set-intersection because of two reasons. Firstly, by only labeling object nodes and assign all non-objects nodes the same labels with the corresponding object nodes, the number of matching nodes for a keyword query is reduced. Secondly, Set-intersection has two phases for finding CAs and filtering some CAs to find SLCAs and ELCAs. In contrast, the processing of our approach is only similar to the first phase of Set-intersection.

7 Related work

LCA-based approaches. XRANK [7] proposes a stack based algorithm to efficiently compute LCAs. XKSearch [24] defines Smallest LCAs (SLCAs) to be the LCAs that do not contain other LCAs. Meaningful LCA (MLCA) [19] incorporates SLCA into XQuery. VLCA [16] and ELCA [26] introduces the concept of valuable/ exclusive LCA to improve the effectiveness of SLCA. XReal [1] proposes an approach based on Information Retrieval techniques. MaxMatch [21] investigates an axiomatic framework that includes the properties of monotonicity and consistency. MESSIAH [22] handles cases of missing values in optional attributes. Recently, XRich [14] takes common descendants into account of answers.

Graph-based approaches. Graph-based approaches can be classified based on the semantics such as the Steiner tree [5], distinct root [8] and subgraph [17, 11]. Later, [15] propose an approach to model XML data as a so-called OR graph. For an XML document with ID/IDREF, graph-based approaches such as [17, 11] can provide more answers by following IDREFs. However, those graph-based



(a) No. of keywords(Basketball) (b) No. of keywords(IMDb)

(c) No. of matching nodes (Basketball) (d) No. of matching nodes (IMDb)

Figure 18: Efficiency evaluation

approaches can do that only if XML documents contain ID/IDREF. Otherwise, those graph-based approaches do not recognize instances of the same object and may still miss meaningful answers.

Although extensive works have been done on improving the effectiveness, no work can provides answers which are independent from schema designs, and their returned answers cannot cover answers which can be found from other schema designs.

Semantics-based XML keyword search. XSEarch [4] focuses on adding semantics into query but the added semantics is for distinguishing a tag name and a value keyword only. XSeek [20] and MaxMatch [21] 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. XKeyword [9] exploits semantics from the XML schema. XReal [1], Bao et. al. [2] and Wu et. al. [23] proposed an object-level for XML keyword search. However, all of these works only consider objects, but they do not have the concepts of object ID, relationship and attribute. Therefore, they can avoid at most the problem of meaningless answers but still suffer from all other problems. Moreover, these works do not use OIDs as ours, and therefore do not always distinguish an object from an aggregation node, a composite attribute, and a multi-valued attribute. So they can only avoid meaningless answers in certain scenarios.

Recently, ORGraph [15] discussed other problems of LCA-based approaches. Nevertheless, this work transfers an XML document to a graph which is similar to relational database, and follows Steiner tree semantics. Thus, it suffers from the inefficiency and may return other kind of meaningless answers because matching nodes may not be (or weakly) related.

8 Conclusion

Since XML has become more and more popular, keyword search in XML data has attracted a lot of research interest. Besides structure, XML data does contain semantics of objects, relationships between/among objects, and their attributes (referred to as the ORA-semantics). However, existing works only rely on the structure of XML but ignore such semantics. We have pointed out that this causes some problems in XML keyword search, including the problems of meaningless answers, incomplete answers, duplicated answers, missing answers and especially schema-dependent answers.

To solve these problems, we exploited the ORA-semantics for the new way of labeling and matching to avoid meaningless answers and incomplete answers. Most importantly, we proposed a novel semantics called CR (Common Relative), which returns all common relatives of matching nodes as answers. Our proposed CR semantics not only provides more meaningful answers than common ancestors, but these answers are independent from schema designs of the same data content as well. Moreover, the CR semantics can be applied for both XML document with and with no IDREF. To efficiently find answers based on the CR semantics, we discovered some properties of relatives and designed an algorithm to find relatives of a node effectively and efficiently.

Experimental results showed the seriousness of the problems of the LCA-based approaches. They also showed that our CR semantics possesses the properties of completeness, soundness and independence from schema designs while the response time is faster than an LCA-based approach because we only work with object nodes.

References

1. Z. Bao, T. W. Ling, B. Chen, and J. Lu. Efficient XML keyword search with relevance oriented ranking. In *ICDE*, 2009.
2. Z. Bao, J. Lu, T. W. Ling, L. Xu, and H. Wu. An effective object-level XML keyword search. In *DASFAA*, 2010.
3. J. Camacho-Rodriguez, D. Colazzo, and I. Manolescu. Building large XML stores in the amazon cloud. In *ICDEW*, 2012.
4. S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A semantic search engine for XML. In *VLDB*, 2003.
5. B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in database. In *ICDE*, 2007.
6. S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, 1971.

7. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD*, 2003.
8. H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD*, 2007.
9. V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *ICDE*, 2003.
10. H. V. Jagadish and S. AL-Khalifa. Timber: A native XML database. Technical report, University of Michigan, 2002.
11. M. Kargar and A. An. Keyword search in graphs: finding r-cliques. *PVLDB*, 2011.
12. J. Kim, D. Jeong, and D.-K. Baik. A translation algorithm for effective RDB-to-XML schema conversion considering referential integrity information. *Journal Inf. Sci. Eng.*, 2009.
13. T. N. Le, Z. Bao, T. W. Ling, and G. Dobbie. Group-by and aggregate functions in XML keyword search. In *DEXA*, 2014.
14. T. N. Le, T. W. Ling, H. V. Jagadish, and J. Lu. Object semantics for XML keyword search. In *DASFAA*, 2014.
15. T. N. Le, H. Wu, T. W. Ling, L. Li, and J. Lu. From structure-based to semantics-based: Effective XML keyword search. In *ER*, 2013.
16. G. Li, J. Feng, J. Wang, and L. Zhou. Effective keyword search for valuable LCAs over XML documents. In *CIKM*, 2007.
17. 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.
18. L. Li, T. N. Le, H. Wu, T. W. Ling, and S. Bressan. Discovering semantics from data-centric XML. In *DEXA*, 2013.
19. Y. Li, C. Yu, and H. V. Jagadish. Schema-free XQuery. In *VLDB*, 2004.
20. Z. Liu and Y. Chen. Identifying meaningful return information for XML keyword search. In *SIGMOD*, 2007.
21. Z. Liu and Y. Chen. Reasoning and identifying relevant matches for XML keyword search. In *PVLDB*, 2008.
22. B. Q. Truong, S. S. Bhowmick, C. E. Dyreson, and A. Sun. MESSIAH: missing element-conscious SLCA nodes search in XML data. In *SIGMOD*, 2013.
23. H. Wu and Z. Bao. Object-oriented XML keyword search. In *ER*, 2011.
24. Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD*, 2005.
25. J. Zhou, Z. Bao, W. Wang, T. W. Ling, Z. Chen, X. Lin, and J. Guo. Fast SLCA and ELCA computation for XML keyword queries based on set intersection. In *ICDE*, 2012.
26. R. Zhou, C. Liu, and J. Li. Fast ELCA computation for keyword queries on XML data. In *EDBT*, 2010.

APPENDIX

A Proof of Property 5

Proof. We can always choose object class of w as the root of an equivalent XML schema, therefore, there always exists an XML document D' in which w_1 and w_2 are ancestors of u' and v' respectively, where w_1, w_2 refer to the same object with w; and u' and v' refer to the same object with u and v respectively.

Since object class of w is the root of the equivalent XML schema, its objects are not duplicated. Therefore, w_1 and w_2 must appear at the same object w' in D' because they refer to the same object. Therefore, w' is the common ancestor of u' and v' in D'. □

B Proof of Property 9

Proof. **If part:** To prove If path, we prove that if $path(LCA(u, v)) = path(LCA(schema(u), schema(v)))$ then all nodes in the chain between u and v have different node paths because then u and v are relatives of each other. We use contradiction.

If there exists two nodes X and Y on the chain from u to v (u - ... - X - ... - Y - ... - v) such that X and Y have the same node path (X and Y can be u and v), then X and Y cannot have ancestor-descendant relationship and the nodes in the chain are as in Figure 19. Hence, X and Y are ancestors of u and v respectively. Thus, $path(X)$ and $path(Y)$ are ancestors of $path(u)$ and $path(v)$ respectively. Therefore, $LCA(u, v) = LCA(X, Y)$ and $LCA(schema(u), schema(v)) = LCA(schema(X), schema(Y))$ (1).

We also have $path(LCA(X, Y)) \neq LCA(schema(X), schema(Y))$ because X and Y have the same node path (2).

From (1) and (2), we infer that $path(LCA(u, v)) \neq path(LCA(schema(u), schema(v)))$.

Therefore, if $path(LCA(u, v)) = path(LCA(schema(u), schema(v)))$, then there does not exist any two nodes two nodes X and Y on the chain from u to v such that X and Y have the same node path. In other words, all nodes on the chain u to v have different node paths. Therefore, by Definition 1, u and v are relatives of each other.

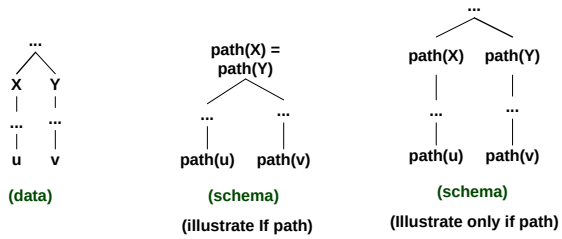


Figure 19: A chain u - ... - X - ... - Y - ... - v (X and Y can be u and v)

Only if part: If u and v are relatives of each other, then all nodes on the chain from u to v have different node paths as illustrated in Figure 19. Argue similarly to the proof of the if path, for all pairs of nodes X and Y on the chain from u to v such that X and Y are ancestors of u and v respectively, we have

$LCA(u, v) = LCA(X, Y)$ and

$LCA(schema(u), schema(v)) = LCA(schema(X), schema(Y))$ (3).

Moreover, for the highest node(s) X and Y in the chain from u to v (X and Y can be the same), we have

$path(LCA(X, Y)) = path(LCA(schema(X), schema(Y)))$ (4).

From (1) and (2), we infer that

$path(LCA(u, v)) = path(LCA(schema(u), schema(v)))$. □

Thuy Ngoc Le received her PhD in computer science from National University of Singapore. Her research interests include XML keyword search, XML query processing, semi-structured Data Model, keyword query processing for relational database and temporal database; big data, data mining, and data analytics over social networks.

Zhifeng Bao received his PhD in computer science from National University of Singapore. He is currently an Assistant Professor in Royal Melbourne Institute of Technology, Australia. He was the winner of the Singapore IDA gold medal. His research focuses on exploratory keyword search, spatio-textual query processing, visualized analytics over social networks.

Tok Wang Ling received his PhD in Computer Science from the University of Waterloo (Canada). He is a professor of the Department of Computer Science at the National University of Singapore. His research interests include Data Modeling, ER approach, Normalization Theory, Semistructured Data Model, XML query processing, and XML and RDB keyword query processing. He has published more than 220 papers, co-authored a book, co-edited a book, and co-edited 9 conference proceedings. He is an ACM Distinguished Scientist, a senior member of IEEE, and an ER Fellow.