THE NATIONAL UNIVERSITY
of SINGAPORE

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

TRA6/14

Schema-independence in XML Keyword Search

Thuy Ngoc Le, Zhifeng Bao and Tok Wang Ling

June 2014

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

Foreword

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

David ROSENBLUM
Dean of School

Schema-independence in XML Keyword Search

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

[1]National University of Singapore {*ltngoc,lingtw*}*@comp.nus.edu.sg*
[2]University of Tasmania & HITLab Australia *zhifeng.bao@utas.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 that meaningful answers can be found beyond LCA and should be independent from schema designs of the same data content. Therefore, we propose a new semantics, called CR (Common Relative), which not only can find more answers beyond LCA, but the returned answers are independent from schema designs as well. To find answers based on the CR semantics, we propose an approach, in which we have new strategies for indexing and processing. Experimental results show that the CR semantics can improve the recall significantly and the answer set is independent from the schema designs.

1 Introduction

Since XML has become a standard for information exchange over the Internet, more and more data are represented as XML. At the same time, there is increasing recognition of the importance of flexible query mechanisms including keyword queries. Therefore, XML keyword search has been studied extensively based on lowest common ancestors such as SLCA [14], VLCA [9], MLCA [12] and ELCA [16].

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.

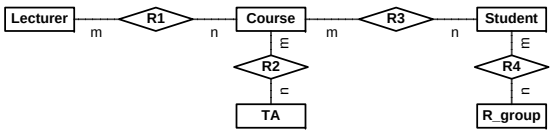


Figure 1: ER diagram of a database

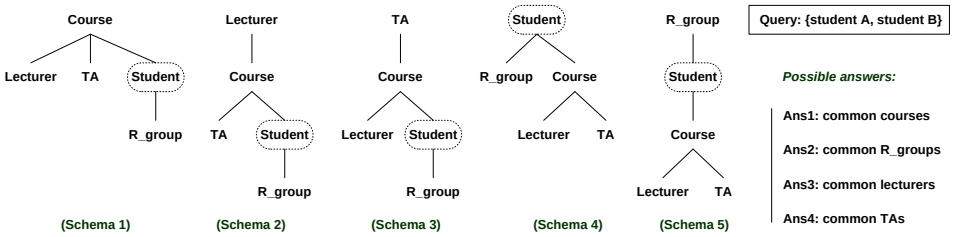


Figure 2: Equivalent XML schemas of the database in Figure 1

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.

Example 1 (Schema dependence) *Users may know a university database about courses, lecturers, teaching assistants (TAs), students, and research groups (R_group)*[1]*, 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

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

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 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[2] and NBA[3]. 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 [6], 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.

Our approach and contributions. We make the following contributions.

- *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 (Section3).
- *Indexing techniques.* Unlike conventional inverted index where each keyword has a set of matching nodes, to find common relatives efficiently, we need to maintain

[2] http://www.imdb.com/interfaces

[3] http://www.nba.com

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 4).
- *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 (Section 4).
- *Experiment.* The experimental results show the completeness, the soundness, and the independence from schema designs of our CR semantics. They also show our approach can find answers based on the CR semantics efficiently (Section 5).

2 Preliminary

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*.

An *object* is identified by *object class* and *object identifier* (OID). In XML, it occurs as object instances, each of which is represented by a group of nodes, rooted at the object class tagged node, followed by a set of attributes and their associated values to describe its properties. In this paper, we refer to the root of this group as an *object node* and the other nodes as *non-object nodes*. Hereafter, in unambiguous contexts, we use object node as the representative for a whole object instance, and nodes are object nodes by default. For example, matching node means matching object nodes.

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.

3 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.

3.1 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 3, `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 3 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.

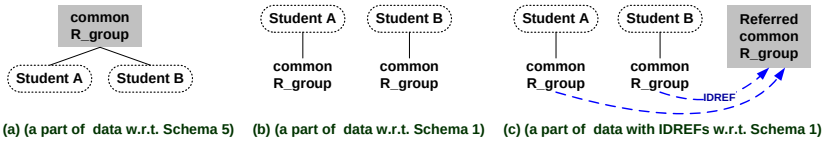


Figure 3: Illustration for Ans2 (common R_groups)

For Ans3 (common `lecturers`): Schema 1 cannot provide it, but Schema 2 can provide it. Figure 4 shows that in Schema 1, common `lecturers` appear as relatives of the two students (formal definition of relative is given in Section 3.2). 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.

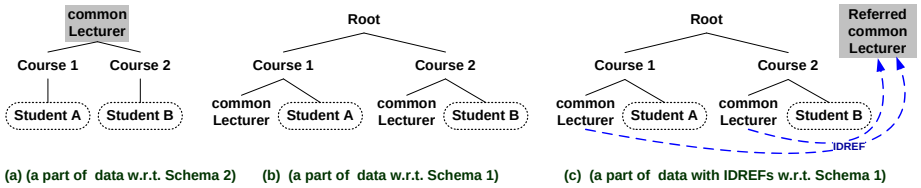


Figure 4: Illustration for Ans3 (common lecturers)

3.2 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 5, 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.

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.

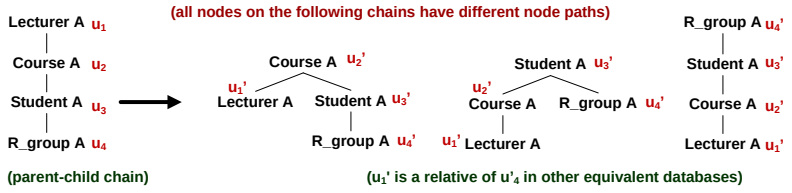


Figure 5: The "same" chain w.r.t. different equivalent databases

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.*

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 $\mathbb{K}$.*

When the XML document contains IDREFs, the referred node and its referrer(s) refer to the same object. For such documents, the two definitions above are still valid with the following extensions: (1) for the condition in Definition 1, the referred object node is not considered if its referrer(s) are already considered, and (2) a relative of a referring object node is also a relative of its referred object node.

Example 3 *Consider query {Student1, Student3} to the data in Figure 6, in which we use ID/IDREFs to connect all instances of the same object. Since* Referred_LecturerA *are referred by LecturerA_(Ref1) and LecturerA_(Ref2) by IDREFs, it is considered as a relative of nodes which its two referrers are relatives. It is similar for* Referred_R_groupA. *As a result, 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.*

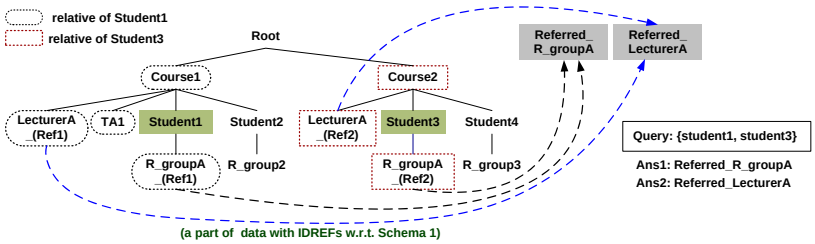


Figure 6: Illustration for query {`Student1, Student3`}

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 6. 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 Our schema-independent approach

Our approach is to find answers for a query under 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. Firstly, 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. Secondly, 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 4.2).

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 4.1). Another challenge 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 [3]. In contrast, keyword search on XML tree is much more efficient based on the hierarchical structure of XML tree. This is because the search in an XML tree can be reduced to find LCAs

of matching nodes, which can be efficiently computed based on the longest common prefix of node labels.

To solve the above challenge, 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 4.2 and Section 4.3). Thereby, we do not have to traverse the XML IDREF graph when processing a query (Section 4.4).

4.1 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 6, 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 [11], 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.

4.2 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 4.1, the data is modeled as an XML IDREF graph which still has hierarchy. Thus, it contains ancestor-descendant relationships among nodes.

Property 6 *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 7. 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.

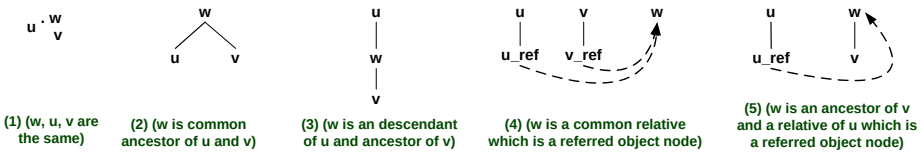


Figure 7: Cases which w is a common relative of u and v

Example 4 *Recall Example 3 with Figure 6 to find answers for query* $\{$*Student1, Student3*$\}$*. By Property 6, 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 7 *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 8. Thus, instead of checking all nodes in $\mathbb{S}_2$, we can proactively stop the checking soon thanks to Property 7.

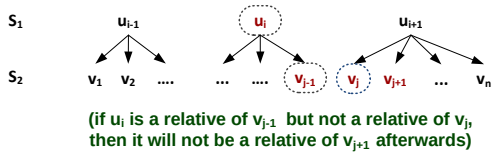


Figure 8: Illustration for Property 7

Property 8 *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.

Proof.
If path: 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 9. 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.

Only if path: 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 9. 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 $LCA(schema(u), schema(v)) = LCA(schema(X), schema(Y))$ (1).

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 $path(LCA(X,Y)) = path(LCA(schema(X), schema(Y)))$ (2).

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

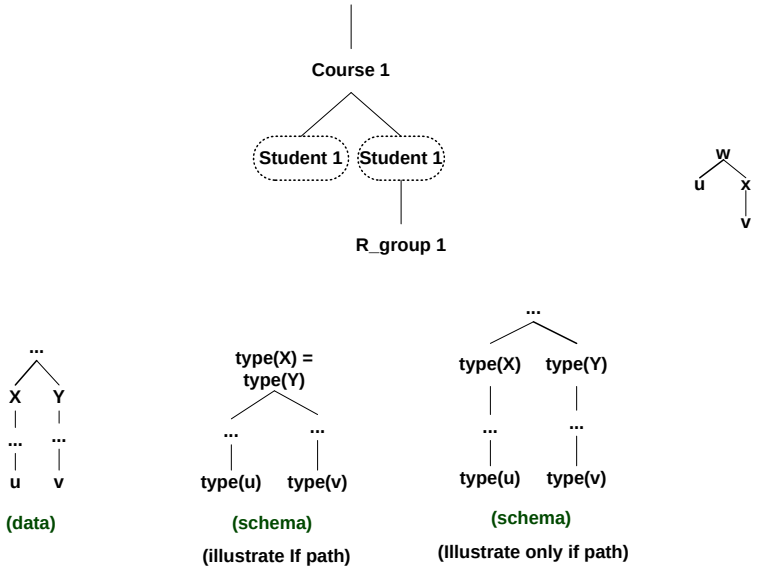


Figure 9: A chain u - . . . - X - . . . - Y - . . . - v (X and Y can be u and v)

This property is used to construct the set of relatives of a node efficiently. It is illustrated in the following example.

Example 5 *In Figure 6, we have:*

- *path(Course1) = path(Course2) = root/Course*
- *path(Student1) = path(Student2) = root/Course/Student.*
- *LCA(Student1, Course1) = Course1*
- *LCA(Student1, Course2) = root*
- *LCA(Student1, Student2) = Course1*

  Therefore, we have:
- *path(LCA(Student1, Course 1)) = root/Course = path(LCA(schema(Student1), schema(Course1))). Thus, Student1 and Course 1 are relatives.*
- *path(LCA(Student1, Course 2)) = root $\neq$ path(LCA(schema(Student1), schema(Course2))) = root/Course. Thus, Student1 and Course 2 are not relatives.*
- *path(LCA(Student1, Student2)) = root/Course $\neq$ path(LCA(schema(Student1), schema(Student2))) = root/Course/Student. Thus, Student1, Student2 are not relatives.*

Thus, for Student1, Course1 is its relatives, but Course2 and Student2 are not.

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 8) then
6                  flag = 1;
7                  if v is an ancestor of u (//Prop 6) then
8                      Add v to Rel(u)
9                  else
10                     (//Prop 6)
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) //Prop7
```

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.

4.3 Labeling 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 in XML IDREF graph, a node can have multiple parents. In addition, 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.

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 4.2.

4.4 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 [15] and has been shown to be more efficient than the traditional computation based on common prefix of labels when dealing with XML tree.

5 Experiment

In this section, we evaluate the completeness, the soundness, the independence from schemas of our proposed CR semantics. We also make a comparison between our semantics and common ancestors, SLCAs and ELCAs [15]. 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.

5.1 Experimental setup

Dataset. We pre-processed two real datasets including **IMDb**[4], and **Basketball**[5]. 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.

Creating equivalent databases. For each dataset, we manually designed all possible schemas. For example, there are three equivalent schemas for Basketball, corresponding to picking up three different object classes (Coach, Team, Player) as the root of the schema (Figure 10). Because of more equivalent schema designs for IMDb and due to space constraints, we do not show these schemas of IMDb. 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

[4] http://www.imdb.com/interfaces

[5] http://www.databasebasketball.com/stats_download.htm

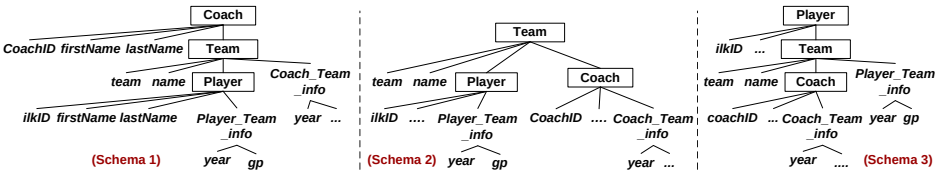


Figure 10: Three equivalent schema designs of Basketball dataset

keyword, such as queries contains only tags, or prepositions, or articles, e.g., query {and, the, to}. 35 remaining queries include 20 and 12 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 [15] 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.

5.2 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 3 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 3.2, 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 3 (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 3, they are represented in different ways due to different schemas such as two answers in Figure 3(a) and 3(c) or two answers in Figure 4(a) and 4(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.

5.3 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 5.2. 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 5.2.

We got the result of 100% for both Basketball and IMDb for automatical checking. This is because based on the discussions in Section 3.2, 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.

5.4 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.

5.5 Comparing with SLCA and ELCA

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 [15] to find SLCAs and ELCAs. Figure 11 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.

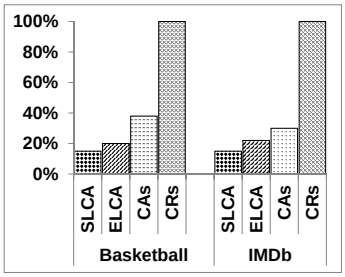


Figure 11: Percentages of CAs, ELCAs, SLCAs in CRs

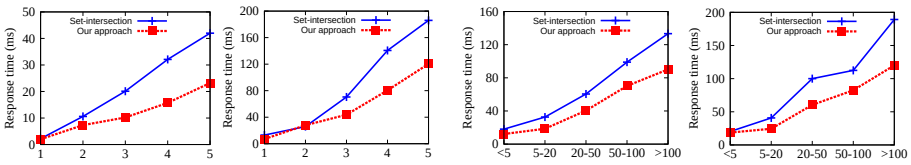
5.6 Efficiency evaluation

The response time of our approach and Set-intersection [15] is shown in Figure 12, 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.

6 Related work

LCA-based approaches. XKSearch [14] defines Smallest LCAs (SLCAs) to be the LCAs that do not contain other LCAs. Meaningful LCA (MLCA) [12] incorporates SLCA into XQuery. VLCA [9] and ELCA [16] introduces the concept of valuable/ exclusive LCA to improve the effectiveness of SLCA. XReal [1] proposes an approach based on Information Retrieval techniques. MESSIAH [13] handles cases of missing values in optional attributes. Recently, XRich [7] 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 [2], distinct root [4] and subgraph [10, 5]. Later, [8]



(a) No. of keywords (Basketball) (b) No. of keywords (IMDb) (c) No. of matching nodes (Basketball) (d) No. of matching nodes (IMDb)

Figure 12: Efficiency evaluation

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 [10, 5] can provide more answers by following IDREFs. However, those graph-based 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.

7 Conclusion

We have argued that meaningful answers can be found beyond common ancestors and when users issue a query, their expectations are independent from the schema designs. Based on these arguments, 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. We proposed an approach to find answers based on the CR semantics in which we introduced properties of relatives and designed an algorithm to find relatives of a node effectively and efficiently. Experimental results showed that our 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. 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.
3. S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, 1971.
4. H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD*, 2007.
5. M. Kargar and A. An. Keyword search in graphs: finding r-cliques. *PVLDB*, 2011.
6. 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.
7. T. N. Le, T. W. Ling, H. V. Jagadish, and J. Lu. Object semantics for XML keyword search. In *DASFAA*, 2014.
8. 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.
9. G. Li, J. Feng, J. Wang, and L. Zhou. Effective keyword search for valuable LCAs over XML documents. In *CIKM*, 2007.
10. 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.
11. L. Li, T. N. Le, H. Wu, T. W. Ling, and S. Bressan. Discovering semantics from data-centric XML. In *DEXA*, 2013.

12. Y. Li, C. Yu, and H. V. Jagadish. Schema-free XQuery. In *VLDB*, 2004.
13. 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.
14. Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD*, 2005.
15. 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.
16. R. Zhou, C. Liu, and J. Li. Fast ELCA computation for keyword queries on XML data. In *EDBT*, 2010.