THE NATIONAL UNIVERSITY
of SINGAPORE

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

TRB4/13

From Structure-based to Semantics-based Approach for Effective XML Keyword Search

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

April 2013

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

Foreword

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

OOI Beng Chin
Dean of School

From Structure-based to Semantics-based Approach for Effective XML Keyword Search

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

[#] National University of Singapore
{*ltngoc,lingtw,luochen*}*@comp.nus.edu.sg*
[*] Institute for Infocomm Research, Singapore
huwu@i2r.a-star.edu.sg
[†] Renmin Univesity of China
jiahenglu@ruc.edu.cn

Abstract. Existing XML keyword search approaches can be categorized into tree-based search and graph-based search. Both of them are structure-based search because they mainly rely on the exploration of the structural features of document. Those structure-based approaches cannot fully exploit hidden semantics in XML document. This causes serious problems in processing some class of keyword queries. In this paper, we thoroughly point out mismatches between answers returned by structure-based search and the expectations of common users. Through detailed analysis of these mismatches, we show the importance of semantics in XML keyword search and propose a semantics-based approach to process XML keyword queries. Particularly, we propose to use Object Relationship (OR) graph, which fully capture semantics of object, relationship and attribute, to represent XML document and we develop algorithms based on the OR graph to return more comprehensive answers. Experimental results show that our proposed semantics-based approach can resolve the problems of the structure-based search, and significantly improve both the effectiveness and efficiency.

1 Introduction

Current approaches for XML keyword search are structure-based because they mainly rely on the exploration of the structure of XML data. They can be classified into the tree-based and the graph-based search. The tree-based search is used when an XML document is modelled as a tree, i.e. without ID References (IDREFs) such as [8, 21, 16, 12, 22], while the graph-based search is used for XML documents with IDREFs such as [9, 11, 3, 13, 4]. Due to the high dependence on hierarchical structure and unawareness of real semantics in XML data, these approaches suffer from several serious limitations as illustrated in the following example.

Consider an XML keyword query $Q = \{$`Bill, John`$\}$ issued to the XML data in Fig. 1, in which the query keywords match first name of two students. Let us discuss answers for this query returned by the LCA-based (Lowest Common Ancestor) approach, a representative of the tree-based search. The LCA-based approach returns the document root as an answer for Q, which is intuitively meaningless for users. Suppose we could tell that two objects are the same if they belong to the same object class and have the same object identifier (ID) value. Then `Course_(11)` and `Course_(35)` refer to the same object `Course CS5201` because they belong to

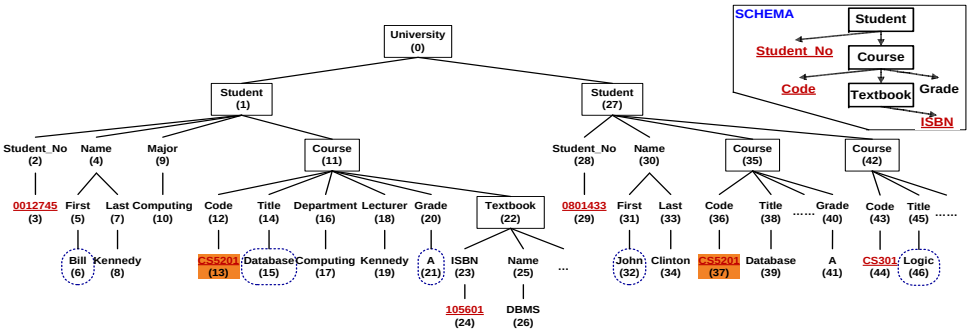


Fig. 1: university.xml

the same object class `Course` and have the same object ID value `CS5201`. `Course CS5201` is the common course taken by both students `Bill` and `John` and should be an answer. The LCA-based approach miss this answer because of unawareness of object, object ID and duplication of the same object. Thus, the common courses taken by both students are not recognized. In fact, other approaches of the tree-based and graph-based search suffer from similar problems, which will be demonstrated in Sec. 3 and 4.

The question we consider in this paper is that besides values, what benefits we can derive from paying attention to the role of tags in XML document, particularly for the problem of keyword queries, which are not used in existing works. For this purpose, we introduce ORA-semantics and exploit it for XML keyword search. ORA-semantics stands for semantics of *Object*, *Relationship* and *Attribute* derived from XML tags and values. Once an XML document is defined with ORA-semantics, we can develop effective semantics-based approach for XML keyword search which can solve limitations of the structure-based search.

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

- We illustrate the limitations of both types of the structure-based search (the tree-based and the graph-based) in XML keyword search in details (Sec. 3 and 4).
- We introduce ORA-semantics and show that ORA-semantics plays an important role to effectively process XML keyword queries. Thus we propose semantics-based approach for XML keyword search, in which we use OR graph, which can capture full ORA-semantics, to represent XML document (Sec. 5 and 6).
- We perform comprehensive experiments to compare our semantic-based approach with the structure-based approaches (including XKSearch [21] and BLINK [9]). Experimental result shows the significant superiority of our methods because it can solve limitations of the structure-based search efficiently (Sec. 6).

2 Background and terminology

This section presents concepts of object, relationship and attribute used in this paper and use the XML data in Fig. 1 for illustrating examples.

Concept 1 (Object) *In an XML data tree, an object is represented by a group of nodes, starting at an object class name, followed by a set of attributes and their associated values. Each object belongs to an object class and has a unique object ID value.*

Object ID is usually single with only one attribute. However, we still have algorithms to discover a composite object ID of several attributes [14].

Concept 2 (Object node vs. non-object node) *Among nodes describe an object, the one w.r.t. an object class is called object node and all remaining nodes are called non-object nodes. Each non-object node is associated with a corresponding object node.*

For example, `Student_(1)` is an object node whereas `Name_(4)` is a non-object node belonging to object node `Student_(1)`.

Concept 3 (The same object) *Two objects are the same if they belong to the same object class and have the same object ID value.*

For example, `Course_(11)` and `Course_(35)` refer to the same object because they belong to the same object class `Course` and they have the same object ID value `CS5201`. Multiple object nodes that refer to the same object are usually identical. If we find two nodes that are not identical, they are still considered as referring to the same object as long as they are of the same object class and have the object ID value.

Concept 4 (Relationship) *Objects may be connected through some relationship which can be explicit or implicit. An explicit relationship explicitly appears in an XML data as a node, whereas an implicit relationship is reflected by the nesting between two or more objects.*

For example, in Fig. 1, there is no explicit relationship but several implicit relationships such as relationships between `Student_(1)` and `Course_(11)`.

Concept 5 (Attribute) *An attribute can be an object attribute or a relationship attribute. In XML data, it can be a child of an object node, a child of an explicit relationship node or a child of the lowest object of an implicit relationship node.*

For example, `Lecturer` is an attribute of object class `Course` whereas `Grade` is a relationship attribute of an implicit relationship between a `Student` and a `Course`.

3 Revisiting the tree-based XML keyword search

Since almost all tree-based approaches are based on LCA semantics such as SLCA [21], VLCA [12], ELCA [22], we use the LCA-based approach as a representative of the tree-based search onward. In this section, we systematically point out limitations of the LCA semantics by comparing answers returned by the LCA semantics and answers that are probably expected by users. We use the XML data in Fig. 1 for illustration. It is worthy to note that `Course_(11)` and `Course_(35)` refer to the same object `Course CS5201` despite of appearing as different nodes because they belong to the same object class `Course` and they have the same object ID value `CS5201`.

3.1 A meaningless answer

Example 3.1. $Q_{3.1}$ = {`Bill`}.
LCA answer. The LCA-based approach returns node `Bill_(6)`. However, this is not useful since it does not provide any supplementary information about `Bill`. This happens when a returned node is a non-object node, e.g., an attribute or a value.
Reason. The LCA-based approach cannot differentiate object and non-object nodes. Returning object node is meaningful whereas returning non-object node is not.
Expected answer. The expected answer should be forced up to `Student_(1)`, the object w.r.t. to `Bill_(6)` since it contain supplementary information related to `Bill`.

3.2 Missing Answer

Example 3.2. $Q_{3.2}$ = {Bill, John}.
LCA answer. The LCA-based approach returns the document root which is definitely not meaningful.
Reason. Due to unawareness of semantics of object, the LCA-based approach can never recognize that, Course_(11) and Course_(35) refer to the same object Course CS5201. This is the common course taken by the two students Bill and John.
Expected answer. The expected answer should be the common course taken by these two students, i.e., Course CS5201 appearing as Course_(11) and Course_(35).

3.3 Duplicated answer

Example 3.3. $Q_{3.3}$ = {CS5201, Database}.
LCA answer. Two answers Course_(11) and Course_(35) of this query are duplicated because the two nodes refer to the same object Course CS5201.
Reason. Similar to Example 3.1, this problem is caused by the unawareness of duplication of object having multiple occurrences.
Expected answer. Either of Course_(11) or Course_(35) should be returned, but not both since they are different occurrences of the same object Course CS5201.

3.4 Problems related to relationships

Example 3.4. $Q_{3.4}$ = {Database, A}.
LCA answer. The LCA-based approach returns Course_(11) and Course_(35) as answers. These answers are incomplete because 'A' grade is not an attribute of a course, but it is grade of a student taking the course instead. On the other hand, Grade is a relationship attribute between Student and Course, not an object attribute.
Reason. The LCA-based approach cannot distinguish between an object attribute and a relationship attribute under an object node.
Expected answer. The proper answer should be all students taking course Database and getting an 'A' grade. To do that, the answer should be moved up to contain other objects (e.g., students) participating in the relationship that 'A' grade belongs to.

3.5 Schema dependence

There may be several designs for the same data source. The XML data in Fig. 1 can be represented by another design as in Fig. 2 with different hierarchical structure among object classes, e.g., Course becomes the parent of Student.
Example 3.5. $Q_{3.5}$ = {Bill, Database}.
LCA answer. With the design in Fig. 1, the LCA-based approach returns Student_(1). With the design in Fig. 2, Course_(1) is returned. As shown,

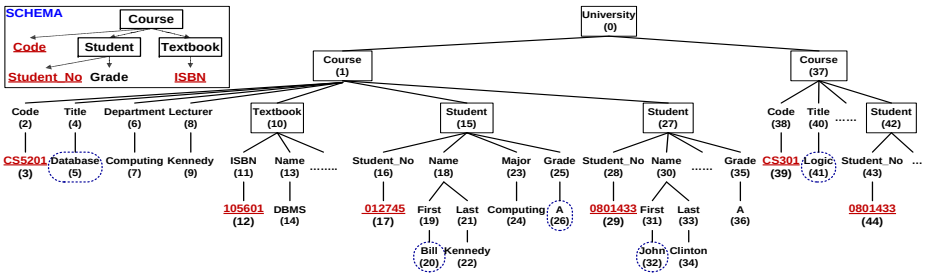


Fig. 2: Another design for the XML data in Fig. 1

answers for different designs are different though these designs refer to exactly the same information and we are dealing with the same query.

Answers of other queries related to more than one object also depend on XML hierarchical structure. Let us recall $Q_{3.2}$ = {Bill, John}, $Q_{3.4}$ = {Database, A} and discuss the answers from the design in Fig. 2 for these queries. For $Q_{3.2}$, Course_(1) is an answer. For $Q_{3.4}$, Course_(1) is also returned. Compared with the answers the root for $Q_{3.2}$ and Course_(11) and Course_(35) (without students as their children) for $Q_{3.4}$ from the design in Fig. 1, the ones from the design in Fig. 2 are different.

Reason. Answers from the LCA semantics rely on the hierarchical structure of XML data. Different hierarchical structures may provide different answers for the same query.

Expected answer. Users issue a keyword query without knowledge about the underlying structure of the data. Thus, their expectation about the answers are independent to the schema design. Therefore, the expected answers should also be semantically the same with all designs of the same data source.

Summary. The main reasons of the above problems are the high dependence of answers returned by the LCA-based search on the hierarchical structure of XML data (e.g., $Q_{3.5}$, $Q_{3.2}$, $Q_{3.4}$), and the unawareness of semantics of object, relationship and attribute. Particularly, unawareness of objects causes *missing answers* (e.g., $Q_{3.2}$), and *duplicated answer* (e.g., $Q_{3.3}$) because the LCA-based approach cannot discover the same object. Unawareness of object and attribute cause *meaningless answer* (e.g., $Q_{3.1}$) because it cannot differentiate XML elements. Unawareness of relationship and attribute cause the *problems related to relationship* (e.g., $Q_{3.4}$) because of it is unable to differentiate an object attribute and a relationship attribute.

4 Revisiting the graph-based XML keyword search

The graph-based search can be applied for both XML tree (without IDREF) and XML graph (with IDREFs). In the absence of IDREF, the graph-based search suffers from the same problems as the LCA-based search does. Under ID Reference mechanism, object and object ID are observed and utilized. An object can be referenced by an IDREF, which has the same value with its object ID to avoid duplication. This helps the graph-based search handle some but not all problems of the LCA-based search. Particularity, the *problems related to relationship* and *meaningless answers* cannot be solved no mater IDREF is used or not. The other problems including *missing answer*, *duplicated answer* and *schema dependence* can be solved if the ID Reference mechanism applies to all objects. Otherwise (there exist some objects without IDREF), they cannot be solved totally.

For generality, in this section, we use the XML data in Fig. 3 which contains both objects with and without IDREF to illustrate problems of the graph-based search. We apply the widely accepted semantics *minimum Steiner tree* [6, 7] for illustrating the problems. In the XML data in Fig. 3, Object Employer HT08 is duplicated with two occurrences Employer_(6) and Employer_(26). Ternary relationship type among Supplier, Project and Part means suppliers supply parts to projects. Quantity is an attribute of this ternary relationship and represents the quantity of a part supplied to a project by a supplier. Besides, binary relationship between

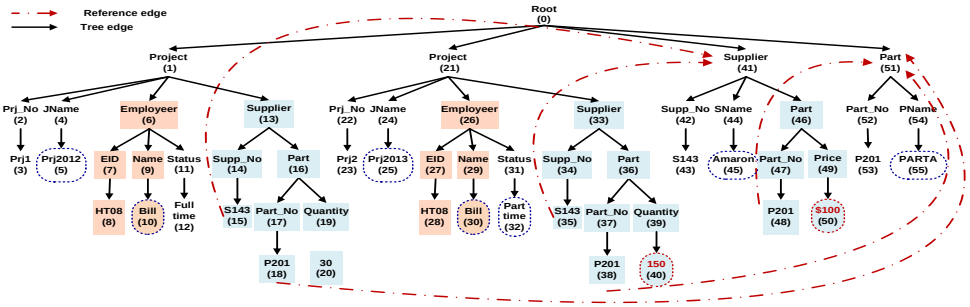


Fig. 3: An XML document with IDREFs

`Supplier` and `Part` has a relationship attribute `Price` to represent the price of a part supplied by a supplier.

4.1 Problems cannot be solved with IDREF

IDREF mechanism is aware of semantics of object and object ID. However, the semantics of relationship and attribute is still not recognized and utilized which cause the problems of *meaningless answer*, and *problems related to relationship*.

Meaningless answer. Unawareness of attribute and not differentiating object and non-object nodes cause meaningless answer when the returned node is a non-object node. For example, for $Q_{4.1}$ = {`Amazon`}, the answer is only `Amazon_(45)` without any other information.

Problems related to relationships. Without semantics of relationship, the graph-based search cannot distinguish object attribute and relationship attribute, and cannot recognize n-ary ($n \geq 3$) relationship. These cause problems related to relationship.

For example, for $Q_{4.2}$ = {`PARTA, 100`}, the subtree rooted at `Part_(46)` is an answer. However, this is not complete since price `100` is the price of a part named `PARTA` supplied by `Supplier_(41)`. It is not the price of `Part_(46)`. Thus, the answer should be moved up to `Supplier_(41)` to include `Supplier_(41)` as well.

For another example related to ternary relationship, $Q_{4.3}$ = {`PARTA, 150`}, the answer is the subtree rooted at `Part_(36)`. This is not complete either since `150` is the quantity of a part named `PARTA` supplied by `Supplier_(41)` to `Project_(21)`. `Quantity` is not an attribute of object `Part_(46)`. Thus, the answer should be moved up to `Project_(21)` to include `Project_(21)` and `Supplier_(41)`.

4.2 Problems can be solved with IDREF

IDREF mechanism is based on semantics of object and object ID, thus using IDREF can avoid problems caused by lack of semantics of object, including the problems of *missing answer*, *duplicated answer* and *schema dependence*. For example, suppose that we use IDREF to refer to object `Employer HT08`, i.e., only one object node `Employer` under the root, to which there are two IDREFs from `Project_(1)` and `Project_(21)`. Then, answers for $Q_{4.4}$ = {`Bill, HT08`} is the only object `Employee HT08` without duplication. For $Q_{4.5}$ = {`Prj2012, Prj2013`}, by following IDREFs, both expected answers which are the subtree containing `Supplier_(41)` (and the matching nodes), and the subtree containing `Employee HT08` (and the matching nodes) are returned.

However, if ID Reference mechanism is not totally applied for all objects, i.e., there exists some objects without IDREF as object Employee HT08 in Fig. 3, then the above problems are not totally solved. For example, $Q_{4.4}$ = {Bill, HT08} has two duplicated answers, Employer_(6) and Employer_(26). For $Q_{4.5}$ = {Prj2012, Prj2013}, only the subtree containing Supplier_(41) can be returned whereas the subtree containing Employee HT08 is *missed*. If object class Employee is designed as the parent of object class Project, the missing answer of $Q_{4.5}$ are found. It shows that the graph-based search also *depends on the design of XML schema* in this case.

Summary. The graph-based search can avoid *missing answer*, *duplicated answer* and *schema dependence* only if the ID reference completely covers all objects. Otherwise, the above limitations cannot avoid. The other problems including *meaningless answer* and *problems related to relationship* are still unsolved no matter IDREFs are used or not because IDREF mechanism only considers semantics of object and object ID but ignores semantics of relationship and attribute.

5 Impact of ORA-semantics in XML keyword search

We pointed out limitations of the structure-based search (the LCA-based and the graph-based approaches) because of unawareness of identification of object, relationship and attribute. We refer such identification as ORA-semantics. This section introduces ORA-semantics, shows the impact of ORA-semantics in XML keyword search and discusses the way to discover ORA-semantics from XML schema and data.

5.1 ORA-semantics

The term *semantics* has different interpretations. In this paper, we define the concept of ORA-semantics to include the identification related to *object*, *relationship* and *attribute*. At data level, an *object* represents a real world entity. Several objects may be connected through some *relationship*. Objects and relationships may have a set of *attribute values* to describe their properties. Object, relationship and attribute value is an instance of *object class*, *relationship type* and *attribute* respectively at the schema level. Besides such major semantics, there are connecting nodes such as composite attributes or aggregation nodes. In brief, ORA-semantics is defined as follows.

Concept 6 (ORA-semantics) *ORA-semantics includes the identification of object, relationship and attribute values in XML document and includes the identification of object class, relationship type and object/relationship attribute in XML schema.*

For example, at schema level, ORA-semantics of the schema in Fig. 1 includes:

- Student, Course and Textbook are object class
- Student_No, Code and ISBN are object ID of the above object classes.
- Grade is the attribute of the relationship between Student and Course.
- For simplicity, we do not include object attribute in the schema in Fig. 1. The hidden object attributes include Student_No, Name, etc of object class Student.

At data level, ORA-semantics in the XML data in Fig. 1 includes that Course_(11) is object; CS5201 is its object ID value; Database, Computing are its attribute values; especially A is an attribute value of the relationship it involves in, etc.

5.2 Impact of ORA-semantics in XML keyword search

Recall $Q_{3.2}$ = {Bill, John} to the XML data in Fig. 1 studied in Sec. 3.2, the LCA-based approaches return the document root, a meaningless answer, since they cannot find the common course taken by both students *Bill* and *John*. One possible structured XQuery query that correctly searches for desired answers for that query is as follows.

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

In this XQuery query, we need to join two paths `$s1/course/code` and `$s2/course/code`, based on the same `Code` value. In order to compose this query, we need to know the *semantics* that `Course` and `Student` are *object class*, `Code` is the *object ID* of `Course` and there is a *relationship* between a `Student` and a `Course`. Without such ORA-semantics, a user cannot issue the above XQuery query. However, in XML keyword search, a keyword query conveys little semantics. Thus, it is desirable that the engine exploits ORA-semantics to find more accurate answers. Otherwise, the answer may be unsatisfactory, as illustrated in Sec. 3 and 4. In brief, no matter what forms of queries, ORA-semantics is necessary to guarantee the correctness of answers. Follows are advantages of ORA-semantics in solving problems in Sec. 3 and Sec. 4.

Object semantics. Object identification helps detect multiple occurrences of the same object (duplicated object) appearing at different places in XML document. This enables us to filter *duplicated answers* and discover *missing answers*.

Relationship semantics. Relationship identification helps discover the degree of a relationship to return a more complete answer for queries involving in *ternary relationship*.

Attribute semantics. Differentiating object attribute and relationship attribute avoids returning incorrect answer for queries involving in *relationship attribute*. Moreover, differentiating object and non-object node also avoids *meaningless answers*.

ORA-semantics. Exploiting ORA-semantics in processing keyword query provides answers independent from the schema designs. In brief, ORA-semantics can resolve all problems of the structure-based search discussed in Sec. 3 and Sec. 4.

5.3 Discovering ORA-semantics

ORA-semantics is usually available to the search engine, because most XML data are initially designed based on such semantics, which is similar to the ER design for relational databases. If such semantics is not supported, some works [18, 19] attempt to discover such semantics from XML schema and data. We also have designed algorithms to automatically discover such semantics with high accuracy for overall process (higher than 94%) [14]. More details can be found in [14]. Other effective algorithms can be studied, however, this task is orthogonal to this paper.

6 Our semantics-based XML keyword search

Discussions in Sec. 4 show that even if all objects follow IDREF mechanism and thus there is no duplication, the existing graph-based search still suffers from problems of *meaningless answer*, and especially *problems related to relationship*. In brief, IDREF cannot solve all problems of the existing structure-based search. To solve these

problems, a more semantics-enriched model is needed, in which not only objects, but relationships and their attributes must be fully captured as well. We illustrate this model by proposing Object Relationship (OR) graph to represent an XML document. Both objects and relationships are represented as nodes in an OR graph while attributes and values are associated with the corresponding objects and relationships. As such, an OR graph can capture all objects, relationships and attributes of an XML data.

Object orientation has been demonstrated as a concept of value in many areas of computation, and is widely used today. Even in the database field, there was a movement towards object-oriented databases and many object-oriented notions were adopted by commercial relational databases, even using the term Object Relational for a time. In the same spirit, we propose OR graph to capture objects and relationships (with their attributes) in XML documents, without making any major modifications to XML itself.

In this section, we first introduce the OR graph, its advantages, and the process of OR graph generation. We then present search semantics, i.e., formally define the expected answers, and query processing based on the OR graph. To show the advantages of the OR graph, we compare our OR graph based search with the structure-based search. To improve efficiency, we propose indexes and optimized search algorithms.

6.1 Object relationship (OR) graph

Definition 1 (OR graph) *An OR graph* $G = (V_O, V_R, E)$ *is an unweighed, undirected* ***bipartite*** *graph with two types of nodes, where* V_O *is the set of* object nodes*,* V_R *is the set of* relationship nodes*,* $V_O \cap V_R = \emptyset$*, and* E *is the set of* edges*. An edge is between an object node in* V_O *and a relationship node in* V_R*. For every relationship* $r \in V_R$*, there is an edge between each of its participating objects and* r*.*

The information stored for each object node is a quadruple ⟨*nodeID, object class, OID, associated keywords*⟩. The information stored for each relationship node is a similar quadruple ⟨*nodeID, relationship type,* {*participating* o_i}*, associated keywords*⟩. Each object/relationship has a randomly generated *nodeID* as label in the OR graph. *Associated keywords* of an object/relationship include its attributes and values as well.

Consider the XML data in Fig. 3. The OR Graph conforming with this XML data is shown in Fig.4, where *square* and *diamond* stand for *object node* and *relationship node* respectively. Each object/relationship node in an OR graph has an identifier, e.g., o_1, o_2, o_3, o_4 and o_5 are objects and r_1, r_2, r_3, r_4 and r_5 are relationships. Additionally, to serve for a clearer explanation, for each object node in Fig.4, we associate its object class and object ID (e.g., `Project Prj1` for o_1). Moreover, we also show matching nodes of query keywords discussed in Sec. 4 (e.g., `Amazon` matches o_4).

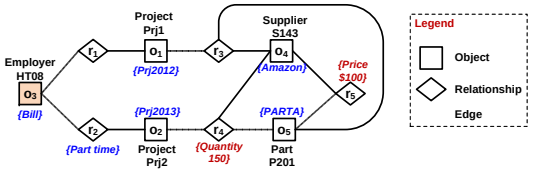


Fig. 4: The OR graph w.r.t. the XML data in Fig. 3

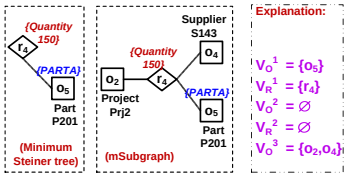


Fig. 5: $mSubgraph$ for $Q_{4.3}$

6.2 Features and advantages of the OR graph

FA1. Object-relationship level. Our OR graph represents XML document at object-relationship level by associating attributes and values with their corresponding objects and relationships. This can significantly *reduces the search space* since the number of nodes of an OR graph is much less than that of the corresponding XML data due to not counting attributes, values and duplicated objects. Moreover, it can avoid *meaningless answers* because an answer must correspond to a whole object or relationship rather than an arbitrary XML element.

FA2. Duplicate-free. An object may appear as multiple occurrences in an XML document. In an OR graph, each object node represents an *object*, not an occurrence of object. Thus, an object is not duplicated because it corresponds to only one object node. Duplicate-free is a very important feature of OR graph, by which the process can find *missing answers* and avoid *duplicated answers* from the LCA-based approach.

FA3. Explicit appearance of relationships. Differentiating object nodes and relationship nodes enables us to distinguish between object attribute and relationship attribute. This avoids returning incorrect answers for queries involving relationship attribute. Moreover, relationship nodes can provide the degree (e.g., binary or ternary) of a relationship. As a result, we can add *all participating object nodes* of relationship nodes in a returned answer to make it more meaningful.

FA4. Schema-independence. The OR graphs conforming with any XML representations of the same data source are the same no matter which schema design is used. In other words, the OR graph is *independent* of the XML structure. This enables us to return the same answers for all designs of the same data source. For example, no matter `Paper ER32` in the data in Fig. 3 is designed with or without IDREF, the OR graphs conforming is in Fig. 4 and `Paper ER32` is an answer of $Q_{4.2}$={`Bill`, `John`}.

6.3 Generating OR graph from an XML document

Algorithm 1: OR graph generation

```
   Input: D: XML document
   Output: OR graph G(V_O, V_R, E)
1  V_O ← ∅
2  V_R ← ∅
3  E ← ∅
4  for each object o visited by DF order do
5      if o is not duplicated with objects in V_O then
6          V_O.Add(o)
7      Rel(o) ← relationships which o participates in
8      for each relationship r in Rel(o) do
9          if r is not duplicated with relationships in V_R then
10             V_R.Add(r)
11             e ← edge between o and r
12             E.Add(e)
```

To generate OR graph $G(V_O, V_R, E)$ from an XML document D (with or without IDREFs), we identify V_O, V_R, E from nodes in D. Only distinct object nodes in D are added to V_O. The process of OR graph generation is presented in Algorithm 1. We traverse XML document by Depth First (DF) order. For each visited object node o, we add o to V_O if o is not duplicated with any existing object in V_O and find the the set of relationships $Rel(o)$ in which o participates. For each relationship r in $Rel(o)$, we add r to V_R if r is new. We finally add the edge between o and r to E. In brief, the sequence is adding an object, then a relationship it participates in, and finally the edge between them.

Fig. 6 shows steps of generating OR graph from the XML data in Fig. 1. Each step corresponds to a loop started at line 4 in Algorithm 1 when an object node is visited.

By Depth First order, object nodes are visited with order as follows: `Student_(1)`, `Course_(11)`, `Textbook_(22)`, `Student_(27)`, `Course_(35)` and `Course_(42)`. Note that, `Course_(11)` and `Course_(35)` are duplicated. Thus, when `Course_(35)` is visited, it is not added to V_O as shown in Step (e).

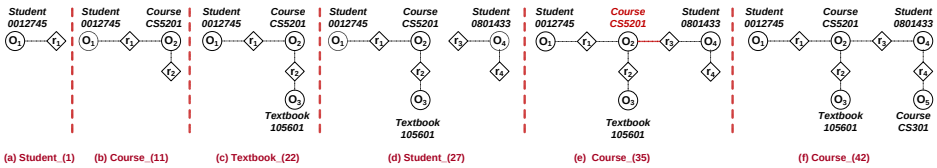


Fig. 6: Steps of generating OR graph from the XML data in Fig. 1

6.4 Search semantics

This paper adopts the *minimum Steiner tree semantics* [6, 7] because it is the widely accepted semantics for the graph-based search. A Steiner tree is a subtree of the data graph that contains all keywords. The weight of a Steiner tree is defined as the total weight of its edges. A *minimum* Steiner tree is a Steiner tree having the smallest weight among all the Steiner trees w.r.t. the same set of matching nodes. However, a subtree returned by this semantics may not contain completely meaningful information to answer a query, especially when XML data has complicated structure. Therefore, we determine what other nodes should be added in order to return a more meaningful answer to users. Consequently, we extend a minimum Steiner tree s to become a more meaningful subgraph (called $mSubgraph$) by *adding all participating objects* of all relationships in s. The rationale is that a relationship itself has no or incomplete meaning without all of its participating objects. An answer $mSubgraph$, is defined as follows.

Definition 2 (mSubgraph) *Given a keyword query Q to the OR graph G = (V_O,V_R,E). An answer mSubgraph of Q is a subgraph of G and is denoted as ${}_mS({}_mV_O,\ {}_mV_R,\ {}_mE)$, where ${}_mV_O = V_O^1 \cup V_O^2 \cup V_O^3$ and ${}_mV_R = V_R^1 \cup V_R^2$ such that*

- *V_O^1 and V_R^1 are the sets of matching objects and relationships.*
- *V_O^2 and V_R^2 are the sets of intermediate objects and relationships connecting objects in V_O^1 and relationships in V_R^1.*
- *V_O^3 is the set of added objects which participate in ${}_mV_R$ but are not in $V_O^1 \cup V_O^2$.*
- *${}_mE$ is the set of edges between each object in ${}_mV_O$ to its relationships in ${}_mV_R$.*

Let us recall $Q_{4.3}$ = {`PARTA, 150`} to illustrate the benefits of $mSubgraph$. Answers for this query (both in form of a minimum Steiner tree and in form of an $mSubgraph$) are shown in Fig. 5. As can be seen, $mSubgraphs$ are more meaningful than minimum Steiner trees with intuitive meaning of `150` parts name `PARTA` are supplied by `Supplier S143` to `Project Prj1` while the minimum Steiner tree does not provide information about the supplier and the project.

6.5 Query processing and output presentation

This work aims to show the impact of ORA-semantics on effectiveness of XML keyword search. The ORA-semantics is exploited in generating the OR graph. After the generation process, searching over OR graph to find minimum Steiner trees can be implemented by existing efficient algorithms. Answers then will be extended to $mSubgraphs$. Space limitation precludes detailed discussion about their algorithms. However, readers may find them in [6, 7]. We also design our own algorithm and propose index and an optimized techniques to improve the efficiency in Sec. 7.

Output presentation. Since a keyword query is issued against an XML document, we use the corresponding XML schema to convert $mSubgraphs$ into XML fragments.

6.6 Comparison on the LCA, graph and semantics based approaches

We compare the limitations of the LCA-based, the graph-based and our OR graph based search and show the reasons behind in Table 1.

Table 1: Comparison on the LCA-based, graph-based and OR graph based approaches

Problem	LCA-based	Graph-based	Reason of the problems of the structure-based search	Semantics needed	OR graph based	Reason that the semantic based search can avoid problems
Meaningless answer	Yes	Yes	Returning non-object element because of not differentiating object and non-object node	-Object -Attribute	No	All attributes and values are associated with objects/relationships (Object-relationship level, FA1)
Missing answer	Yes	Partial	- Unable to discover the same object appearing at different places in XML document	Object	No	The same objects can be discovered (Duplicate-free, FA2 of OR graph)
Duplicated answer	Yes	Partial	- Partial because it can be solved with IDREF			
Problems related to relationships	Yes	Yes	- Unable to distinguish relationship attribute and object attribute - Not aware ternary relationships and above	-Attribute -Relationship	No	Relationships and relationship attributes are discovered (Explicit appearance of relationships, FA3 of OR graph)
Schema dependence	Yes	Partial	- Relying on hierarchy (LCA) - Partial because it can be solved with IDREF	Object	No	Different designs of the same data have the same ORA-semantics (Schema-independence, FA4)

7 XSOR: our XML keyword Search system with OR graph

In this section, we present an overview of XSOR, our OR graph based system for XML keyword search in Section 7.1, followed by technique details in Section 7.2 to 7.3.

7.1 Overview of XSOR

Framework. A high-level representation of XSOR framework is shown in Fig. 7. XSOR has four main components: (1) discovering ORA-semantics, (2) indexing, (3) OR graph generation, and (4) searching. The first three components is for each XML document while the last one is for each keyword query. In the discovering ORA-semantics component, XSOR discovers semantics of objects, relationships and attributes, referred to as ORA-semantics, from XML data and schema (if any). In the indexing component, XSOR maintains indexes to facilitate the searching. In the OR graph generation component, the OR graph is automatically generated from the XML document. Actually, it is derived from the semantic-inverted index which captures ORA-semantics corresponding to each XML document nodes. In the searching

component, XSOR searches over the OR graph to find answers for an XML keyword query. XSOR uses indexes maintained in the second component to facilitate the searching.

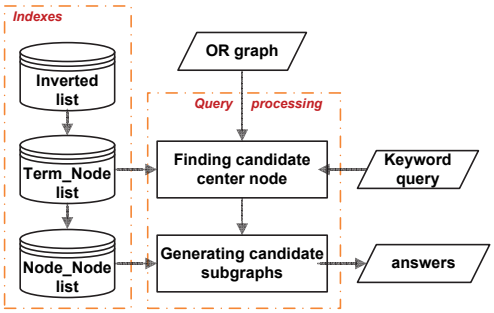


Fig. 7: The framework of XSOR

Query processing. Given a keyword query $Q = \{k_1, k_2, \ldots, k_n\}$, let $mNode(k_i)$ be the set of nodes matching keyword k_i. XSOR aims to find top-k *mSubgraphs*, each of which contains all keywords and is ranked by the *size* of the subgraphs, where the size is the number of edges. Let $Ans(Q)$ be the list of subgraphs returned as answers to Q. Recall that a returned mSubgraph $g =_m S({}_mV_O,\ {}_mV_R,\ {}_mE) \in Ans(Q)$ is defined in Section 6.4.

Among all nodes in g, one node, e.g., node v, $v \in_m V_O \cup {}_mV_R$, is considered as a *connecting node*, which plays the role of connecting all nodes in $V_O^1 \cup V_R^1$. Given a query Q which has m instances, theoretically, a connecting node v can connect all nodes in each instance of Q to generate a returned subgraph. However, not all m instances of Q give good answers. XSOR aims to find only a number l of instances of Q, where $l \leq m$, and $l \leq k$ (k in top-k), so that l generated subgraphs are in *current* top-k answers. The phase of identifying a node v as a connecting node is called *keyword expansion* and the phase of finding a number l of instances of Q is called *keyword navigation*.

7.2 Indexes of XSOR

To make distinction between query keywords with keywords appearing in nodes (values and tags) in an XML document, we call the latter as *term*.

Semantic-inverted list XSOR works at object and relationship level which means that each query keyword or document term is associated to the corresponding objects and/or relationships. Each term t in an XML document is indexed with two sets: a set of matching objects *O_Set(t)* and a set of matching relationships *R_Set(t)*. Each entry $o_i \in O\_Set(t)$ is a triplet $\langle$*nodeID, object class, OID*$\rangle$. Each $r_i \in R\_Set(t)$ is a triplet $\langle$*nodeID, relationship type,* $\{$*participating* $o_i\}\rangle$. Each object/relationship has a randomly generated *nodeID* to represent the corresponding node in the OR graph.

Following is the way XSOR exploits ORA-semantics to create the semantic-inverted index. Each document term (attribute value, attribute) is co-related with an attribute att from *A_List*. XSOR gets object class $ObjCls$ or relationship type $RelTyp$ to which att belongs. If att is an object attribute, XSOR then gets the OID of $ObjCls$ from *O_List*. Otherwise, XSOR then gets all participating objects of $RelTyp$ from *R_List*.

Term-node lists Term-node lists are pre-computed to speed up the keyword expansion. For each term t in an XML document, there is a corresponding term-node list $L_{TN}(t)$. Each entry e in $L_{TN}(t)$ is a quadruple $\langle dist, source, first, dest\rangle$ to represent the shortest distance $dist$ from node $source$ to node $dest \in mNode(t)$ via the first node $first$. Intuitively, e stores the shortest distance from a node $source$ in the OR graph (conforming from the XML document) to $mNode(t)$, i.e., the shortest distance from $source$ to any node in $mNode(t)$. $L_{TN}(t)$ is increasingly ordered by $dist$ of entries. For example, term-node lists corresponding to term *Brown* and term *Database* of the OR graph in Fig. 4 are given in Fig. 8. The term-node lists are similar to the keyword node lists in BLINKS [9]. However, as the ways BLINKS and XSOR expand keywords are different, entries of a term-node list in XSOR are also different from entries of a keyword node list in BLINKS.

L_{TN}(Brown) (0, o_2, o_2, o_2)→(0, o_4, o_4, o_4)→(1, r_2, r_2, o_2)→(1, r_3, r_3, o_2)→(1, r_4, r_4, o_4)→……
L_{TN}(Database) (0, o_2, o_2, o_2)→(0, o_3, o_3, o_3)→(1, r_2, r_2, o_2)→(1, r_3, r_3, o_2)→(1, r_1, r_1, o_3)→……

Fig. 8: Term-node lists of term *Brown* and term *Database*

Node-node lists In the process of keyword navigation, once a new connecting node v is found, l subgraphs which are expected to be in top-k answers are generated. To accelerate this process, for each term t in an XML document, XSOR pre-computes distances from the considered connecting node v to each node u in $mNode(t)$, and organizes these distances in a node-node list $L_{NN}(v,t)$. Intuitively, $L_{NN}(v,t)$ stores shortest distances from node v to each node matching term t, and each entry e in $L_{NN}(v,t)$ is the shortest distance from v to some node $u \in mNode(t)$. e is a triplet $\langle dist, first, dest\rangle$ where $dist, first$ and $dest$ are defined identically as in $L_{TN}(t)$. $L_{NN}(v,t)$ is increasingly ordered by $dist$ of entries. $L_{NN}(v,t)$ can be derived from $L_{TN}(t)$. However, it is pre-computed to facilitate the process of keyword navigation. Fig. 9 shows an example of node-node lists, corresponding to the term-node lists in Fig. 8, where $v = o_2$, $t_1 = Brown$, and $t_2 = Database$.

L_{NN}(o_2,Brown) (0, o_2, o_2)→(2, r_3, o_4)
L_{NN}(o_2,Database) (0, o_2, o_2)→(2, r_2, o_3) →(2, r_3, o_3)
⋮

Fig. 9: Node-node lists

Full list Full index facilitates the construction of subgraph answers. It returns the whole object or relationship which corresponds to a node in an OR graph.

7.3 Search Algorithm

Keyword expansion The process in which XSOR finds top-k answers for a keyword query Q is presented in Algorithm 2. Let $mNode(Q)$ be the set of nodes, each of which matches at least one keyword in Q, $mNode(Q) = \cup_{i=1}^{n} mNode(k_i)$, and $cNode(Q)$ be the list of connecting nodes v, ordered by the *distance* from v to $mNode(Q)$, i.e., the smallest distance among those distances from v to nodes in $mNode(Q)$. When a keyword query Q is issued, XSOR adds all nodes in $mNode(Q)$ to $cNode(Q)$ and then expands $cNode(Q)$ to all directions in the OR graph. In each step of this expansion, $cNode(Q)$ is added one node obtained from the Breadth First Search (BFS) on the OR graph. The BFS guarantees all nodes in $cNode(Q)$ are increasingly ordered by distance. For example, given query Q = {*Brown, Database*} to the OR graph in Fig. 4, $cNode(Q)$ is step-by-step expanded to the list $\langle o_2, o_4, o_3, r_2, r_3, r_4, r_1, \ldots\rangle$.

Algorithm 2: XSOR search algorithm

```
   Input: Query Q = {k_1, k_2, ..., k_n}
          Term-node lists
   Output: top-k answers in Ans(Q)
1  Variables: T_prune: pruning threshold
2  T_prune ← 0
3  for i ∈ [1, n] do
4      c_i ← new Cursor(L_TN(k_i),0)
5  while ∃j ∈ [1, n] : c_j.Next() ≠ NULL do
6      i ← pick from [1,n] by the BFS order;
7      < v > ← c_i.Next()
8      if v ≠ NULL then
9          KeywordNavigation(v);
10     if c_i.Dist() > T_prune and T_prune ≥ 0 then
11         exit and output top-k answers in Ans(Q).
```

To speed up this expansion, XSOR uses the term-node lists presented in Section 7.2. For each keyword k_i, XSOR uses a cursor c_i to traverse its term-node list $L_{TN}(k_i)$. XSOR picks an entry $\langle d, v, first, dest\rangle$ in some term-node list by the BFS order, and gets a new connecting node v (line 6, 7). $cNode(Q)$ can be expanded to contain all nodes in the OR graph if the OR graph is connected. However, once XSOR gets a new connecting node, it follows the process *keyword navigation* to find candidate answers (line 8, 9). This expansion stops when top-k answers are found (line 10, 11). XSOR keyword expansion is inspired from *backward* search [3, 9, 11]. However, backward search follows backward direction and is suitable for directed graph, while XSOR keyword expansion expands nodes to all directions and searches over an undirected graph.

Keyword navigation Once $cNode(Q)$ adds a new connecting node v, XSOR generates a number l of subgraphs which are expected to be in top-k answers as in Algorithm 3. Each subgraph g is generated from v and an instance V_K of Q, where V_K = $\{v_1, \ldots, v_n\}$, $v_i \in mNode(k_i)$ (line 6, 7, 8). These subgraphs are generated by the *Best First Search* (BestFS) order, which means that the subgraph having smallest total distances from v to all nodes in V_K is generated first. For each keyword k_i, XSOR looks up the *node-node list* $L_{NN}(v, k_i)$ to find distances from v to nodes in $mNode(k_i)$.

Let us consider Q = {*Brown, Database*} to the OR graph in Fig. 4. When o_2 is considered as a connecting node, based on the node-node lists corresponding to node o_2 in Fig. 9, step-by-step, subgraphs generated and their corresponding V_K are $g_1 : \{o_2\}$, $g_2 : \{o_2, o_3\}$ (via r_2), $g_3 : \{o_2, o_3\}$ (via r_3), $g_4 : \{o_2, o_4\}$ (via r_3) ....

A generated subgraph g to be included in the current top-k answers needs to satisfy two conditions. (1) g is new, i.e., not in current $Ans(Q)$ (line 9) and (2) its *size* is not greater than the size of the current k^{th} subgraph in $Ans(Q)$ (line 10). As the latter subgraph always has greater total distances compared with the previous ones, once a subgraph has size greater than threshold T_{prune}, XSOR stops generating subgraphs (line 13,14).

Algorithm 3: KeywordNavigation(v)

Input: a connecting node v
Node-node lists
Variables: $cNode(Q)$: connecting nodes visited, initially $\emptyset$
if $cNode(Q)$.*contain(v)* **then**
  return
$cNode(Q)$.add(v)
while *TRUE* **do**
  for $j \in [1..n]$ **do**
    $< u_j >\leftarrow$ pick from $L_{NN}(v)$ in *BestFS* order;
  $g \leftarrow \langle \{u_1, u_2, \ldots, u_n, \}, V_I, E \rangle$
  if $g \notin Ans(Q)$ **then**
    if *size(g)* $> T_{prune}$ *and* $T_{prune} \geq 0$ **then**
      return;
    $Ans(Q)$.add(g)// $size$ in ascending order
    if $|Ans(Q)| \geq k$ **then**
      $T_{prune} \leftarrow$ the k^{th} biggest of $\{size(\alpha) | \alpha \in Ans(Q)\}$

Apart from these major phases, we have some other secondary steps below to ensure that the returned answers are meaningful.

Adding extended nodes. A subgraph g may not be fully meaningful if there exists keyword matching relationship nodes. To make answers more meaningful, XSOR adds some extended nodes, which are objects participating of relationship nodes in g, to g.

Finding intermediate nodes. For each node $u \in V_K$, all nodes in the paths from the connecting node v to u are intermediate nodes. The $source$ and $dest$ fields in term-node lists correspond to v and u respectively. Using $first$ fields (via node), XSOR can trace back the path from v to u.

Generating full answers. Finally, XSOR uses *Full list* to generate a subgraph with full information, e.g., attribute and attribute values.

8 EXPERIMENTS

We compare the quality of answers returned by our OR graph based search with XKSearch [21] and BLINK [9] (two state-of-the-art algorithms to represent the LCA-based and graph-based search respectively). Using these two algorithms already show the improvement of our semantic-based search over the structure-based search because other structure-based approaches suffer from the same problems with them. The experiments were performed on a Intel(R) Core(TM)2 Duo CPU 2.33GHz with 3.25GB of RAM with three real data sets: `eBay`[1] (0.36MB), `NBA`[2](45.2MB), `DBLP`[3] (127MB).

8.1 Rating answers

We compare the quality of the answers returned by our approaches and the structure-based approaches by rating their answers. We randomly generated 25 queries from all document keywords. After filtering out some meaningless queries, we chose 10 queries

[1] www.cs.washington.edu/research/xmldatasets/data/auctions/ebay.xml
[2] http://www.databasebasketball.com/
[3] www.cs.washington.edu/research/xmldatasets/data/dblp/dblp.xml

Query	Query Keywords	Dataset
Q1	**wizbang4**	eBay
Q2	**Bill**	Player
Q3	**id_num, 511364992**	eBay
Q4	**ct-inc, CyberTech**	eBay
Q5	**Michael, Coach**	Player
Q6	**Player, Bill, Sam**	Player
Q7	**Farshad, Benjamin**	DBLP
Q8	**Farshad, article**	DBLP
Q9	**Kasidit, 8747**	XMark
Q10	**Item, item5, category**	XMark

Fig. 10: 10 keyword queries for users to rate

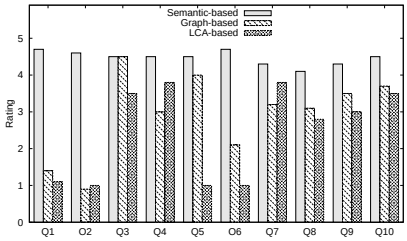


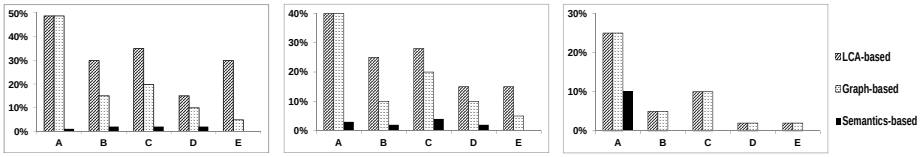
Fig. 11: Scores of answers

as shown in Table. 10. We asked 21 students major in computer science to rate the top-10 answers on scale [0-5] (0 for totally mismatch and 5 for perfectly match the user expectation). The scores of answers are shown in Fig. 11.

Discussion. As shown, our approach gets the highest scores for all queries, which means that our approach returns more meaningful answers to users. Moreover, the scores of our approach are very high (above 4) and stable, which infers that users are satisfied with our answers. In contrast, the structure-based search gets lower scores since its answers mismatch the expectations of users. Especially for queries Q_1, Q_2 and Q_6, the scores are around 1 because they returns hundreds of duplicated attribute nodes without any detailed information. Generally, the graph-based search has higher score than the LCA-based search because it can find missing answers and avoid duplicated answers when IDREFs are considered.

8.2 Statistics on problems of answers

We collected 102 keyword queries for the above three data sets: 12 queries for eBay, 44 queries for NBA and 46 queries for DBLP from 21 students working in computer science. Based on the discussion in Sec. 3 and Sec. 4, the answers which cannot match the users' search intention are classified into five categories: including *(A) Meaningless answer; (B) Missing answer; (C) Duplicate answer; (D) Relationship problem; and (E) Schema dependence*. Fig. 12 shows the percentage of answers containing a certain problem over total answers. Note that an answer may contain more than one problem.



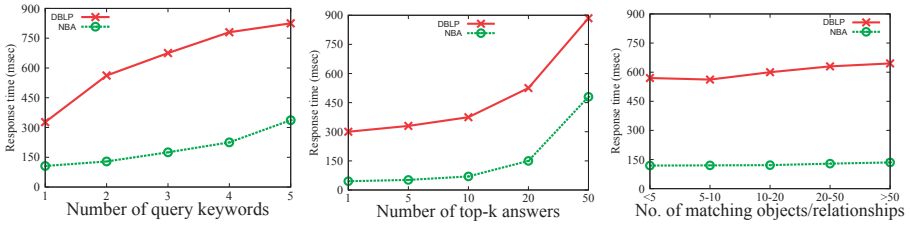
(a) DBLP (b) NBA (c) eBay

Fig. 12: Statistics on problems of answers: *(A) Meaningless answer; (B) Missing answer; (C) Duplicate answer; (D) Relationship problem; (E) Schema dependence*

Discussion. Generally, answers of our approach have less problems than those of the structure-based search. The most frequent problem is the meaningless answer. It usually occurs when a query contains only one keyword and the structure-based search returns only one non-object node matching that keyword. Sometimes our approach returns a meaningless answer when it discovers a composite attribute as an object. Missing and duplicated answers are also frequent problems because the structure-based search cannot discover the same object appearing in multiple nodes.

8.3 Response time of our approach

We evaluate the efficiency of our approach by varying the number of query keywords N, the number of top-k answers K and the number of matching objects and relationships M. The query processing time w.r.t. these variables is shown in Fig. 13(a), 13(b) and 13(c) respectively. We ran data sets NBA, and DBLP and used the average time of 10 runs for each query. We did not run eBay because its size is too small to evaluate efficiency.



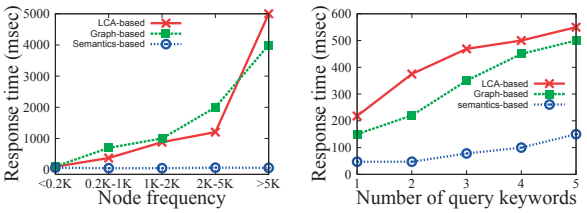
(a) $K = 10$ and $5 \leq M \leq 10$ (b) $N = 2$ and $5 \leq M \leq 10$ (c) $N = 2$ and $K = 10$

Fig. 13: Query processing time of our approach

Discussion. By exploiting efficient indexes, namely the term-node lists and the node-node lists, and by working at object and relationship level, our approach can process queries efficiently. Moreover, our approach's processing time increases with the number of query keywords and the number of top-k answers. The reason is that for each query processing, the number of index lists to be accessed increases with the number of keywords, and the more top-k answers there are, the more processing time is. In contrast, the running time is almost unchanged with the number of matching objects and relationships since this number does not increase the number of index lists needed to process a query.

8.4 Comparison on efficiency

We evaluate the efficiency of our approach, the LCA-based and the graph-based approach by varying the number of query keywords and node frequency of keyword, i.e., the frequency that a query keyword appears in the XML data. We ran data set NBA and reported the average time of 10 runs for each query. The result is shown in Fig. 14.
Discussion. The response time of varying node frequencies of keywords is plotted in Fig. 14(a). A set of queries with two keywords are randomly chosen. The response time of our approach depends on the frequency of matching objects and relationships rather than the node frequency since it works at object and relationship level instead of node level as SLCA and the graph-based search. Thus, our approach runs stably while the others' response time increases very fast with the node frequency. Moreover, working at object and relationship level enables our approach to run faster than SLCA and the graph-based approach since the number of matching objects and relationships needed for processing is less than the number of nodes required by SLCA and the graph-based search.



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

Fig. 14: Efficiency comparison

The processing time of varying the number of query keywords from 1 to 5 is given in Fig. 14(b). A set of queries whose keywords with medium node frequency, i.e., from 1000 to 2500, are randomly chosen. The response time of all systems increases almost linearly with the number of keywords. However, our approach runs faster than the others since it works at object and relationship level.

9 RELATED WORK

Tree-based XML keyword search. Most existing tree-based XML keyword search methods are LCA-based and depend on hierachical structure of the data. XRANK [8] proposes a stack based algorithm to efficiently compute LCAs.XKSearch [21] defines Smallest LCAs (SLCAs) to be the LCAs that do not contain other LCAs.Meaningful LCA (MLCA) [15] incorporates SLCA into XQuery.VLCA and ELCA [12, 22] introduce the concept of valuable/ exclusive LCA to improve the effectiveness of SLCA. Although researchers have put efforts on improving LCA-based effectiveness, their works are still based on the hierarchical structure without ORA-semantics. Thus, they face problems as studied in Sec. 3.

Graph-based XML keyword search. BANKS [3] uses backward search to find Steiner tree in labeled, directed graph. Later, Bidirectional [11] improves BANKS by using bidirectional search. BLINKS [9] proposes a bi-level index to prune and increase speed of bidirectional search for top-k answers. EASE [13] introduces a unified graph index to handle keyword search on heterogeneous data. Some of these work consider the schematic information to reduce search space in graph, but they do not consider ORA-semantics in query processing. Thus, they face problems as discussed in Sec. 4.

Semantics-based XML keyword search. XSEarch [5] focuses on adding semantics into query but the added semantics is for distinguishing a tag name and a value keyword only. XSeek [16] and MaxMatch [17] 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 [10] exploits semantics from the XML schema. XReal [1], Bao et. al. [2] and Wu et. al. [20] 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 answer but still suffer from all other problems discussed in Sec. 3 and 4.

10 Conclusion and future work

We have systematically illustrated limitations of the existing LCA-based search, including the problems of *meaningless answer, missing answer, duplicate answer,*

problems related to relationship, and *schema dependence* which are caused by unawareness of semantics of object, relationship and attribute. We have also demonstrated that even with IDREFs, the graph-based search can avoid at most the problems of *missing answer*, *duplicate answer* and *schema dependence* because IDREF mechanism is aware of only semantics of object and object ID but not semantics of relationship and attribute. Thus, the graph-based search still suffers from the problems of *meaningless answer* and *problems related to relationship*. We introduced *ORA-semantics* which is semantics of object, relationship and attribute and showed its importance in XML keyword search. To take ORA-semantics into account for XML keyword search, we propose Object Relationship (OR) graph to represent XML data, in which objects and relationships correspond to nodes, while attributes and values are associated with the corresponding object/relationship nodes. As such, OR graph can capture all ORA-semantics of an XML data. To process an XML keyword query, we search over the OR graph. Our index and optimization provide an efficient search algorithm. Experimental results showed that our OR graph based approach outperforms the structure-based search in term of both effectiveness and efficiency.

Therefore, semantics-based approach could be a promising direction for XML keyword search in solving problems of the current structure-based search. More broadly, this paper demonstrates the benefit of ORA-semantics in XML. By recognizing the concept of objects, relationship and attribute, we are able to add substantial semantics to XML represented data. We showed how this can greatly benefit XML keyword search. As part of our future work, we will explore how other XML processing can similarly benefit the ORA-semantics.

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. G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.
4. S. Cohen, Y. Kanza, B. Kimelfeld, and Y. Sagiv. Interconnection semantics for keyword search in XML. CIKM, 2005.
5. S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A semantic search engine for XML. In *VLDB*, 2003.
6. 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.
7. K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *SIGMOD*, 2008.
8. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD*, 2003.
9. H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD*, 2007.
10. V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *ICDE*, 2003.
11. V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, and R. D. Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.

12. G. Li, J. Feng, J. Wang, and L. Zhou. Effective keyword search for valuable LCAs over XML documents. In *CIKM*, 2007.
13. 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.
14. L. Li, T. N. Le, T. W. Ling, H. Wu, and S. Bressan. Discovering semantics from XML. *TRA3/12, 2012, School of Computing, NUS*.
15. Y. Li, C. Yu, and H. V. Jagadish. Schema-free XQuery. In *VLDB*, 2004.
16. Z. Liu and Y. Chen. Identifying meaningful return information for XML keyword search. In *SIGMOD*, 2007.
17. Z. Liu and Y. Chen. Reasoning and identifying relevant matches for XML keyword search. In *PVLDB*, 2008.
18. L. Ribeiro and T. Härder. Entity identification in XML documents. In *Grundlagen von Datenbanken*, 2006.
19. M. Weis and F. Naumann. Detecting duplicate objects in XML documents. In *IQIS*, 2004.
20. H. Wu and Z. Bao. Object-oriented XML keyword search. In *ER*, 2011.
21. Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD*, 2005.
22. R. Zhou, C. Liu, and J. Li. Fast ELCA computation for keyword queries on XML data. In *EDBT*, 2010.