THE NATIONAL UNIVERSITY
of SINGAPORE

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

TRA5/13

Object Semantics for XML Keyword Search

Thuy Ngoc Le, Tok Wang Ling, H. V. Jagadish, Chunbin Lin, and Jiaheng Lu

May 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

Object Semantics for XML Keyword Search

Thuy Ngoc Le, Tok Wang Ling, H. V. Jagadish, Chunbin Lin and Jiaheng Lu

Abstract—We know that some XML elements correspond to objects (in the sense of object-orientation) and others do not. The question we consider in this paper is what benefits we can derive from paying attention to such object semantics, particularly for the problem of keyword queries. Keyword queries against XML data have been studied extensively in recent years, with several lowest-common-ancestor based schemes proposed for this purpose, including SLCA, MLCA, VLCA, and ELCA. It is easy to see that identifying objects can help each of these techniques return more meaningful answers than just the LCA node (or subtree). It is more interesting to see that object semantics can also be used to benefit the search itself. For this purpose, we introduce a novel nearest common object node semantics (NCON), which includes not just common ancestors but also common descendants and referenced objects in evaluating a query. We have developed `XComplete`, a system for our NCON-based approach, and used it in our extensive experimental evaluation. The experimental results show that our proposed approach outperforms the existing LCA-based approaches in terms of both effectiveness and efficiency.

1 Introduction

Since XML has become a standard for information exchange over the Internet, research on XML keyword search has gained substantial interest. Inspired by hierarchical structure of XML data-centric documents, Guo et al. [4] and Schmidt et al. [15] proposed *LCA semantics* (Lowest Common Ancestor) for handling XML keyword queries. Many extensions of the LCA semantics such as SLCA [17], MLCA [9], ELCA [18] and VLCA [7] have been proposed to improve effectiveness of XML keyword search. The LCA-based approaches are widely accepted and work well for several types of XML keyword queries. However, they commonly suffer from two major drawbacks including *meaninglessness* of the answers and *incompleteness* of the answer set as discussed in the following examples.

Example 1 (Meaningless answer) *Consider an XML data tree as illustrated in Fig. 1 and its corresponding XML schema in Fig. 2. For query* {`Clinton`, `Kennedy`}*, an answer such as value node* `Clinton & Kennedy` *is meaningless since it does not provide any additional information about* `Clinton` *and* `Kennedy`*. A meaningful answer should contain supplementary information about* `Clinton` *and* `Kennedy` *such as the subtree rooted at node* `Paper(1.1.1.1)`*.*

Example 2 (The root as an LCA) *In the same XML data, suppose that a user wants to know the relationship between professors* `Stanley` *and* `Tony`*, he/she can issue query* {`Stanley`, `Tony`} *where* `Stanley` *and* `Tony` *are first names of professors. LCA-based approaches return only the root as an LCA node for this query. However, the whole document, the subtree rooted at the root, is surely not a meaningful answer.*

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 to use object semantics in XML keyword search, without making any major modifications to XML itself.

An *object* is a real world entity and is reflected by *object nodes* in XML data. An object is identified by its *object class* and *object ID (identifier)*. Thus, two objects are the same if they belong to the same object class and have the same object ID value. This is all the object orientation we rely upon. Using just this much, we show in this paper the benefits that accrue to keyword search.

Based on object identification, we introduce a novel semantics, called *Nearest Common Object Node* (NCON) for XML keyword search. The NCON semantics has two major features. First, an NCON must be an *object node*. Second, an NCON can be either an LCOA *(lowest common object ancestor)* or an HCOD (*highest common object descendant*). Intuitively, an LCOA is similar to an LCA [4], [7], [17]. However, an LCOA must be an *object node* while an LCA can be an arbitrary node. An HCOD is a common object descendant of a set of keywords and it has no ancestor which is also a common object descendant of that set of keywords. The first feature of NCON ensures that an answer of a keyword query is semantically *meaningful* for users. The second feature integrates *common descendants* into the answer set to return a more *complete* set of answers

- *Thuy Ngoc Le and Tok Wang Ling are with CS Department, School of Computing, National University of Singapore. E-mail: {ltngoc,lingtw}@comp.nus.edu.sg*
- *H. V. Jagadish is with Department of EECS, University of Michigan. E-mail: jag@eecs.umich.edu*
- *Chunbin Lin and Jiaheng Lu are with Renmin University of China. E-mail: {chunbinlin,jiahenglu}@ruc.edu.cn*

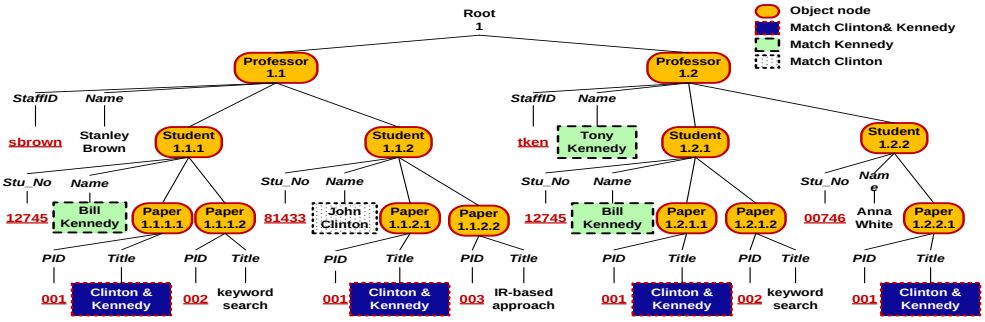


Fig. 1. The original XML data

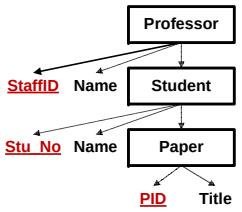


Fig. 2. Original schema

to users. To the best of our knowledge, this is the first attempt taking common descendants into account for XML keyword search.

Let us revisit the motivating examples introduced above and see how object-orientation may help.

EXAMPLE 3 (Example 1 Reprise) *`Paper(1.1.1.1)` is an object node that is the parent of the (non-object) `title` node with value `Clinton & Kennedy`. Returning the former is meaningful, whereas returning the latter is meaningless. LCOA semantics will force us up to the former, rather than stopping at the latter.*

EXAMPLE 4 (Example 2 Reprise) *We recognize that nodes `Student(1.1.1)` and `Student(1.2.1)`, actually refer to the same student since they belong to the same object class `Student` and have the same object ID value `12745`. This student is the common information related to both Professor `Stanley` and Professor `Tony`, i.e., the student co-supervised by both professors. HCOD will find this student, and this will be the returned answer. In contrast, LCA based approaches cannot detect this common student because it appears as a descendant, not as an ancestor. This is because LCA based approaches do not consider the semantics of object and their query results depend on the hierarchical structure of the XML data while NCON based approach makes use semantics of object and does not depend on the hierarchical structure of objects in the XML data.*

Incompleteness can be a problem for traditional LCA techniques even when there is a meaningful common ancestor as the following example shows:

EXAMPLE 5 (Incompleteness) *Suppose a user issues query {`Bill`, `John`} where keywords match two students `Student(1.1.1)` and `Student(1.1.2)` respectively. Is the answer `Professor(1.1)` (common ancestor) returned by LCA-based approaches complete? On careful examination, it is not. `Paper(1.1.1.1)` and `Paper(1.1.1.2)` refer to the same object because they belong to the same object class `Paper` and have the same object ID value `001`. Thus, the paper with `PID 001`, which is the common descendant of these student nodes, should also be included in the answer set. Intuitively, these students are not only supervised by the same professor `Stanley Brown`, but also co-authors of the same paper `Paper 001`. Therefore, not only common ancestors (LCA nodes), but also common descendants should be answer candidates for an XML keyword query.*

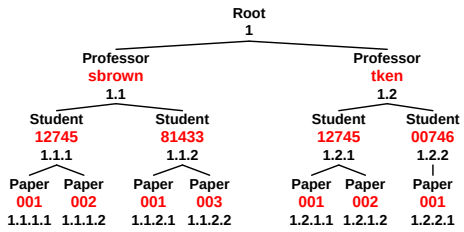
A final answer obtained by LCA-based approaches includes two parts: a returned node (LCA node) and presentation of the answer, e.g., a subtree or a path. Arguably, the presentation of an answer as a subtree rooted at a returned node may contain common descendants, but this presentation is too cumbersome and contains a maze of irrelevant information. This clouds the common descendants and makes it hard to determine them.

In contrast, NCON semantics clearly identify both common ancestors and common descendants, e.g., `Professor(1.1)`, and `Paper` with object ID `001` in Example 5.

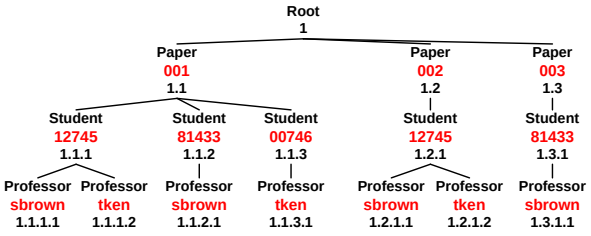
EXAMPLE 6 (Duplicated answer) *Returning to the query of Example 1, we see that there are several (four, to be precise, in Fig. 1) occurrences of the searched value {`Clinton`, `Kennedy`}. Even with NCON semantics, each of these `value` nodes has a distinct parent object node. However, we can see that all of these object nodes refer to the same object because they belong to the same object class `Paper` and have the same object ID value `001`. As such, it is enough to return this object just once.*

A common problem with XML is that objects are duplicated in multiple places to achieve a tree structure. Queries which match each of these duplicate instances, as we saw in the example above, often return overwhelmingly large result sets. Exploiting object semantics, we are able to filter out all such duplicated answers. Furthermore, to improve user comprehension, we merge answers for the same context to provide full comprehension about the context that a query is translated.

To facilitate finding NCONs, we define an XML object tree *(O-tree)* extracted from an XML data tree by keeping only object nodes. Non-object nodes are associated with their corresponding object nodes. Fig. 3(a) shows the O-tree extracted from the XML document in Fig. 1. To find HCODs, we also define a *reversed O-tree*, whose paths from the root to leafs



(a) The O-tree extracted from the data in Fig.1



(b) The reversed O-tree of the O-tree in Fig. 3(a)

Fig. 3. The XML object trees (O-trees)

are reversed from those of the given O-tree. Fig. 3(b) shows the reversed O-tree w.r.t. the O-tree in Fig. 3(a). If this reversal is done correctly, HCODs of the original O-tree can be found as LCOAs of the *reversed O-tree*.

In addition, to accelerate query processing, we develop *object index* and *keyword index* for both the reversed and the original O-trees. Moreover, we derive several *properties* related to Dewey labels and *lemmas* about the use of the reversed O-tree to facilitate the NCON computation. These index and optimization techniques provide an efficient algorithm for retrieving NCONs.

We have implemented all of these ideas in `XComplete`, our system for XML keyword search with more `complete` set of answers. We then use this system in our extensive experimental evaluation.

Our contributions can be summarized as follows.

- We present the impact of *object identification* in XML keyword search. It helps return more meaningful answer and improve the search quality.
- We introduce a new semantics, called NCON, for XML keyword search. Under this semantics, an answer node is an *object* node, and not only the LCOAs but also the HCODs are considered as answers for a query. Therefore, the answer set is more *complete* and each answer is more *meaningful* compared to answers returned by the LCA semantics and its extensions.
- We propose an NCON-based approach, which uses both *original* and *reversed* O-trees to find NCONs and return a more complete set of meaningful answers for an XML keyword query. Our post-processing techniques improve user comprehension by filtering out duplicated answers and merging answers from the same context. Our indexes and optimization techniques enable our approach to return NCONs efficiently.
- We have developed `XComplete` system to evaluate our approach. Experimental results on real datasets show that `XComplete` outperforms existing LCA-based approaches in terms of both *effectiveness* and *efficiency*.

Roadmap. We introduce the concept of NCON semantics in Section 2 and propose an NCON-based approach in Section 3. Section 4 describes index and optimization techniques. Experimental results are shown in Section 5. We briefly review related works in Section 6. We discuss benefits of object identification in XML keyword search in Section 7. Finally, we conclude the paper in Section 8.

2 Object Semantics

Not all nodes in XML documents correspond to objects. In theory, XML elements are objects and attributes. In practice, many XML elements are not objects. It is straightforward, with only a bare minimum of domain knowledge, to take an XML document and mark up the nodes that are objects. We assume that this has been done, in this paper. The corresponding terminology is introduced in Section 2.1.

The value of recognizing objects in XML documents is recognized both in terms of managing the returned results, and in terms of obtaining the match, for which a novel definition, called NCON, is introduced in Section 2.3.

XML allows the notion of reference links from a node to other nodes not its children, using an IDREF (ID Reference) mechanism. Such references are frequently important in an object-oriented system. However, they are typically not considered by XML keyword search systems. In Section 2.4, we describe our technique for managing IDREF.

2.1 Terminology

XML elements can be classified into object and non-object nodes. Among non-object nodes, a special attribute (or a set of attributes) that can uniquely define an object is called object identifier (object ID). For example, in XML data in Fig. 1, `Professor(1.1)` is an object node while `staffID`, `Name` are its attributes (non-object nodes), and `staffID` is its object ID. This section introduce concepts related to object used in this paper.

Concept 2.1 (Object class vs. object ID) *Similar objects, i.e., objects with the same (or similar) set of*

attributes, are grouped into an object class. An object class has an object ID to uniquely identify objects.

Object ID is usually single with only one attribute. However, there are still cases where object ID is a composite of several attributes. We have algorithms to discover such object ID in [8].

CONCEPT 2.2 (Object) *An object represents a real-world entity or concept. In an XML data tree, an object is represented by a tag node, followed by a set of attributes and their associated values to describe its properties. Each object belongs to an object class and has a unique object ID value.*

Since an object is identified by object class and object ID value, the same object is determined as follows.

CONCEPT 2.3 (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, `Student(1.1.1)` and `Student(1.2.1)` refer to the same object because they belong to the same object class `Student` and they have the same object ID value `12745`.

CONCEPT 2.4 (Object node) *An object can occur at many places in an XML data. Each occurrence is represented by an object node. An object is identified by object class and object ID value while its object nodes are identified by its node identifier, i.e., label.*

Multiple object nodes that refer to the same object are usually identical. However, XML does not enforce this. If we find two nodes that are not identical, they are still considered as referring to the same object as long as they have the same object class and object ID value. For example, hobbies and degrees of a staff can be represented at different object nodes corresponding to the same staff object. If such object is queried, the returned object should include the union of attributes together with their values of the matching object nodes. We do not include attributes and values of the unmatching object nodes because they are irrelevant. We assume that the data is consistent because data integration and data uncertainty are out of the scope of this work.

CONCEPT 2.5 (Non-object node) *Apart from object nodes, all other nodes, e.g., attribute, value and connection node, in an XML document are non-object nodes. Each non-object node is associated with a corresponding object node.*

Since we differentiate object and non-object node in XML data, a matching node is re-defined as follows.

CONCEPT 2.6 (Matching object node) *A keyword k matches an object node u if k matches u or k matches non-object nodes associated with u*[1].

2.2 Object identification

An object is identified by its *object class* and *object ID value*. In many cases, these are directly available, because the XML schema was initially designed based on those semantics. For example, in the XML schema in Fig. 2, `Professor`, `Student` and `Paper` are object classes, and `StaffID`, `Stu_No` and `PID` are their object IDs, respectively. When this is not the case, these values can be discovered from XML schema and data by third-party algorithms such as [14], [6], [8]. We apply algorithm [8] to automatically discover such semantics and achieve high accuracy (greater than 98% for object classes and greater than 93% for object IDs). More details can be found in [8].

2.3 The NCON semantics

This section introduces our proposed NCON semantics, which includes both common ancestors and common descendants to provide a more complete answer set for XML keyword search. Our proposed NCON semantics can be built on the LCA semantics [4] or any of its variants such as SLCA [17], VLCA [7] and ELCA [18]. For simplicity of presentation, we work with LCA semantics in the rest of this paper. It is straightforward to make the necessary minor modifications required if any of the variant semantics are preferred. Let $u \prec_a (\succ_a) v$ denote that node u is an *ancestor* (a *descendant*) of node v. The NCON semantics and its two components, which are LCOA (Lowest Common Object Ancestor) and HCOD (Highest Common Object Descendant), are defined as follows.

DEFINITION 1 (The LCOA of a set of nodes) *Object node u is the LCOA of a set of nodes $\{u_1, \ldots, u_n\}$ if (1) $u \preceq_a u_i \ \forall i = 1..n$ and (2) there exists no object node $v \succ_a u$ s.t. $v \preceq_a u_i \ \forall i = 1..n$.*

The LCOA semantics is similar to the LCA semantics. However, an LCOA must be an *object node* while an LCA can be either an object node or a non-object node. It is this difference which provides the advantage of the NCON semantics, i.e., not returning *meaningless* answers.

DEFINITION 2 (The HCOD of a set of object nodes) *Given a set of object nodes $\mathbb{I} = \{u_1, \ldots, u_n\}$, the set of object nodes $\mathbb{H} = \{h_1, \ldots, h_n\}$ is the HCOD of $\mathbb{I}$ if*

- *(1) all h_i's refer to the same object and*
- *(2) $u_i \preceq_a h_i \quad \forall i = 1..n$ and*

1. Specifically, k is contained in (1) object class of u, (2) object attributes and their values of u, (3) composite attributes and aggregational nodes associated to u, (4) relationship attributes and their values of the relationship in which u is the lowest participating objects.

- *(3) there exists no set of object node* $\mathbb{H}' = \{h'_1, \ldots, h'_n\}$ *where* $h'_i \prec_a h_i \ \forall i = 1..n$ *which satisfies the above two conditions.*

HCOD is the distinguishing feature of the NCON semantics. An HCOD contains a set of object nodes which refer to the same object because common descendant can not be represented as one node.

DEFINITION 3 (An NCON of a set of nodes) *An NCON of a set of nodes is either an LCOA or an HCOD of that set of nodes.*

2.4 Object reference

An XML data with ID References (IDREF) is not represented as a tree any more, but as a graph instead. However, such graph is not an arbitrary graph. It still has hierarchical structure with parent-child and ancestor-descendant relationships between elements and sub-elements. An IDREF is the link from a node to the node that it refers to. We can consider these nodes have ancestor-descendant relationship where the ancestor is the referee node and the descendant is the referred node. As such, ancestor-descendant relationship in an XML data can be re-defined as follows.

$ancestor(x, y) : -ancestor(x, z), ancestor(z, y).$
$ancestor(x, y) : -parent(x, y).$
$ancestor(x, y) : -IDREF(x, y).$
$ancestor(x, x).$

As a result, object identification discussed in Section 2.2 and the NCON semantics introduced in Section 2.3 are still valid to an XML data with IDREFs.

3 OUR NCON-BASED APPROACH

This section presents our proposed NCON-based approach for XML keyword search, which applies our proposed NCON semantics.

3.1 The mechanism of our approach

By following the NCON semantics, our approach has two main goals which are improving *precision* by ensuring that each answer is *meaningful* to users, and improving *recall* by returning a more *complete* set of answers. To achieve the first goal, our approach works at object level, i.e., it searches for returned nodes in an *XML object tree* (O-tree) instead of in an XML data tree. An O-tree is extracted from the corresponding XML data tree and is defined as follows.

CONCEPT 3.1 (O-tree) *An O-tree* OT_O *is an XML hierarchical structure which is extracted from an XML data tree* DT_O *by keeping all object nodes, but removing all non-object nodes. For any two adjacent object nodes* u *and* v *in* DT_O*, there is a parent-child edge between* u *and* v *in* OT_O*.*

For example, the O-tree extracted from the XML data in Fig. 1 is shown in Fig. 3(a), in which Dewey label is used to identify object node while object class and object ID value are used to identify object. Due to not counting non-object nodes, e.g., attributes and values, the extracted O-tree is much smaller than the corresponding XML data tree.

To achieve the second goal, the essence of the idea is a *reverse mechanism* by which HCODs of the given O-tree are turned into LCOAs in its reversed O-tree which is defined as follows.

CONCEPT 3.2 (Reversed O-tree) *Given an O-tree* OT_O*, the reversed O-tree w.r.t.* OT_O *is the O-tree* OT_R *such that (1) for each path of object nodes* $/u_1/u_2/\ldots/u_{n-1}/u_n$ *from the root to a leaf in* OT_O*, there is a corresponding reversed path* $/u'_n/u'_{n-1}/\ldots/u'_2/u'_1$ *in* OT_R *where each pair of object nodes* u_i *and* u'_i *refer to the same object, and (2) there is no other object node in* OT_R*.*

For example, Fig. 3(b) shows the reversed O-tree extracted from the O-tree in Fig. 3(a). The reversed O-tree is materialized in external memory and is generated offline. The reversed O-tree serves for searching HCODs only. Although there is duplication in O-trees, such duplication does not affect the efficiency of processing thank to our indexes and optimization techniques, which will be discussed in Section 4.

The completeness of using the reversed O-tree. The set of answers for an XML keyword query includes both common ancestors and common descendants of nodes matching the query keywords. Common ancestors can be found as in LCA-based approaches while common descendants are found by exploiting the reversed O-tree. Thus, although many other O-trees, apart from the reversed O-tree, capture the same information with the original O-tree, only the duo of the original and reversed O-tree is *self-sufficient* to return the more complete set of answers.

3.2 Overview of the process

The process of our approach, as shown in Fig. 4, comprises two components for *offline* and *online* computations. For offline computations, the three main tasks are extracting the O-tree, generating the reversed O-tree and indexing. For online computations, there are three phases, namely finding NCONs, generating answers and post-processing answers. For comprehensive understanding, we briefly describe the main processes as follows. Details will be discussed in the remaining of this section and Section 4.

3.2.1 Offline computation

Extracting the O-tree. We extract the O-tree from a given XML document by keeping only object nodes.

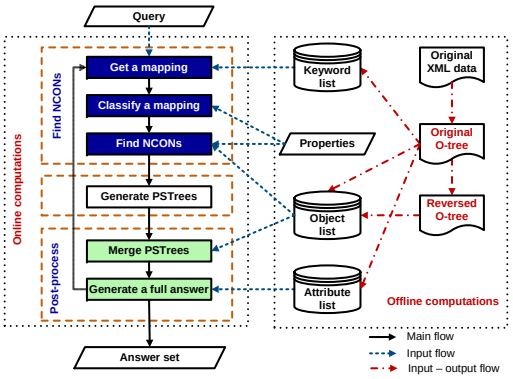


Fig. 4. The process of our approach

In an XML document, an object node corresponds to an object class in the corresponding XML schema.

Generating the reversed O-tree. The generation process can be done in two main steps. First, we traverse the original O-tree backward from each leaf node to the root to form reversed paths and then connects them to form an intermediate O-tree. After that, it then merges the nodes of the the intermediate O-tree which have the same set of object ancestors. Details are given in Section 3.3.

3.2.2 Online computation

When a query has some keyword(s) which match multiple objects, there are different *mappings* for that query. Each mapping shows a map from the set of query keywords to a set of matching objects. Let $Obj(k)$ denote the set of objects matching keyword k[2], a query mapping is defined as follows.

CONCEPT 3.3 (A query mapping) *Given a keyword query $Q = \{k_1, \ldots, k_n\}$, a mapping of query Q is $\mathcal{M}_Q = \cup_{i=1}^{n}\{o_i\}$, $o_i \in Obj(k_i)$.*

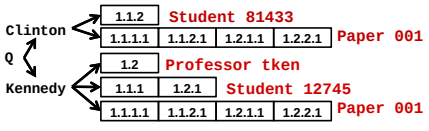


Fig. 5. Matching objects of `Clinton` and `Kennedy`

TABLE 1
Query mappings and their corresponding case

Mapping	Objects (sorted)	Case
$\mathcal{M}_Q{}^1$	`Professor:tken, Student:81433`	Case 3B
$\mathcal{M}_Q{}^2$	`Student:81433, Student:12745`	Case 3B
$\mathcal{M}_Q{}^3$	`Student:81433, Paper: 001`	Case 2
$\mathcal{M}_Q{}^4$	`Professor:tken, Paper: 001`	Case 2
$\mathcal{M}_Q{}^5$	`Student:12745, Paper: 001`	Case 2
$\mathcal{M}_Q{}^6$	`Paper: 001`	Case 1

2. An object matches keyword k when any of its object node matches k

For example, consider query `Q = {Clinton, Kennedy}` issued to the data in Fig. 1. Objects and Dewey labels of matching object nodes are shown in Fig. 5. Query mappings of Q are given in Table 1, in which cases for optimization purposes will be discussed in Section 4.

We handle query mappings on finding NCONs because of the following advantages: (1) *merging answers* of the same query mapping to avoid overwhelming users by a plethora of returned answers and to make the merged answer more comprehensive to users, and (2) processing a query more *efficiently* and *effectively* because many *duplicated* and *irrelevant* answers are filtered out *on the fly*. Therefore, we process a query based on its mappings. The number of mappings is not much because of two reasons. First, the number of object nodes in an O-tree is much less than the total number of arbitrary nodes in the corresponding XML data tree. Second, duplicated objects are excluded based on object ID of object class. We provide a basic algorithm for query processing in Section 3.4 and an optimized algorithm in Section 4.4.

3.2.3 Improving efficiency

The challenging issue is how to exploit the reversed O-tree for a more complete answer set while the cost is not double. We observe that in many cases, the reversed O-tree is not necessary because it provides no new answer. Thus, we propose a lemma for the necessity of the reversed O-tree as follows.

LEMMA 1 (The necessity of the reversed O-tree) *Given a set of matching nodes $S = \{u_1, u_2, \ldots, u_n\}$ where u_i matches keyword k_i, the reversed O-tree does not provide any new answer for S if there exist two nodes $u_i, u_j \in S$ such that they are the leaf nodes in the original O-tree.*

The rationale behind is that when u_i and u_j are the leaf nodes in the original O-tree, they become the highest nodes in the reversed O-tree with no ancestor beside the root. They do not have ancestor-descendant relationship for one of them to become a common ancestor either. As such, there is no common ancestor of these two nodes. Thus, there is no common ancestor of S.

Another technique to improve efficiency is filtering out duplicated answers efficiently. Duplicated answers provide nothing new or even annoy users. Removing them at post-processing step when we already need cost for retrieving them is very inefficient. Thus, we propose some properties to filter out duplicated answers on the fly and get the optimal cost $O(1)$ for some cases. These optimization techniques are described in Section 4

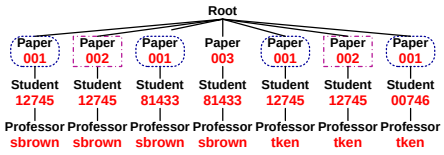


Fig. 6. Intermediate O-tree

3.3 Generating the reversed O-tree

Before generating the reversed O-tree, there is an intermediate step of generating the intermediate O-tree. We also consider XML documents with complicated schema.

3.3.1 Generating the intermediate O-tree

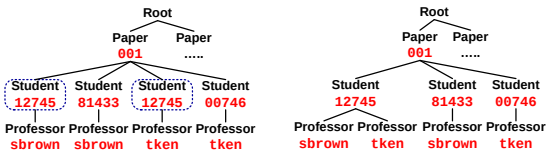
Algorithm 1 presents the process of generating the intermediate O-tree from the original O-tree OT_O. We traverse OT_O backward from each leaf node to the root to form a reversed path. Then, all reversed paths are connected to form the intermediate O-tree. We use an *array-like-stack* S to store all object nodes in OT_O. An array-like-stack is an array in which *push* and *pop* operators are used in similar way to a stack while we still can access any element in S like an array. We traverse OT_O by depth first order and push visited object nodes into S. To handle the branches in the tree, we maintain the parent of each object node. Thus, we use the triple $\langle i, (objCls(i) - OID(i)), pre(i)\rangle$ to represent each object node i, where i is the *index* by depth first order (i starts from 1), $objCls(i)$ and $OID(i)$ are the *object class* and *object ID* of i and $pre(i)$ is the *index* of the *parent* of i. Fig. 6 shows the intermediate O-tree w.r.t. the original O-tree in Fig. 3(a).

Algorithm 1: Generating the intermediate O-tree

Input: The original O-tree OT_O
Output: Intermediate O-tree OT_I
Variables: Array-like-Stack S to store object nodes in OT_O
Traverse OT_O by depth first order
for *visited object node* $i \in OT_O$ **do**
  S.Push $(\langle i, (objCls(i) - OID(i), pre(i)\rangle)$
OT_I. Add (Root)
OT_I. NewBranch
while $S \neq \emptyset$ **do**
  $\langle i, (objCls(i) - OID(i), pre(i)\rangle \leftarrow S$.Pop
  OT_I. Add $(objCls(i) - OID(i))$
  if $pre(i) = 0$ **then**
    OT_I. NewBranch
  if $pre(i) \neq i - 1$ *and* $pre(i) \neq 0$ **then**
    $k \leftarrow pre(i)$
    while $k \neq 0$ **do**
      Access element k $\langle k, (objCls(k) - OID(k), pre(k)\rangle$
      OT_I. Add $(objCls(k) - OID(k))$
      if $pre(k) = 0$ **then**
        OT_I. NewBranch
      if $pre(k) = k - 1$ **then**
        k. Next
      $k \leftarrow pre(k)$

3.3.2 Generating the reversed O-tree

To generate the reversed O-tree from the intermediate O-tree, we merge object nodes having the same set of ancestors. Particularly, at the first level of the intermediate O-tree, we merge branches in which the starting object nodes refer to the same object. Then we recursively merge the lower levels. Fig. 7 demonstrates merging processes w.r.t. the intermediate object node in Fig. 6.



(a) The first level (b) The second level

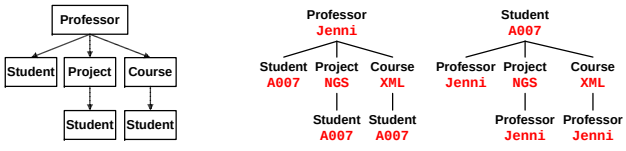
Fig. 7. Merging branches having the same set of ancestors

Space complexity. We store two O-trees in the memory. Thus, the space complexity is $O(2N)$ where N is the average number of object nodes in an O-tree. The number of object nodes is much smaller than the total number of nodes in an XML data tree, thus the space complexity is feasible even for a large XML document.

Time complexity. Each element in the array-like-stack is processed once so the cost is $O(N)$ where N is the number of elements in the array-like-stack. To generate the reversed O-tree, for each level of the tree, it costs $O(M)$ where M is the average number of nodes in each level. Hence, the total cost is $O(N + d \times M)$ where d is the depth of the schema tree.

3.3.3 XML document has complicated structure

If an object class has more than one occurrence in the XML schema, e.g., object class `Student` in Fig. 8(a), its corresponding objects may have more than one role in the XML data, e.g., object `Student A007` in Fig. 8(b). Our process of the reversed O-tree generation is still valid. Fig. 8(c) shows the reversed O-tree w.r.t. the original O-tree in Fig. 8(b).



(a) Multi class paths (b) Original data (c) Reversed data

Fig. 8. Object with multiple roles

When XML document has complicated structure related to relationship, the above process of generation is extended by adding some extra operators as follows.

- An explicit relationship is kept in the O-tree.
- An n-arry relationship ($n \geq 3$) is reversed as a set of binary relationships, because such reversion already enables us to find common descendants.
- An one-to-many relationship is not reversed.
- A relationship attribute is added to the lowest object node of the relationship it belongs to.

3.4 Query processing

When a query Q is issued on an XML document, we first translate it into all possible query mappings. We then process each query mapping separately because each query mapping can provide at least one meaningful answer that *distinct* with existing answers, and the number of query mappings is acceptable. For ease of processing, objects in a query mapping are sorted ascendingly by the number of object nodes. To process a query mapping $\mathcal{M}_Q$, we find $LCOA^O(\mathcal{M}_Q)$ and $LCOA^R(\mathcal{M}_Q)$, where $LCOA^O(\mathcal{M}_Q)$ and $LCOA^R(\mathcal{M}_Q)$ are the set of LCOAs for $\mathcal{M}_Q$ w.r.t. the original O-tree OT_O and the reversed O-tree OT_R, respectively. In implementation aspect, we adopt SLCA semantics and the Indexed Lookup Eager Algorithm in [17] to find LCOAs. By the *reverse mechanism*, we then translate the $LCOA^R(\mathcal{M}_Q)$ to $HCOD^O(\mathcal{M}_Q)$. By Lemma 1, the reversed O-tree is not necessary if more than one object in $\mathcal{M}_Q$ corresponds to leaf nodes. NCONs for $\mathcal{M}_Q$ is the union of $LCOA^O(\mathcal{M}_Q)$ and $HCOD^O(\mathcal{M}_Q)$. Let $lbl(o)$ be the list of Dewey labels corresponding to the set of object nodes referring to object o, the basic progress of finding NCONs for a query mapping is presented Algorithm 2 while the optimized algorithm is studied in Section 4.

Algorithm 2: Processing a query mapping

Input: A query mapping $\mathcal{M}_Q = \{o_1, \ldots, o_m\}$
Output: $NCON(\mathcal{M}_Q)$
1 $NCON(\mathcal{M}_Q) \leftarrow \emptyset$
2 $LCOA^O(\mathcal{M}_Q) \leftarrow$ find LCOAs ($\mathcal{M}_Q$) w.r.t. OT_O
3 $NCON(\mathcal{M}_Q) \leftarrow LCOA^O(\mathcal{M}_Q)$
4 **if** $\nexists ij \in [1..m]$ *such that* $Class(o_i)$ *and* $Class(o_j)$ *are leaf nodes* **then**
5 $\quad LCOA^R(\mathcal{M}_Q) \leftarrow$ find LCOAs ($\mathcal{M}_Q$) w.r.t. OT_R
6 $\quad HCOD(\mathcal{M}_Q) \leftarrow LCOA^R(\mathcal{M}_Q)$
7 $\quad NCON(\mathcal{M}_Q) \leftarrow LCOA^O(\mathcal{M}_Q) \bigcup HCOD(\mathcal{M}_Q)$

Complexity. Searching a match (left match, right match) in $lbl(o_i)$ costs $O(log(|lbl(o_i)|))$. The complexity of finding $LCOA^O(\mathcal{M}_Q)$ is $O(|lbl(o_1)| \times \sum_{i=2}^{m} log(|lbl(o_i)|))$ where m is the number of objects, $|lbl(o_1)|$ and $|lbl(o_m)|$ are the smallest and the biggest among all $|lbl(o_i)|$'s. Thus, the complexity of finding $LCOA^O(\mathcal{M}_Q)$ is $O(|lbl(o_1)| \times m \times log(|lbl(o_m)|))$. The complexity of finding $LCOA^R(\mathcal{M}_Q)$ is similar to that of finding $LCOA^O(\mathcal{M}_Q)$. Thus, the total cost of finding the set of NCON nodes w.r.t. a query mapping having m objects is $O(|lbl(o_1)| \times m \times log(|lbl(o_m)|))$ and $O(|lbl(o_1)| \times 2m \times log(|lbl(o_m)|))$ for the best and the worst case respectively, where the reversed O-tree is always unnecessary/necessary to process a query.

Algorithm 3: finding LCOAs($\mathcal{M}_Q$)

Input: A query mapping $\mathcal{M}_Q = \{o_1, \ldots, o_m\}$ `//sorted ascendingly by the number of object nodes)`
Output: $LCOAs$
1 **for** $i = 1 \rightarrow |lbl(o_1)|$ **do**
2 $\quad LCOA \leftarrow lbl(o_1)[i]$
3 $\quad$ **for** $j = 2 \rightarrow m$ **do**
4 $\quad\quad lm \leftarrow$ get left match ($LCOA, lbl(o_j)$)
5 $\quad\quad rm \leftarrow$ get right match ($LCOA, lbl(o_j)$)
6 $\quad\quad lLCOA \leftarrow$ lcoa($LCOA, lm$)
7 $\quad\quad rLCOA \leftarrow$ lcoa($LCOA, rm$)
8 $\quad\quad LCOA \leftarrow$ get smaller LCOA ($lLCOA, rLCOA$)
9 $\quad LCOAs$. Add ($LCOA$)
10 $LCOAs$. RemoveAncestorLCOAs

3.5 Output presentation and post-processing

3.5.1 Output presentation

Presentation of an answer. To avoid returning irrelevant information, we present an answer as a *partial subtree* $psT(R, N)$ where R and N are the root and the set of object nodes in psT. A PSTree is a subtree rooted at a returned NCON and contains only object nodes on the paths from the returned object node to the matching object nodes.

Converting answers from the reversed O-tree to the original O-tree. For easy comprehension, we only show outputs on the original XML document because the reversed document is used without users' awareness. Answers from the reversed document are converted into the original document.

3.5.2 Output post-processing

Removing duplicated answers. Duplicated answers can overwhelm users with extraneous information. If an object o appears 100 times in an XML document, when o is queried, there may be 100 duplicated answers which overwhelm and annoy users. Without object identification, these duplicates have to be determined through syntactic analysis, since they are actually distinct nodes in the XML tree. With object identification, they become trivial to detect. In a simple post-processing step, we filter out these duplicates. We also propose techniques to filer such duplicates on the fly.

Merging answers for the same query mapping. Intuitively, a query mapping provides the context of a query which describes the meaning of the set of query keywords. An answer for a query mapping shows a view of the context. To show the global view of the context, we merge answers for the same query mapping. For example, query {`XML`, `Database`} has a mapping containing two `papers`

about XML and Database. Suppose there are two answers for this mapping: the common conference and the common authors. If these two answers are merged, the merged answer means that both papers are written by the common authors, and are accepted by the common conference. As can be seen, the merged answer provides more complete comprehension of the context rather than the two separated ones. On the other hand, we do not merge answers from different mappings because they provide different aspects of a query. For example, if the above query has another mapping with the meaning of two courses about XML and Database, the answer about common students taking these courses should not be merged with the above two answers. The other benefit of merging answers is to deal with information overload when query keywords match a lot of objects.

For illustration of the output presentation and post-processing, the final answers for all mappings of query {Clinton, Kennedy} after post-processing are shown in Fig. 9.

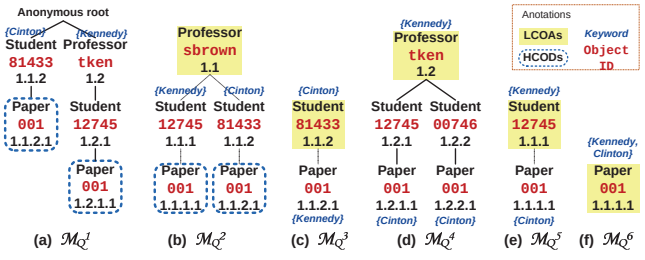


Fig. 9. Final answers of query {Clinton, Kennedy}

4 Index and Optimization

To enhance online performance, we maintain two main indexes, and propose some optimization techniques to filter out duplicated answers on the fly. We have two observations. First, each query has at least one distinct meaningful answer and thus we still need to process each query mapping. Second, in some cases, a query mapping provides only one answer. In such cases, after finding the first answer, we can stop processing because all new answers will be duplicated. Thereby, the efficiency is largely improved.

In this section, we classify a query mapping into different cases to remove duplicated answers efficiently. We also optimize the classification by leveraging some properties. Moreover, among classified cases, we carefully check the ones that do not need to use the reversed O-tree beside the case discussed in Lemma 1.

4.1 Indexes

4.1.1 Keyword list

Similar to traditional *inverted list*, our approach pre-computes the keyword list to efficiently retrieve the set of object nodes in the original document matching a keyword. However, the way our approach maintains the keyword list is different from that of the inverted list. In our approach, each keyword matches a *list of objects* ordered decreasingly by hierarchical level of objects. A matching object corresponds to a *list of Dewey labels* sorted by preorder numbering. Table. 2 shows part of the keyword list of the XML data in Fig. 1, in which keyword Kennedy matches three objects and keyword Clinton matches two objects.

TABLE 2
Keyword list of the original document

Keyword	Labels grouped by object
Kennedy	{1.2}, {1.1.1, 1.2.1}, {1.1.1.1, 1.1.2.1, 1.2.1.1, 1.2.2.1}
Clinton	{1.1.2}, {1.1.1.1, 1.1.2.1, 1.2.1.1, 1.2.2.1}
...	...

4.1.2 Object list

Object list is created for two purposes. It is used to determine whether two object nodes refer to the same object or not, and more importantly, to identify the set of object nodes in the reversed O-tree corresponding to a given object. This set of object nodes will be used to find answers from the reversed O-tree. Each object, represented as a pair of *object class* and *object ID value*, is indexed with a *list of Dewey labels* of its object nodes in both the original and the reversed O-trees. Part of the object list of the O-trees in Fig. 3 is given in Table 3.

TABLE 3
Object list of the original and reversed O-trees

Object class	Object ID	Labels in OT_O	Labels in OT_R
Professor	sbrown	1.1	1.1.1.1, 1.1.2.1, 1.2.1.1, 1.3.1.1
Student	12745	1.1.1, 1.2.1	1.1.1, 1.2.1
Paper	001	1.1.1.1, 1.1.2.1, 1.2.1.1, 1.2.2.1	1.1
...	...	...	...

Besides the two main lists, our approach maintains the *attribute list* in which each object node corresponds to an object class, an object ID, a set of object attributes and their associated values. This list is used to restore the output with full attributes and values.

4.2 Cases of query mappings

Before introducing cases of query mappings, we introduce some related concepts.

Object class path. In an XML schema, an object class may have multiple occurrences. An occurrence

is identified by the path from the root to the occurrence, referred to as *object class path* in this paper. For example, in the XML object class tree Fig. 8(a) (extracted from an XML schema by keeping only object classes), object class `Student` has three occurrences corresponding to three object class paths: `Root/Professor/Student`, `Root/Professor/Project/Student`, and `Root/Professor/Course/Student`. An object is an instance of an object class in XML document.

Let $ObjNode(o)$ be the set of object nodes referring to object o and $ObjNode(o, p)$ as the set of object nodes w.r.t. object class path p of object o.

Ancestor object w.r.t. object class path. Consider two objects o_1 and o_2. If each object node in $ObjNode(o_1, p1)$ has some node in $ObjNode(o_2, p2)$ as its descendant, then object o_1 is considered as an ancestor of object o_2 w.r.t. object class paths $p1$ of o_1 and $p2$ of o_2. For example, in Fig. 3(a), `Student:12745` is an *ancestor object* of `Paper:001`.

CONCEPT 4.1 (Ancestor-descendant (AD) chain w.r.t. a set of object nodes) *Object nodes $n_1, \ldots, n_m$ (ordered decreasingly by hierarchical level) have AD chain iff n_i is an ancestor object node of n_{i+1} $\forall i = 1..(m-1)$.*

CONCEPT 4.2 (AD chain w.r.t. a set of objects) *Objects $o_1, \ldots, o_m$ have AD chain iff there exists a set of object node $\mathbb{S} = \{n_1, \ldots, n_m\}$, $n_i \in ObjNode(o, p_i)$ such that object nodes in $\mathbb{S}$ have AD chain.*

For example, objects `Professor:sbrown`, `Student:12745` and `Paper:001` have AD chain because there exists an AD chain between their object nodes `Professor(1.1)`, `Student(1.1.1)` and `Paper(1.1.1.1)`.

Cases of query mapping. We classify a query mapping $\mathcal{M}_\mathcal{Q}$ into the following cases based on the relationships of objects in $\mathcal{M}_\mathcal{Q}$.

- (Case 1) $\mathcal{M}_\mathcal{Q}$ contains only *one object*.
- (Case 2) $\mathcal{M}_\mathcal{Q}$ has *multiple objects* and objects in $\mathcal{M}_\mathcal{Q}$ have AD chain.
- (Case 3) Objects in $\mathcal{M}_\mathcal{Q}$ have *no* AD chain. This case is further divided into two sub-cases.
  - (Case 3A) At least two objects correspond to leaf node.
  - (Case 3B) At most one object corresponds to leaf node.

The rationale of this classification is to determine the cases which provide only one answer and the cases which do not need the reversed O-tree to be processed. By Lemma 1, only Case 3A does not need to use the reversed O-tree. However, by carefully examination, we find that Case 1 and Case 2 do not need the reversed O-tree either. For Case 1 ($\mathcal{M}_\mathcal{Q} = \{o\}$), each object node in $ObjNode(o)$ is the NCON of itself and the reversed O-tree does not provide any additional answer from the original O-tree. For Case 2, the chain among objects in $\mathcal{M}_\mathcal{Q}$ is an answer. If the reversed O-tree is used, the answer is also this chain with the reversed order. Thus, the reversed O-tree does not return any new answer. Therefore, we extend Lemma 1 as follows.

LEMMA 2 (Lemma 1 - Extension) *Given a query mapping $\mathcal{M}_\mathcal{Q}$, the reversed O-tree is necessary to process $\mathcal{M}_\mathcal{Q}$ only when objects in $\mathcal{M}_\mathcal{Q}$ have no AD chain and at most one object in $\mathcal{M}_\mathcal{Q}$ corresponds to leaf node.*

4.3 Case classification

Recall that $ObjNode(o)$ denotes the list of *object nodes* referring to object o. This section provides identification of cases of a query mapping.

4.3.1 Identifying Case 1

Identifying Case 1 requires removing duplicated objects, which can be done by exploiting the following properties.

Property 1: $ObjNode(o_1) \cap ObjNode(o_2) = \emptyset$ for all objects $o_1 \neq o_2$.

Proof: One object node cannot refer to two objects. Thus, the list of Dewey labels of any two different objects cannot be overlapped. □

Two objects o_1 and o_2 are the same if their lists of object nodes are the same. $o_1 = o_2$ if $ObjNode(o_1) = ObjNode(o_2)$, where $ObjNode(o_1) = ObjNode(o_2)$ if $|ObjNode(o_1)| = |ObjNode(o_2)|$ and $ObjNode(o_1)[i] = ObjNode(o_2)[i]$ $\forall i = 1..|ObjNode(o_1)|$. With Property 1, we can figure out whether two objects are the same or not right after testing their *first object nodes*.

Property 2: Objects o_1 and o_2 are the same if $ObjNode(o_1)[1] = ObjNode(o_2)[1]$.

Proof: Suppose that objects $o_1 \neq o_2$, then $ObjNode(o_1) \cap ObjNode(o_2) = \emptyset$ (Property 1). However, if $ObjNode(o_1)[1] = ObjNode(o_2)[1]$, then $ObjNode(o_1) \cap ObjNode(o_2) \neq \emptyset$. Thus, if $ObjNode(o_1)[1] = ObjNode(o_2)[1]$, then $o_1 = o_2$. □

Complexity of checking Case 1. Property 2 enables our approach to remove duplicated objects in $\mathcal{M}_\mathcal{Q}$ efficiently. The complexity of checking whether two objects are duplicated is $O(1)$ because only their first Dewey labels are tested. After removing duplicated objects in $\mathcal{M}_\mathcal{Q}$, the complexity of checking *Case 1* is $O(1)$ since $\mathcal{M}_\mathcal{Q}$ falls into Case 1 if $|\mathcal{M}_\mathcal{Q}| = 1$.

4.3.2 Identifying Case 2

Without loss of generality, we present the case where an object class has only one object class path in an XML schema. The case where an object class has multiple class paths in an XML schema is presented in our Section 4.5. To speed up the checking of Case 2, we exploit some other properties as follows.

Property 3: For an object o, the subtrees rooted at all object nodes in $ObjNode(o)$ are the same if o and its descendant objects are not involved in n-arry relationships ($n \geq 3$).

For example, two object nodes `Student(1.1.1)` and `Student(1.2.1)` in Fig. 1 refer to the same object `Student 12745`. As can be seen, the subtrees rooted at these two object nodes are the same. Property 3 is useful for testing AD chain of objects since we only need to test the subtrees rooted at the *first node* of the highest object as stated in Property 4.

Proof: Property 3 is based on the fact that if two objects o_1 (the ancestor) and o_2 (the descendant) has a *binary* relationship, then for each node $u \in Node(o_1)$, the number of nodes in $Node(o_2)$ which are descendants of u are the same. Property 3 can be proved by using this fact *recursively*. Suppose there exists a set of objects $\{o_1, \ldots, o_n\}$ (sorted decreasingly by hierarchical level of objects), in which there is a binary relationship between any two adjacent objects o_i and o_{i+1}, $i = 1..(n-1)$. For all o_i, $i = 1..(n-1)$, for each node $u \in Node(o_i)$, the number of nodes in $Node(o_{i+1})$ which are descendants of u are the same. Thus, for all nodes $v_i \in Node(o_1)$, the set of all descendants of v_i's are the same. In other words, the subtrees rooted at v_i's are the same. □

Property 4: Objects $o_1, \ldots, o_m$ (ordered decreasingly by hierarchical level of objects) have AD chain if there exists an AD chain among object nodes $u_1, \ldots, u_m$ where $u_1 = ObjNode(o_1)[1]$ and $u_i \in ObjNode(o_i)$ $\forall\ i = 2..m$.

Proof:

Phase 1: If there exists a chain $(u_1, \ldots, u_m)$, $u_1 = Node(o_1)[1]$, $u_i \in Node(o_i)$, $i = 2..m$ s.t. $u_i \prec_n u_{i+1}$, $i = 1..(m-1)$, then there exists a chain $(u_1, \ldots, u_m)$, $u_i \in Node(o_i)$, $i = 1..m$ s.t. $u_i \prec_n u_{i+1}$, $i = 1..(m-1)$. Thus, $o_i \prec_o o_{i+1}$, $\forall i = 1..(m-1)$. Therefore, objects $o_1, \ldots, o_m$ have AD chain (by Concept 4.2).

Phase 2: If for $u_1 = Node(o_1)[1]$, there exists no chain $(u_1, \ldots, u_m)$, $u_i \in Node(o_i)$, $i = 2..m$ s.t. $u_i \prec_n u_{i+1}$, $i = 1..(m-1)$, then for other node $u_1 \in Node(o_1)$ there exist no such chain either (by Property 3). Therefore, objects $o_1, \ldots, o_m$ have no AD chain (by Concept 4.2). □

Checking AD chain. Algorithm 4 provides the checking of the AD chain of objects. By Property 4, only the chain started with the *first object node* of the *highest object* is checked. Let $o_i \sim o_j$ denote that objects o_i and o_j are at the same level, and $Dw(u)$ denote Dewey label of object node u.

Complexity. $|S_i|$ is usually much smaller than $|ObjNode(o_i)|$. In the worst case, the complexity is $O(\sum_2^m log(|S_i|))$. Usually, $|S_m|$ is the biggest number among $|S_i|$'s. Thus, the complexity is $O(mlog(|S_m|))$.

4.3.3 Identifying Case 3A and Case 3B

Object o has no descendant object if $Class(o)$ is a leaf node in XML schema. This checking operation costs $O(1)$.

Algorithm 4: Checking AD chain w.r.t. objects

```
Input: {o_1, ..., o_m} (ordered decreasingly by hierarchical level of objects)
if ∃o_i, o_j, o_i ≁ o_j then
    return FALSE;
cur = ObjNode(o_1)[1] //Property 4
for i = 2 → m do
    S_i ← {n | n ∈ ObjNode(o_i) s.t. Dw(n) precedes Dw(cur)}
    if ∃u ∈ S_i, cur ≺_a u then
        cur ← u
    else
        return FALSE
return TRUE;
```

4.4 The optimized algorithm

Follows are discussion on the way we filter out duplicated answers for Case 1 and Case 2.

Case 1 ($\mathcal{M}_\mathcal{Q} = \{o\}$). Any object node in $ObjNode(o)$ is the NCON of itself. However, our approach returns only $ObjNode(o)[1]$ and filters out the remaining nodes because they provide duplicated answers.

Case 2 (Objects in $\mathcal{M}_\mathcal{Q} = \{o_1, \ldots, o_m\}$ have AD chain). By Concept 4.2, $LCOA^O(\mathcal{M}_\mathcal{Q}) = \{u \mid u \in ObjNode(o_1)\}$. Moreover, by Property 3, all PSTrees rooted at u's provide duplicated answers. Thus only $ObjNode(o_1)[1]$ is returned and all the remaining duplicated answers are filtered.

Based on Lemma 2 and the above analysis, we propose the optimized algorithm (Algorithm 5) for query mapping processing, in which duplicated answers are filtered out on the fly. As a result, we get the optimal cost $O(1)$ to process query mappings of Case 1 and Case 2.

Algorithm 5: The optimized algorithm

```
Input: A query mapping M_Q = {o_1, ..., o_m}
Output: NCON(M_Q)
NCON(M_Q) ← ∅
//Case 1: M_Q containing only one object
if |M_Q| = 1 then
    NCON(M_Q) = {ObjNode(o_1)[1]}
else
    isAD ← Check AD chain ({o_1, ..., o_m})
    //Case 2: objects in M_Q having AD chain
    if isAD = TRUE then
        NCON(M_Q) = {ObjNode(o_1)[1]} //Property 3
    //Case 3: objects in M_Q having no AD chain
    else
        //finding NCONs in the original O-tree
        LCOA^O(M_Q) ← find LCOAs (M_Q) w.r.t. OT_O
        NCON(M_Q). AddAll (LCOAs^O(M_Q))
        //finding NCONs in the reversed O-tree
        if more than one object having descendant objects then
            LCOA^R(M_Q) ← find LCOAs (M_Q) w.r.t. OT_R
            HCOD(M_Q) ← transfer from LCOA^R(M_Q)
            NCON(M_Q) ← LCOA^O(M_Q) ⋃ HCOD(M_Q))
```

Complexity of Algorithm 5. Complexities for checking conditions and for finding NCONs in each case are given in Table 4.

TABLE 4
Complexities

	Case 1	Case 2	Case 3A	Case 3B
Proportion of queries	α_1	α_2	α_3	α_4
Checking conditions	$O(1)$	C_1	$O(1)$	0
Finding NCONs	$O(1)$	$O(1)$	C_2	$2C_2$

$C_1 = O(mlog(|S_m|))$ where S_m is the number of labels in $ObjNode(o_m)$ used in the checking progress. $C_2 = O(|lbl(o_1| \times m \times log(|lbl(o_m|))$. The total complexity of processing a query mapping is $O(\ \alpha_1 + (\alpha_2 \times C_1) + (C_1 + C_2) \times (\alpha_3 + 2\alpha_4)$. Since the costs of finding NCONs of Case 1 and Case 2 are too small, and C_1 is dominated by C_2, the complexity becomes $O(C_2 \times (\alpha_3 + 2\alpha_4))$ which is always better than $O(2C_2)$ of the algorithm without optimization. Since our approach can return NCONs for query mappings of Case 1 and Case 2 with the optimal $O(1)$ cost, it outperforms all existing systems in terms of efficiency when handling such cases.

4.5 Multiple class path

In this section we handle the case where an object class has multiple paths in an XML schema. The query mapping is re-defined as a set of distinct objects w.r.t. a certain path, each of which matches at least one keyword in the query, and each query keyword has at least a match in the mapping. Let (o_i, p) be the object w.r.t. the object class p, the extension of a query mapping is defined as follows.

CONCEPT 4.3 (A query mapping - Ext) *Given a keyword query $Q = \{k_1, \ldots, k_n\}$, a mapping of query Q is $\mathcal{M}_Q = \cup_{i=1}^{n}\{(o_i, p)\}$ where $(o_i, p) \in Obj(k_i, p)$.*

By redefining the concept of a query mapping, the techniques and optimization to handle each query mapping is still valid.

5 EXPERIMENT

This section evaluates our approach on three aspects including efficiency, effectiveness and quality of the generated reversed O-tree.

5.1 XComplete system

We have developed `XComplete`, a system for XML keyword search, based on all of our ideas including the proposed NCON-based approach, index and optimization techniques. `XComplete` was implemented using Java and was used for experimental evaluation.

5.2 Experimental Setup

Environment. Experiments were performed on a dual-core Intel Xeon CPU 3.0GHz running Windows XP operating system with 4GB of RAM and a 320GB hard disk.

Datasets. Three real datasets were employed including **NBA**[3], which contains all teams and players of the basketball leagues during $1946 - 2004$; **DBLP**[4], which includes all the conference papers during $1959 - 2010$; and **SIGMOD Record**[5], which contains on-line issues of SIGMOD Record. Table 5 gives the statistics of the three datasets.

TABLE 5
Statistics of datasets

Dataset	No. of nodes	No. of object nodes	No. of object classes	No. of words	No. of keywords	Data size
NBA	135,940	1,140	4	223,500	8,302	2.3M
DBLP	17,501,788	8,487,422	9	48,191,004	2,893,195	738M
SIGMOD	31,627	6,449	4	46,311	6,511	500K

Query set. We randomly generated 200 queries from document keywords. To avoid meaningless queries, we filtered out generated queries whose keywords are not related to each other at all. 143 remaining queries include 25, 100 and 18 queries for NBA, DBLP and SIGMOD Record datasets, respectively.

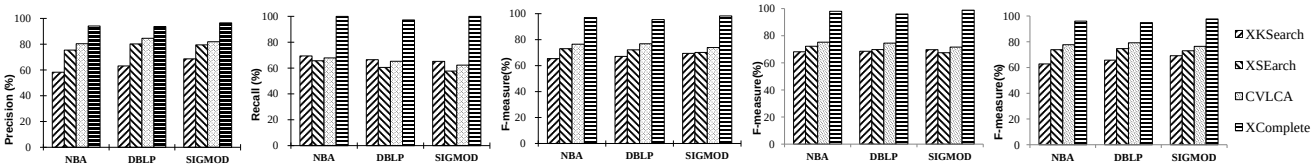
Compared Algorithms. We compared `XComplete` (the optimized algorithm) with the state-of-art algorithms, XKSearch [17], XSEarch [3] and VLCA [7]. We adopted Indexed Lookup Eager Algorithm of XKSearch and VLCAStack Algorithm of VLCA for computation since they achieve better performance.

Metrics. To measure the efficiency, we compared the running time of finding returned nodes, e.g., NCONs in `XComplete` and LCAs in the compared algorithms. For each kind of queries, e.g., 2-keyword query, we selected five queries among 143 generated queries sharing the same properties. For each query, we ran it ten times to get the average response time. We finally reported the average response time of five queries for one kind of query.

To evaluate the effectiveness, we used standard *Precision* ($\mathcal{P}$), *Recall* ($\mathcal{R}$), and *F-measure* ($\mathcal{F}$) metrics. *Precision* measures the percentage of answers that are desired. *Recall* measures the percentage of the desired answers that are output. *F-measure* is the harmonic mean of precision and recall, and is calculated as $\mathcal{F}_\alpha = \frac{(1+\alpha^2)\times\mathcal{P}\times\mathcal{R}}{\alpha^2\times\mathcal{P}+\mathcal{R}}$. $\alpha = 1$, $\alpha = 0.5$ and $\alpha = 2$ correspond to evenly weight to precision and recall, double weight to precision, and double weight to recall respectively.

We randomly selected a subset (35 queries) of 143 generated queries for effectiveness evaluation. To compute precision and recall, we conducted surveys on the above 35 queries and the test datasets. We

3. http://www.nba.com
4. http://dblp.uni-trier.de/xml
5. http://www.dia.uniroma3.it/Araneus/Sigmod/



(a) Precision (b) Recall (c) F_1-measure($\alpha = 1$) (d) F_2-measure($\alpha = 2$) (e) $F_{0.5}$-measure($\alpha = 0.5$)

Fig. 10. Effectiveness Evaluation

asked 25 researchers of our database labs to interpret 35 queries. Interpretations from at least 18 out of 25 researchers are manually reformulated into schema-aware XQuery queries and the results of these XQuery queries are used as the ground truth. Let T and K denote the result set of XQuery queries and that of the compared algorithms respectively. The precision and recall are $\mathcal{P} = \frac{|T \cap K|}{|T|}$ and $\mathcal{R} = \frac{|T \cap K|}{|K|}$.

5.3 Effectiveness Evaluation

Precision. Fig. 10(a) shows the precision of all algorithms, among which, `XComplete` obtains the highest precision for all datasets (higher than 96%) for two reasons. First and most importantly, by applying the NCON semantics, `XComplete` does not return meaningless answers because only object nodes are returned as NCONs. Second, the output presentation and post-processing enable `XComplete` to filter out duplicated answers, and to improve user comprehension, while the compared methods do not focus on the post-processing phase.

Recall. Fig. 10(b) plots the recall of all algorithms, among which, `XComplete` achieves the highest recall (higher than 98%) for all datasets. The difference of recall between `XComplete` and other algorithms is more than 25%. The most important reason is the more completeness of its answer set. When query mappings fall into case 3B, `XComplete` is the only system that is able to return common descendants. The difference in terms of recall is higher than precision. `XComplete` improves both precision and recall, but the more important contribution is improving recall because `XComplete` is the first system to return common descendants.

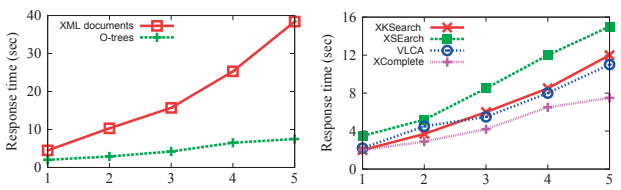
F-measure. Since `XComplete` achieves the best performance in term of both precision and recall, it outperforms the other algorithms and achieves the best F-measure (higher than 97%) as shown in Fig. 10(c)(d)(e).

5.4 Efficiency and Scalability Evaluation

Efficiency. The response time of algorithms is shown in Fig. 11, in which we varied the number of query keywords, the number of matching nodes and the percentage of datasize. `XComplete` outperforms the other algorithms for all datasets because of two reasons. First, `XComplete` searches over the O-tree which is much smaller than the XML document. Second, `XComplete` can filter out duplicated and irrelevant answers on the fly. Particularly, `XComplete` processes query mappings of Case 1 and Case 2 efficiently with the optimal cost $O(1)$. Moreover, the running time of `XComplete` increases at a much slower rate (w.r.t. the number of matching nodes) compared to the other algorithms. This is because the running time of `XComplete` depends on the number of matching objects rather than matching nodes. Among the other algorithms, XSEarch is inefficient since it requires computing an all-pairs interconnection index. XKSearch and VLCA have the similar performance.

Impact of object identification. The impact of object identification on efficiency is shown in Fig 12, in which we varied the number of query keywords. We chose DBLB because its size is biggest and thus the difference between the XML document and the O-tree are significant. Fig 12(a) shows the response time of `XComplete` when it searches over the original and reversed O-trees versus the corresponding XML documents. It shows that `XComplete` runs much faster with the O-trees, especially when the number of keywords increases. This is because the size of the O-trees is much less than that of the XML documents.

Fig 12(b) shows the response time of all compared algorithms over the two O-trees (original and reversed). As can be seen, `XComplete` still outperforms the other algorithms because based on our proposed lemma and optimization techniques, `XComplete` obtains the optimal cost $O(1)$ for query mappings of Case 1 and Case 2 and only needs to use the reversed O-tree for Case 3B. However, the



(a) O-tree vs. XML document (b) O-trees for all algorithms

Fig. 12. Impact of object identification on efficiency [DBLP]

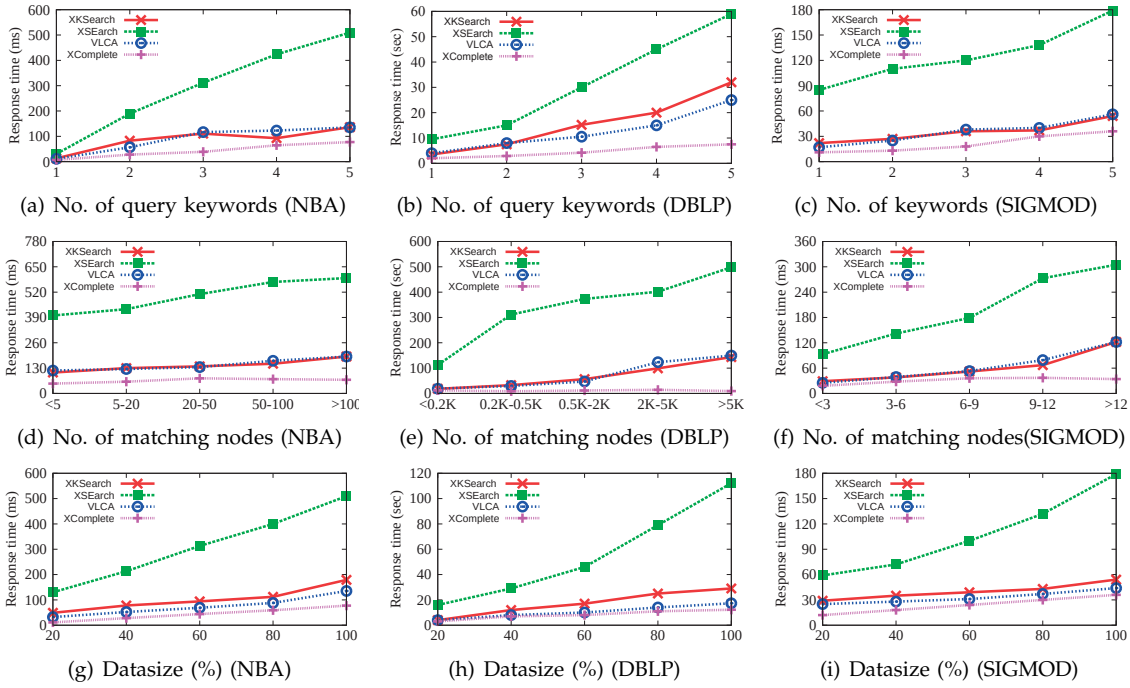


Fig. 11. Efficiency and scalability evaluation on NBA, DBLP and SIGMOD record datasets

difference is not as much as shown in Fig. 11(b) because the other algorithms also improve efficiency when they use the O-trees.

Execution time. Fig. 13 shows the executing time of finding NCONs and generating outputs for 9 queries containing $2-7$ keywords. Low, medium and high frequencies of keywords are denoted as L, M and H and correspond to the number of matching objects between 1-1000, 1000-10000, and above 10000, respectively. $Q(f,k)$ denotes a query containing k keywords with frequency f. As shown, the time for generating outputs is around 24.7% of the time for finding NCONs. They both increase with the number of matching objects due to more mappings to be processed.

The performance with optimization. The running time of the optimized and the basic algorithm of `XComplete` (abbreviated as `XComplete` and `XComplete-NonOPT`) is given in Fig. 14. Let $Q(x,y)$ denote a y-keyword query containing at least one query mapping belonging to Case x. As shown, `XComplete` outperforms `XComplete-NonOPT` significantly when query mappings belong to Case 1 and Case 2 because answers are returned with the optimal cost $O(1)$.

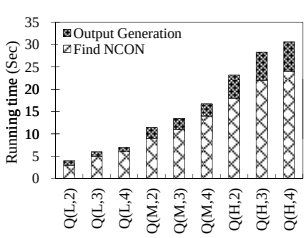


Fig. 13. Finding NCON and generating output

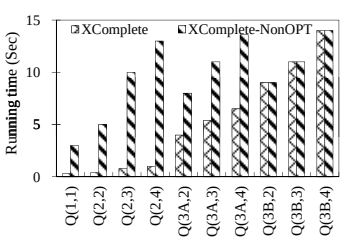


Fig. 14. `XComplete` with and without optimization

5.5 Quality of the extracted and reversed O-trees

To test the quality of the O-tree extracted from XML document, we check the accuracy of the object class and object ID discovery. To test the reversed O-tree, we computed the ratio of the number of satisfied object nodes over the total number of object nodes in the reversed O-tree. The satisfied nodes are those in the reversed O-tree that satisfy the reversed schema (object class) which is manually generated. Quality and time of extracting the O-tree from XML document and generating the reversed O-tree are given in Table 6. As can be seen, the quality of the reversed O-tree depends on the quality of the O-tree extracted from XML document, which is very high since our technique can discover object class and object ID with high accuracy. Once the O-tree is extracted, the reversed O-tree can be derived accurately. The cost of these processes is not expensive since this computation is performed offline and only once. The computation time is also acceptable.

6 Related Work

LCA-based approaches for XML keyword search. XRANK [4] proposes a stack based algorithm to efficiently compute LCAs. XKSearch [17] defines Smallest LCAs (SLCAs) to be the LCAs that do not

TABLE 6
Quality and time of extracting the original O-tree and generating the reversed O-tree

	NBA	DBLP	SIGMOD
Quality of the extracted O-tree (%)	98.5	100	100
Quality of the reversed O-tree (%)	98.5	100	100
Time to extract the O-tree (sec)	3.4	687.5	0.8
Time to generate the reversed O-tree (min)	1.3	25.8	0.4

contain other LCAs. Meaningful LCA (MLCA) [9] incorporates SLCA into XQuery. VLCA [7] and ELCA [18] introduces the concept of valuable/exclusive LCA to improve the effectiveness of SLCA. MaxMatch [11] investigates an axiomatic framework that includes the properties of monotonicity and consistency. Although extensive works have been done on improving the effectiveness of LCA-based approaches, these works commonly return incomplete answer sets for an XML keyword query since they find only common ancestors but ignore common descendants. Moreover, an answer returned by these approaches may be meaningless if LCA nodes are not object nodes.

Object-oriented approaches for XML keyword search. XSeek [10] identifies object (entity) based on * (star) node. However, in many cases, this heuristic is not hold because (*) node can be multi-valued attribute. XReal [1] considers object as parent of a group of attributes, but this cannot distinguish composite attribute and object. Bao et. al. [2] assumed that an object is a group of piece of information. Later, Wu et. al. [16] considered that an object is the parent node of each property node. An object in their work does not represents a real entity in many cases because a group of piece of information or the parent node of properties node can be a relationship type, a composite attribute, or a grouping node. Moreover, none of the above works considers object ID. Thus, they cannot discover duplicated objects and suffer the *incompleteness* problems as LCA-based approaches.

Output presentation and post-processing. eXtract [5] generates result snippets for XML keyword search to help users pick relevant results quickly. Liu et al. [13] propose techniques for result differentiation to investigate and compare multiple relevant results. Liu and Chen [12] cluster the results according to the roles of keywords, and then by the root of the subtree. XSeek [10] outputs the data nodes according to whether they match search predicates or returned nodes. Although these works improve the comprehension of answers, there has been little attention on removing duplicated answers. Moreover, they do not consider merging answers to give a complete understanding about the context to which the set of query keywords can be mapped.

7 Discussion

Object identification brings valuable advantages both in terms of handling the returned results as well as in terms of finding the match. Besides these advantages in the effectiveness, object identification also brings benefits in efficiency. This section discusses advantages of object identification in XML keyword search.

A1. Improving precision. In XML keyword search, if a returned node is just an attribute or a value, the subtree rooted at this node is itself. Intuitively, it is meaningless because it does not provide any supplementary information beside the returned node itself. When object identification is considered, if the returned node is a *non-object node*, we can return the corresponding *object node* instead. Thereby, the result is more *meaningful* because it contains information of the *whole object*, not just an attribute or value. Therefore, the precision of the search is improved because a returned object node satisfies users more than an arbitrary returned node.

A2. Improving recall. Existing approaches for XML keyword search only return common ancestors but none of them is aware of *common descendants*. As discussed in Section 1, common descendants are as meaningful as common ancestors and should be returned as answers as well. To find common descendants, the essential idea is to discover the two objects appearing at different places in an XML document are the same. This cannot be done without object identification. Taking common descendants into the answer set helps the search provide more answers for users and thus improves the recall.

A3. Improving user comprehension. Duplicated answers are from different returned nodes referring to the same object. Such answers annoy users because all of them provide the same information. To filter out duplicated answers, object identification is necessary to discover the same object.

A4. Improving efficiency. All nodes (object node, and its attributes and values) describing an object can be grouped. Each group represents an *object*. Among those nodes, *object node* is the most important one and should be chosen as the *representative* of the group. Therefore, instead of searching the whole XML document, which is usually large, we can dramatically reduce the search space by only consider object nodes (representatives of objects). These object nodes form the O-tree discussed in Section 3.1. Suppose that the average number of attributes for an object class is N, then the number of all nodes in the XML document is at least $2 \times N$ times larger than that of object nodes

(due to not counting attributes and values). This extensively reduces the complexity of the search.

The problems of LCA-based approaches, including meaningless answer, incomplete set of answers and duplicated answer, are caused by the unawareness of object semantics and the dependence on hierarchical structure of answers. Without the semantics of object, they cannot detect duplicated objects. Thus, if a query matches parent objects of these duplicated objects, LCA-based approaches can return only common ancestors and miss common descendants. This is a significant drawback because data-centric XML documents commonly contain a great deal of such duplicated objects. Object identification enables us not only to resolve all of these problems but also to improve the efficiency of the search.

8 Conclusion

This paper shows advantages of object identification in XML keyword search. We introduced the NCON semantics for XML keyword search, by which an answer corresponds to an object and the answer set includes not only common ancestors but also common descendants. We also proposed an NCON-based approach to return a more complete set of meaningful answers by searching over the original and the reversed O-trees. Our post processing steps improve user comprehensive by removing all duplicated answers and merging answers from the same context. Our optimization techniques determine the cases which do not need to use the reversed O-tree and the cases which we can filter out duplicated answers on the fly to improve the efficiency. We have implemented all our ideas in `XComplete` system and use it for experiment evaluation. Experimental results showed that `XComplete` outperforms LCA-based approaches in terms of both effectiveness and efficiency. Thus, the NCON-based approach could be a promising direction for XML keyword search to return a more complete set of distinct meaningful and comprehensive answers.

More broadly, this paper demonstrates the benefit of object orientation in XML. Without even requiring full-blown object orientation, merely by recognizing the concept of objects and object identifiers, we are able to add substantial semantics to XML represented data. We showed how this small amount of additional annotation can greatly benefit keyword search. In future work, we will explore how other XML processing can similarly benefit.

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

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

[5] Y. Huang, Z. Liu, and Y. Chen. Query biased snippet generation in XML search. In *SIGMOD*, 2008.

[6] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell. SAWSDL: Semantic annotations for wsdl and XML schema. *IEEE Internet Computing*, 2007.

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

[8] L. Li, T. N. Le, T. W. Ling, H. Wu, and S. Bressan. Discovering semantics from XML. *TRA3/12, NUS*.

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

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

[11] Z. Liu and Y. Chen. Reasoning and identifying relevant matches for XML keyword search. In *PVLDB*, 2008.

[12] Z. Liu and Y. Chen. Return specification inference and result clustering for keyword search on XML. In *TODS*, 2010.

[13] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. *VLDB*, 2009.

[14] L. Ribeiro and T. Härder. Entity identification in XML documents. In *Grundlagen von Datenbanken*, 2006.

[15] A. Schmidt, M. L. Kersten, and M. Windhouwer. Querying xml documents made easy: Nearest concept queries. In *ICDE*, 2001.

[16] H. Wu and Z. Bao. Object-oriented XML keyword search. In *ER*, 2011.

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

[18] R. Zhou, C. Liu, and J. Li. Fast ELCA computation for keyword queries on XML data. In *EDBT*, 2010.