THE NATIONAL UNIVERSITY
of SINGAPORE

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

TRA9/12

Effective XML Keyword Search with Nearest Common Object Node Semantics

Thuy Ngoc Le, Tok Wang Ling, Chunbin Lin, and Jiaheng Lu

September 2012

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

Effective XML Keyword Search with Nearest Common Object Node Semantics

Thuy Ngoc Le [#1], Tok Wang Ling [#2], Chunbin Lin [†3], Jiaheng Lu [†4]
[#]*National University of Singapore,* [†]*Renmin University of China*
[1,2]{ltngoc,lingtw}@comp.nus.edu.sg, [3,4]{chunbinlin,jiahenglu}@ruc.edu.cn

ABSTRACT

Most approaches for handling XML keyword search are based on Lowest Common Ancestor (LCA) semantics and its extensions such as SLCA, MLCA, VLCA and ELCA. However, these approaches commonly do not return a complete answer set for a query because they can only find the common ancestors of a set of keywords but cannot find their common information appearing at their descendants in an XML document.

In this paper, we introduce a new semantics, called Nearest Common Objects Node (NCON), which guarantees that both common ancestors and common descendants are included in the answer set for a query and therefore enables us to answer a query more completely. We also propose an NCON-based approach for XML keyword search, which exploits not only the index of the original XML document, but also the index of its reversed XML document, and devise optimization techniques to facilitate the process of finding NCONs.

We have developed *XComplete*, a system for our NCON-based approach, which essentially uses the NCON semantics and post-processing techniques, altogether enable XComplete to return an answer set with completeness, meaningfulness, no irrelevance, no duplicate and comprehension to users. The results of our extensive experiments 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 interests. Inspired by hierarchical structure of XML documents, Guo et al. [6] and Schmidt et al. [22] proposed *LCA semantics* (Lowest Common Ancestor) for handling XML keyword queries. Many extensions of the LCA semantics such as SLCA [26], ELCA [28] and VLCA [12] have been proposed to improve effectiveness of XML keyword search. The LCA-based approaches are widely accepted as classical approaches and work well for several types of XML keyword queries. However, since these approaches rely only on LCA, they commonly suffer from two major drawbacks including *incompleteness* of the answer set and *meaninglessness* of the answers as discussed in the following.

The first drawback can be justified by the following motivation examples. Consider an XML data tree as illustrated in Fig. 1 and its corresponding XML schema (Fig. 4). Suppose that a user wants to know the relationship between professors `Stanley` and `Tony`, he/she can input 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. Intuitively, the subtree rooted at the root, i.e., the whole document, is a meaningless answer. Furthermore, these LCA-based approaches *miss some meaningful answers*. Particularly, despite appearing as different nodes, subtrees rooted at `Student(1.1.1)` and `Student(1.2.1)` provide exactly the same information because they have the same object class name `Student` and the same object identifier value `12745`. Therefore, this student is the *common information* related to both Professor `Stanley` and Professor `Tony`, i.e. a student co-supervised by both professors, and should also be a proper answer for query {`Stanley, Tony`}. From now on, we use the term *common descendant* to refer to such common information appearing as descendants of nodes matching query keywords. In fact, if the XML schema is designed in the way that `Student` is the parent of `Professor` for answering student-centric queries efficiently, that student will become an LCA node and thus will certainly be an answer.

Therefore, not only *common ancestors* (LCA nodes), but also *common descendants* should be answer candidates for an XML keyword query. Hence, the answer set returned by LCA-based approaches is *incomplete* because they can only return *common ancestors*, but none of them can discover *common descendants*. The reason is that they fail to detect that a piece of information could be *duplicated* in many places in an XML data tree. Once this situation happens, if a query matches parent nodes of the duplicated information, LCA-based approaches will return only the ancestors (parent nodes) and miss the descendants (duplicated information). This is a significant drawback because data-centric XML documents commonly contain a great deal of such duplicated information. A final answer obtained by LCA-based approaches includes two parts: a returned node (LCA node) and a presentation of answer, e.g., a subtree or a path. Arguably, the presentation of a final 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.

The second drawback of LCA-based approaches is that they may return *meaningless* answers for some queries. For example, for a keyword query {`Clinton, Kennedy`}, an answer such as

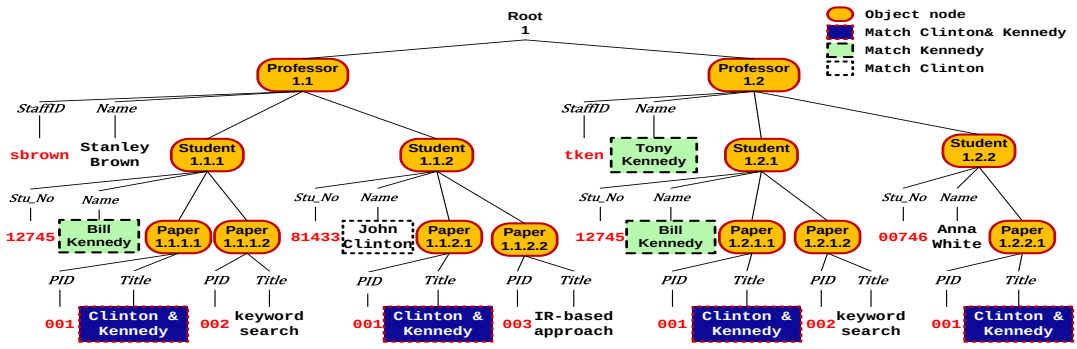


Fig. 1: The original XML data

`title` 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 both `Clinton` and `Kennedy` such as the subtree rooted at node `Paper(1.1.1.1)`. This node can be considered as an *object node*. Intuitively, an object node represents an *object* in the real-world. In an XML data tree, an *object node* usually starts with a tag name, followed by a group of attributes, while a *node* in general can be an *object* node or a *non-object* node (such as an attribute or a value). This example shows that LCA-based approaches return *meaningless answers* when an LCA node is a *non-object* node.

Motivated by the above observations, we propose a new 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 [6, 26, 12]. However, an LCOA must be an object node while an LCA can be any node in an XML data tree. An HCOD is a common descendant of a set of keywords and it has no ancestor which is also a common descendant of that set of keywords. The first feature ensures that an answer of a keyword query is semantically meaningful for users. The second feature integrates *common descendants* into the answers for an XML keyword query to return a more *complete* set of answers to users. To the best of our knowledge, this is the first attempt taking common descendants into account for XML keyword search.

We further propose an NCON-based approach, called XComplete, which applies the NCON semantics for handling XML keyword queries. XComplete uses the *reversed XML document* to find the set of NCONs. For each path of object nodes from the root to a leaf in a given XML document, the corresponding path of object nodes in its reversed XML document has the reversed order with the path in the given XML document. Fig. 2 shows the reversed XML document of the XML document in Fig. 1. We have designed algorithms to automatically generate the reversed XML document corresponding to an XML document. LCOAs can be identified as in similar way as traditional LCA-based approaches [6, 26, 16]. By the reverse mechanism, in order to get HCODs of the original document, XComplete finds LCOAs of the *reversed XML document* since these LCOAs are equivalent to those HCODs in the original document. By considering both *original* and *reversed* XML documents, XComplete can return a more *complete* set of NCONs which includes both LCOAs and HCODs to answer an XML keyword query. This overcomes the key issue of existing LCA-based approaches, which return only the *common ancestors*. This is the most significant contribution of this work.

In addition, to accelerate query processing we develop two types of indexes, namely *object index* and *keyword index*, for both the reversed and the original XML documents. Moreover, we discover several *properties* related to Dewey labels and a *lemma* about the use of the reversed XML document to facilitate the NCON computation. These optimization techniques provide an efficient algorithm for retrieving NCONs.

Last but not least, XComplete can deal with the problem of overwhelming answer sets. Particularly, a query with keywords that can match many nodes in an XML document often results in too many answers which can overwhelm the users. However, there has been little attention to this problem. To avoid such overwhelming answer sets, XComplete provides two canonical steps. First, XComplete *filters* all duplicated answers. Then, answers for the same context are *merged* to provide full *comprehension* about the context that the query is translated. These post-processing techniques provide users with *non-duplicated*, *relevant* and *comprehensive* answers which are easy for users to understand and follow.

Our contributions can be summarized as follows.

- We propose a new semantics, called NCON, for XML keyword search. With the NCON 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 for a query 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 for XML keyword search, called *XComplete*, which uses both the *original* and the *reversed* XML documents to find NCONs and return a more *complete* set of answers for an XML keyword query. Our optimization techniques enable XComplete to return NCONs efficiently.
- Applying the NCON semantics and post-processing techniques enable XComplete to return an answer set that possesses advanced properties including completeness, meaningfulness, no irrelevance, no duplicate and comprehension for users.
- 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 XComplete, our NCON-based approach in Section 3. Section 4 provides implementation issues of

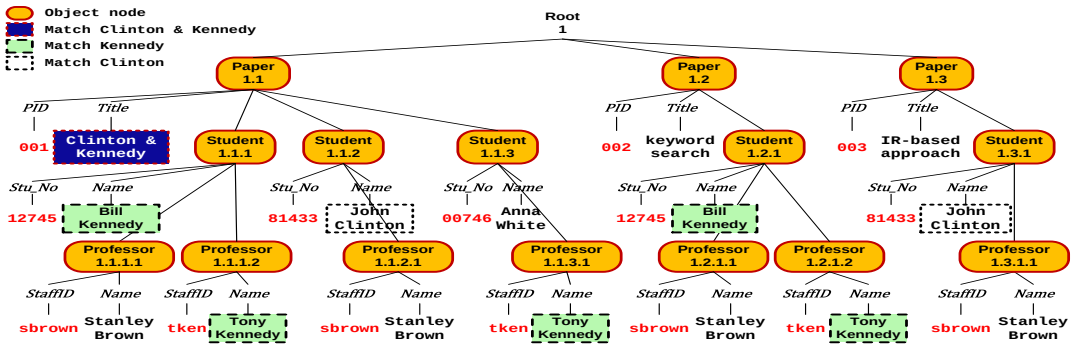


Fig. 2: The reversed XML data of the XML data in Fig. 1

XComplete. Section 5 describes optimization techniques. Experimental results are shown in Section 6. We briefly review related works in Section 7 and conclude the paper in Section 8.

2. THE NEAREST COMMON OBJECT NODE SEMANTICS

In this section, we first describe related notations and concepts used in the paper, and then introduce NCON (*Nearest Common Object Node*) semantics and its role in addressing the drawbacks of LCA-based approaches.

2.1 Terminologies

Notations and relationships between concepts and the NCON semantics are summarized in Table 1 and Fig. 3, respectively.

Table 1: Summary of notations

Notation	Description (for an XML document)
$ObjNode(o)$	The list of *object nodes* referring to object o, by DFS order
$u \prec_a (\succ_a) v$	Node u is an *ancestor* (a *descendant*) of node v
$u \rightsquigarrow v$	The path from object node u to object node v
$ObjNode(u \rightsquigarrow v)$	The set of object nodes on the path from object node u to object node v
$Class(o)$	Object class name of object o
$Obj(k)$	The set of objects matching keyword k

Object. An *object* represents a real-world entity or concept. Each object has an object ID (identifier) value, object attributes and their associated values.

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 should have an *object ID* to uniquely identify objects. Object classes, their object IDs and attributes are described in the XML schema of an XML document.

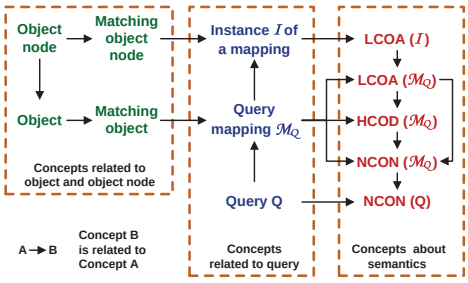


Fig. 3: Relationships of concepts

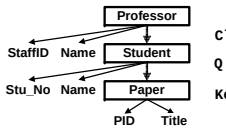


Fig. 4: The schema of the data in Fig. 1

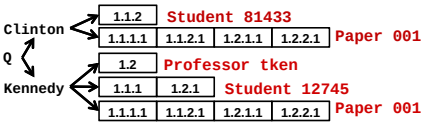


Fig. 5: Matching objects of `Clinton` and `Kennedy`

Object node. An object can have *many occurrences* in an XML document as *object nodes* together with their attributes and values as XML nodes. A real-world object is identified by its *object class name* and *object ID value* while an object node in an XML document is identified by its node identifier (usually Dewey label). Two object nodes represent the same object if both their object class names and their object ID values are the same. For example, in the XML data tree in Fig. 1, `Paper:001` is an *object* with the meaning of the paper with Object ID `PID = 001` and its corresponding *object nodes* are `Paper(1.1.1.1)`, `Paper(1.1.2.1)`, `Paper(1.2.1.1)` and `Paper(1.2.2.1)`.

Non-object node. Apart from object nodes, all other nodes, e.g., attributes and values, in an XML document are non-object nodes.

Matching object vs. matching object node. A keyword k matches an object node u if k matches u or k matches one of non-object nodes associated with u[1]. A keyword k matches an object o if there exists an object node $u \in ObjNode(o)$ such that k matches u (usually all nodes in $ObjNode(o)$ are the same).

Query mapping. 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.

Concept 2.1 **(A query mapping)** Given a keyword query $Q = \{k_1, \ldots, k_n\}$, a mapping of query Q is $\mathcal{M}_Q = \cup_{i=1}^{n}\{o_i\}$, $o_i \in Obj(k_i)$.

For example, consider query `Q = {Clinton, Kennedy}` issued to the data in Fig. 1. Objects and Dewey labels of their object nodes that match the two keywords are shown in Fig. 5. Query mappings of Q are given in Table 2, in which cases are for optimization purposes will be discussed in Section 5.

[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.

Mapping instance. Given a query mapping $\mathcal{M}_\mathcal{Q} = \{o_1, \ldots, o_m\}$, the set of nodes $\mathbb{I} = \{u_1, \ldots, u_m\}$ where $u_i \in ObjNode(o_i)$, $i = 1..m$, is called an *instance* of $\mathcal{M}_\mathcal{Q}$. For example, ${\mathcal{M}_\mathcal{Q}}^2$ has two instances, namely $\mathbb{I}_1$ = {(1.1.1, 1.1.2)} and $\mathbb{I}_2$ = {(1.2.1, 1.1.2)}.

2.2 The NCON semantics

Before introducing the NCON semantics, we present its important components, which are LCOA (*Lowest Common Object Ancestor*) and HCOD (*Highest Common Object Descendant*).

Definition 2.1 **(The LCOA of a set of object nodes)** Object node u is the LCOA of a set of object nodes $\mathbb{I} = \{u_1, \ldots, u_n\}$ if (1) $u \preceq_n u_i \; \forall i = 1..n$ and (2) there exists no object node $v \succ_a u$ s.t. $v \preceq_n u_i \; \forall i = 1..n$.

The LCOA of a set of object nodes is similar to the LCA semantics [6]. 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 2.2 **(An LCOA of a query mapping)** Object node u is an LCOA of a query mapping $\mathcal{M}_\mathcal{Q}$ if

- (1) there exists an instance $\mathbb{I}$ of $\mathcal{M}_\mathcal{Q}$ s.t. u is the $LCOA(\mathbb{I})$
- (2) there exists no object node $v \succ_a u$ s.t. there exists another instance $\mathbb{I}'$ of $\mathcal{M}_\mathcal{Q}$ s.t. v is the $LCOA(\mathbb{I}')$.

Intuitively, an object node is an LCOA of a query mapping $\mathcal{M}_\mathcal{Q}$ if it is an LCOA of an instance of $\mathcal{M}_\mathcal{Q}$ and it has no descendant object node which is also an LCOA of another instance of $\mathcal{M}_\mathcal{Q}$. If $\mathcal{M}_\mathcal{Q}$ has different instances $\mathbb{I}_i$'s, and the LCOAs of these $\mathbb{I}_i$'s are not comparable because they do not have ancestor-descendant relationship, then $\mathcal{M}_\mathcal{Q}$ has different LCOAs, each of which corresponds to an instance of $\mathcal{M}_\mathcal{Q}$. This is similar to the concept of SLCA [26]. However, there are two differences: (1) an LCOA of $\mathcal{M}_\mathcal{Q}$ must be an object node while an SLCA can be either an object node or a non-object node, and (2) an LCOA of $\mathcal{M}_\mathcal{Q}$ is the smallest LCOA *among the LCOAs of an query mapping*, while an SLCA is the smallest LCA *among all the LCAs of the query*. In the later part of this section, we will show that the second difference enables the NCON semantics to avoid false negative problems of SLCA and other LCA extensions. Another concept, whose importance is equivalent to LCOA, is HCOD. Precisely, we have the following definitions for HCOD of a set of object nodes and HCOD of a query mapping. These are counterparts of Defn. 2.1 and Defn. 2.2.

Definition 2.3 **(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 node $\mathbb{H} = \{h_1, \ldots, h_n\}$ is the HCOD of $\mathbb{I}$ if

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

Table 2: Query mappings and their corresponding case

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

Definition 2.4 **(An HCOD of a query mapping)** The set of object nodes $\mathbb{H} = \{h_1, \ldots, h_n\}$ is an HCOD of a query mapping $\mathcal{M}_\mathcal{Q}$ =$\{o_1, \ldots, o_n\}$ if

- (1) there exists an instance $\mathbb{I}$ of $\mathcal{M}_\mathcal{Q}$ s.t. $\mathbb{H}$ is the $HCOD(\mathbb{I})$
- (2) there exists no set of object nodes $\mathbb{H}' = \{h'_1, \ldots, h'_n\}$ where $h_i \prec_a h'_i$, and there exists another instance $\mathbb{I}'$ of $\mathcal{M}_\mathcal{Q}$ s.t. $\mathbb{H}'$ is an $HCOD(\mathbb{I}')$.

Table 3: An example of LCOAs and HCODs

Query	Q = {Clinton, Kennedy}
A mapping of Q	${\mathcal{M}_\mathcal{Q}}^2$ = {Student:81433, Student:12745}
Instance 1 of ${\mathcal{M}_\mathcal{Q}}^2$	$\mathbb{I}_1$ = {Student(1.1.1), Student(1.1.2)}
Instance 2 of ${\mathcal{M}_\mathcal{Q}}^2$	$\mathbb{I}_2$ = {Student(1.1.1), Student(1.1.2)}
LCOA($\mathbb{I}_1$)	Professor(1.1)
LCOA($\mathbb{I}_2$)	The root
LCOA(${\mathcal{M}_\mathcal{Q}}^2$)	{Professor(1.1)}
HCOA($\mathbb{I}_1$)	{Paper(1.1.1.1), Paper (1.1.2.1)}
HCOA($\mathbb{I}_2$)	{Paper(1.2.1.1), Paper (1.1.2.1)}
HCOD(${\mathcal{M}_\mathcal{Q}}^2$)	{{Paper(1.1.1.1), Paper (1.1.2.1)}}

Table 3 gives an example of LCOAs and HCODs. The *complete* set of answers should include both LCOAs and HCODs which motivates the following definitions for NCON semantics.

Definition 2.5 **(An NCON of a query mapping)** An NCON of a mapping $\mathcal{M}_\mathcal{Q}$ is either an LCOA or an HCOD of $\mathcal{M}_\mathcal{Q}$.

Definition 2.6 **(An NCON of a query)** An NCON of a query Q is an NCON of some mapping $\mathcal{M}_\mathcal{Q}$ of Q .

Most extensions of the LCA semantics such as SLCA [26], VLCA [12], and ELCA [28] impose constraints to filter out *less irrelevant* answers, but these constraints may filter out many *meaningful* answers as well. For example, for a query {Clinton, Kennedy} on the data in Fig. 1, node Professor(1.1) should be an answer because it provides the common professor supervising both students Clinton and Kennedy. However, this answer is filtered out by SLCA, VLCA, and ELCA semantics. It is not an SLCA because its descendant Paper(1.1.1.1) is an LCA. It is not a VLCA because node Student(1.1.1) and Student(1.1.2) are homogeneous. It is not an ELCA either because Student(1.1.1) and Student(1.1.2) both contain all the keywords, and after removing them, Professor(1.1) does not contain all the keywords any more.

By exploiting the NCON semantics, our approach can filter out less relevant answers whereas still keeping all meaningful answers. The idea is that we only perform filtering within a query mapping, not across different query mappings. More precisely, for one mapping, we return the smallest LCOA among its LCOAs whereas for across different mappings, we keep all the corresponding smallest LCOAs. The rationale includes two facets. (1) For two LCOAs u_1 and u_2 of the same mapping, u_1 provides less meaningful answers if u_1 is an ancestor node of u_2. (2) Different mappings capture different contexts, hence there should be no filtering across mappings. For example, for query {Clinton, Kennedy} on the data in Fig. 1, both answers for different mappings ${\mathcal{M}_\mathcal{Q}}^2$ and ${\mathcal{M}_\mathcal{Q}}^6$ shown in Fig. 14(b) and Fig. 14(f) are meaningful although Professor(1.1) is the ancestor of Paper (1.1.1.1). These answers capture different meanings. Specifically, one is about the professor who supervises both students Clinton and Kennedy, and the other is about the paper which discusses about Clinton and Kennedy. By not filtering out LCOAs of across *different*

mappings, the NCON semantics can address *the false negative problem* of SLCA, ELCA and VLCA, while filtering out LCOAs of the *same* mapping allows it to remove *less relevant answers* of the LCA semantics. Therefore, applying the NCON semantics can improve both *precision* and *recall* for XML keyword search.

Advantages of the NCON semantics. In summary, our proposed NCON semantics improves both precision and recall over the LCA-based semantics for the following advantages. First, answers are not meaningless since an NCON is not a non-object node. Second, the answer set is more complete with both LCOAs and HCODs. Third, the NCON semantics filters out less relevant answers while still keeps all meaningful answers.

3. OUR NCON-BASED APPROACH

To return a more complete answer set for an XML keyword query, we propose an NCON-based approach, called XComplete, which applies the proposed NCON semantics for handling XML keyword search. This section provides an overview of XComplete while its implementation details are given in the next section.

3.1 The features of XComplete

Now we discuss important features of XComplete and their advantages in generating answer sets with advanced properties.

F1. Working at object level. Working at object level can avoid returning *meaningless* answers when the returned nodes are not object nodes. All *non-object* nodes, e.g., attributes or values, are associated with the *object nodes* they belong to.

F2. Applying the NCON semantics. Applying the NCON semantics for XML keyword search improves both *precision* and *recall* due to getting the *complete* set of *meaningful* answers, and filtering out less relevant answers while still not missing meaningful answers.

F3. Presenting an answer as a partial subtree. An answer presented as a subtree contains a lot of *irrelevant* information while an answer presented as a path does not capture enough information. To address these limitations, XComplete presents an answer as a partial subtree, which is a subtree rooted at an NCON and containing only object nodes on the paths from the returned root to the matching object nodes.

F4. Removing all duplicated answers. *Duplicated answers* 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. Although this is very annoying to users, little attention has been paid for eliminating this problem. XComplete is capable of removing all of them on the fly.

F5. Merging answers. *Merging answers* from the same query mapping allows XComplete to deal with information overload when query keywords match a lot of objects. More importantly, this can provides a complete comprehension about the context that the set of query keywords is mapped to a set of matching objects.

3.2 The properties of the answer set

With the above features of XComplete, the set of answers for an XML keyword query has the following enhanced properties in comparison with that obtained from LCA-based approaches.

- *Completeness* (by F2): this is the most important property of the answer set. By applying the NCON semantics, the answer set is complete with both LCOAs and HCODs.
- *Meaningfulness* (by F1, F2): by returning only object nodes as NCONs, XComplete avoids returning meaningless answers.
- *No redundancy (no duplicate)* (by F3): all duplicated answers are filtered out.
- *No irrelevance* (by F4): an answer presented as a partial subtree does not contain irrelevant information.
- *Comprehension* (by F5): by merging answers from the same context, XComplete provides more comprehensive answers.

3.3 The mechanism of XComplete

The main goal of XComplete is to return a more *complete* set of answers for XML keyword search. To achieve this goal, XComplete applies the NCON semantics, which is finding a set of NCONs, including both LCOAs and HCODs, to answer an XML keyword query. To find NCONs, the essence of the idea is *reverse mechanism* by which HCODs of the given XML data tree are turned into LCOAs in its reversed data tree. The concept of the reversed XML data tree w.r.t. an XML data tree is introduced in Concept 3.3. Before that, we present the related concept of XML object node tree. Fig. 6 shows the XML object node tree extracted from the XML data tree in Fig. 1.

Concept 3.1 **(XML object node tree)** An XML object node 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 object nodes u and v in DT_O where u is the lowest ancestor object node of v, there is a parent-child edge between u and v in OT_O.

Concept 3.2 **(Reversed XML object node tree)** Given an XML object node tree OT_O, the reversed XML object node tree w.r.t. OT_O is the XML object node 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.

Concept 3.3 **(Reversed XML data tree)** Consider two XML data trees DT_O and DT_R, and their corresponding object node trees are OT_O and OT_R respectively, DT_R is the reversed XML data tree w.r.t. DT_O if and only if OT_R is the reversed XML object node tree w.r.t. OT_O.

The completeness of using the reversed XML document. The complete 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 XML document. Thus, although many other XML documents, apart from the reversed document, capture the same information with the original XML document, only the duo of the original and reversed XML document is *self-sufficient* to return the complete set of answers.

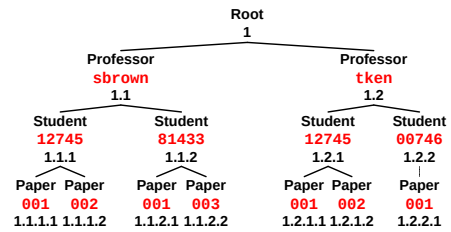


Fig. 6: The object node tree extracted from the data tree in Fig.1

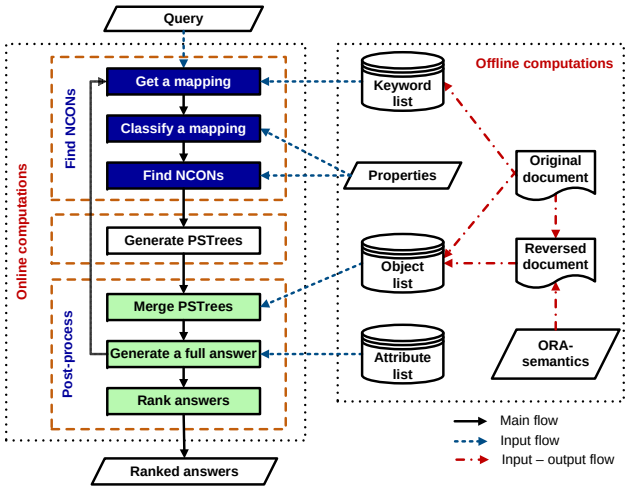


Fig. 7: The process of XComplete

3.4 The process of XComplete

The process of XComplete, as shown in Fig. 7, comprises two components for *offline* and *online* computations. For offline computations, the two main tasks are generating the reversed XML document and indexing. For online computations, the set of ranked answers is returned for an input XML keyword query. There are three phases, namely finding NCONs, generating answers and post-processing answers. The above processes can be briefly described as follows. Details will be discussed in Section 4.

Generating the reversed XML document. The workflow of generating the reversed XML document from an XML document is shown in Fig. 8. The original document D_O and the reversed document D_R are represented as XML data trees. Thus, generating D_R from D_O is equivalent to generating the reversed XML data tree DT_R from the original XML data tree DT_O. This can be done in two main steps.

- Extracting the original XML object node tree OT_O from DT_O by keeping only the object nodes of DT_O.
- Generating the reversed XML object node tree OT_R from OT_O by traversing backward from each leaf node of OT_O to the root to form reversed paths, then connecting all these paths and finally merging nodes of these paths having the same set of ancestors.

After that, recovering DT_R from OT_R is just a reversed step of extracting OT_O from DT_O, by associating non-object nodes to the corresponding object nodes. Details are discussed in Section 4.2.

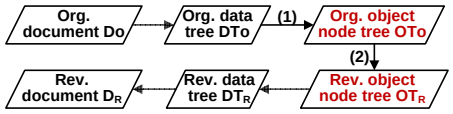


Fig. 8: Generating the reversed XML document

Finding NCONs. Handling each query mapping has the following advantages: (1) *merge answers* of the same query mapping to avoid overwhelming users by a plethora of returned answers and make the merged answer more comprehensive to users, (2) process a query more *efficiently* and *effectively* because many *duplicated* and *irrelevant* answers are filtered out *on the fly*, and (3) improve both *precision* and *recall* because it can address false negative problems of SLCA, ELCA and VLCA, while can still remove less relevant answers returned by the LCA semantics. Therefore, when a query is issued, XComplete first translates it into all possible mappings and then processes those query mappings one by one. For each query mapping, XComplete finds LCOAs in similar way with LCA-based approaches and exploits the reversed XML document to find HCODs.

Output presentation and post-processing. XComplete presents an output as a *partial subtrees* from the returned NCONs to the matching object nodes. After that, it filters all duplicated answers and then merges all answers for the same query mapping. Finally, it ranks all answers from different query mappings.

4. XCOMPLETE SYSTEM

This section describes the implementation of XComplete, our NCON-based approach for XML keyword search.

4.1 Indexes

4.1.1 *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 XML data tree corresponding to a given object. This set of object nodes will be used to find answers from the reversed XML document. Each object, represented as a pair of *object class* and *object ID*, is indexed with a *list of Dewey labels* of its object nodes in both the original and the reversed documents. Only object nodes are labeled while non-object nodes are associated with the object nodes they belong to. Part of the object list of the original data (Fig. 1) and its corresponding reversed data (Fig. 2) is given in Table 4.

Table 4: Object list of the original and reversed documents

Object class	Object ID	Labels in D_O	Labels in D_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
...	...	...	...

4.1.2 *Keyword list*

Similar to traditional *inverted list*, XComplete pre-computes the keyword list to efficiently retrieve the set of object nodes in the *original* document matching a keyword. However, the way XComplete 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. 5 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 5: 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}
...	...

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

4.2 Deriving the reversed XML data tree

This section presents the two main steps for deriving the reversed XML data tree from its original XML data tree.

4.2.1 Extracting the XML object node tree from an XML data tree

In an XML data tree, a node can be an object node or a non-object node. To extract the XML object node tree OT_O from an XML data tree DT_O, we keep only object nodes in DT_O. These object nodes can be identified by using object classes and object IDs in the XML schema of DT_O. Each object node in an XML data tree corresponds an object class in the corresponding XML schema. We assume that the XML schema of an XML document is available. Otherwise, there are existing works on extracting XML schema from XML data [20, 5, 24]. This task is orthogonal to this paper.

In most cases, the semantics of *object classes* and *object IDs* is available because most XML schemas are initially designed based on those semantics. For example, in the XML schema in Fig. 4, `Professor`, `Student` and `Paper` are object classes and `StaffID`, `Stu_No` and `PID` are their object IDs, respectively. If the above semantics is not supported by the database owner or administrator, such semantics can be discovered from XML schema and data by third-party algorithms [21, 11]. We have also designed algorithms [14] to automatically discover such semantics with high accuracy (greater than 98% for object classes and greater than 93% for object IDs). We summarize the heuristics for discovering object classes and object IDs in [14] as follows.

Heuristic 1 **(Object class)** A node in an XML schema refers to an object class if it satisfies either of the following conditions:

- It has an ID attribute as its child node or
- (1) It is the parent of attribute nodes, (2) it has more than one child node; and (3) it has at least one functional dependency among all of its child nodes.

Heuristic 2 **(Object ID)** A node in an XML schema refers to an object ID if it satisfies either of the following conditions:

- It is specified as an ID attribute or
- (1) It is a leaf node, (2) its parent is an object class, and (3) it can functionally determine all of its siblings.

There may be several object ID candidates for an object class. Thus, to choose the best object ID from candidates, we proposed Observation 1 based on structural and linguistic characteristics of object IDs designed in real-world. Moreover, we have designed an algorithm to discover the composite object IDs (with multiple attributes). Due to space constraint, more details are given in [14].

OBSERVATION 1. *In an XML schema, given an object class* o*, its object ID* id_o *is likely to be designed with some of the following features: (1)* id_o *is a single attribute of* o*; (2) The first child node of* o *is (part of)* id_o*; (3)* id_o *contains substring 'Identifier', 'Number', 'Key' or their abbreviations; (4)* id_o *has numeric as (part of) its value, and the numerical part is in sequence and/or has patterns.*

For example, the XML object node tree extracted from the XML data in Fig. 1 is shown in Fig. 6, in which Dewey label is used to identify an object node while object class name and object ID value are used to identify the object that an object node belongs to.

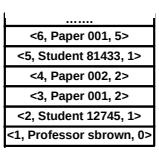


Fig. 9: Array-like-stack

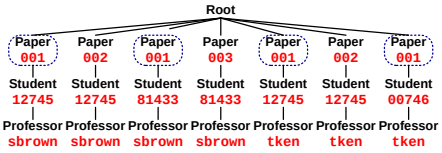


Fig. 10: Intermediate object node tree

4.2.2 Generating the reversed object node tree

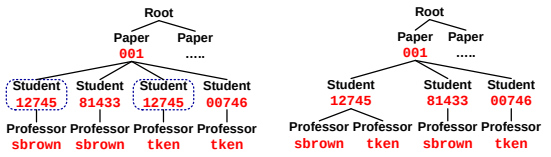
To derive the reversed XML object node tree OT_R from the original XML object node tree OT_O, we first generate the intermediate XML object node tree OT_I from OT_O. Algorithm 1 presents this process. We traverse *backward* from each leaf node to the root in OT_O to form a reversed path. Then, all reversed paths are connected to form OT_I. 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. We traverse OT_O by depth first search (DFS) 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 DFS order (i starts from 1), $objCls(i)$ and $OID(i)$ are the *object class name* and *object ID* of i and $pre(i)$ is the *index* of the *parent* of i. Index of the root is 0 to distinguish with other object nodes. Given the object node tree in Fig. 6, the array-like-stack in Fig. 9 is generated from traversing it by DFS order and Fig. 10 shows the intermediate object node tree w.r.t. it.

Algorithm 1: Generating the intermediate object node tree

Input: The original XML object node tree OT_O
Output: Intermediate XML object node tree OT_I

Variables: Array-like-Stack S to store object nodes in OT_O
Traverse OT_O by DFS 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
      $k \leftarrow pre(k)$

To generate the reversed XML object node tree OT_R from the intermediate object node tree OT_I, we merge object nodes having the same set of ancestors, i.e., ancestor object nodes represent to the same object. Particularly, at the first level of OT_I, we merge branches in which the starting object nodes represent the same object. Then we recursively merge the lower levels. Fig. 11 demonstrates the merging process w.r.t. the intermediate object node in Fig. 10.



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

Fig. 11: Merging branches having the same set of ancestors

Space complexity. For each step, we store two object node trees in the memory. Thus, the space complexity is $O(2N)$ where N is the number of object nodes in an object node 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. It costs one pass $O(K)$ to scan the XML document to extract the object node tree, where K is the total number of nodes in the XML document. To generate the intermediate object node tree, each element in the array-like-stack is processed once so the cost is $O(N)$. To generate the reversed object node tree, for each level of the tree, it costs $O(M)$ where M is the average number of nodes in each level. Therefore, the total cost is $O(K + N + d \times M)$ where d is the depth of the schema tree.

4.3 Query processing

When a query Q is issued on an XML document, XComplete first translates it into all possible query mappings and then processes each query mapping $\mathcal{M}_Q$ separately. For ease of processing, objects in $\mathcal{M}_Q$ are sorted ascendingly by the number of object nodes. Let $LCOA^O(\mathcal{M}_Q)$ and $LCOA^R(\mathcal{M}_Q)$ be the set of LCOAs w.r.t. the original document D_O and the reversed document D_R, respectively. To process a query mapping $\mathcal{M}_Q$, XComplete finds $LCOA^O(\mathcal{M}_Q)$ and $LCOA^R(\mathcal{M}_Q)$. The way that XComplete finds LCOAs w.r.t. an XML document is similar to the Indexed Lookup Eager Algorithm in [26]. By the *reverse mechanism*, XComplete then translates the $LCOA^R(\mathcal{M}_Q)$ to $HCOD(\mathcal{M}_Q)$. NCONs for $\mathcal{M}_Q$ is the union of $LCOA^O(\mathcal{M}_Q)$ and $HCOD(\mathcal{M}_Q)$. Algorithm 2 presents the progress of finding NCONs for a query mapping without optimization. This process is optimized in Section 5.

Algorithm 2: Processing a query mapping

Input: A query mapping $\mathcal{M}_Q = \{o_1, \ldots, o_m\}$
Output: $NCON(\mathcal{M}_Q)$

$NCON(\mathcal{M}_Q) \leftarrow \emptyset$
$LCOA^O(\mathcal{M}_Q) \leftarrow$ find LCOAs $(\mathcal{M}_Q)$ w.r.t. D_O
$LCOA^R(\mathcal{M}_Q) \leftarrow$ find LCOAs $(\mathcal{M}_Q)$ w.r.t. D_R
$HCOD(\mathcal{M}_Q) \leftarrow LCOA^R(\mathcal{M}_Q)$
$NCON(\mathcal{M}_Q) \leftarrow LCOA^O(\mathcal{M}_Q) \bigcup HCOD(\mathcal{M}_Q))$

Complexity. Searching a 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. Therefore, 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 $O(|lbl(o_1| \times 2m \times log(|lbl(o_m|))$.

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: $LCOAs$

for $i = 1 \rightarrow |lbl(o_1)|$ **do**
  $LCOA \leftarrow lbl(o_1)[i]$
  for $j = 2 \rightarrow m$ **do**
    $lm \leftarrow$ get left match $(LCOA, lbl(o_j))$
    $rm \leftarrow$ get right match $(LCOA, lbl(o_j))$
    $lLCOA \leftarrow$ lcoa$(LCOA, lm)$
    $rLCOA \leftarrow$ lcoa$(LCOA, rm)$
    $LCOA \leftarrow$ get smaller LCOA $(lLCOA, rLCOA)$
  $LCOAs$. Add $(LCOA)$
$LCOAs$. RemoveAncestorLCOAs

4.4 Output presentation and post-processing

4.4.1 Output presentation

Several LCA-based approaches present output as a *subtree* [6, 26, 27], and hence irrelevant information from the whole subtree overwhelms users and makes users difficult to identify essential answers. Other LCA-based approaches present the output in the form of a *path* [8, 16], which may miss some useful information which users may want to know. To deal with these limitations, XComplete returns an answer as a *partial subtree* (*PSTree*) $psT(R, N)$ where R and N are the root and the set of object nodes in psT. A PSTree is a subtree rooted at an NCON but all irrelevant nodes are *excluded*. Particularly, a PSTree contains only object nodes on the paths from the root of the PSTree to the matching object nodes. Let $ObjNode(u \rightsquigarrow v)$ be the set of object nodes on the path from object node u to object node v, an answer w.r.t. an instance of a query mapping is defined as follows.

Concept 4.1 **(Answer.)** An answer w.r.t. an instance $\{u_1, \ldots, u_m\}$ of a query mapping is a *PSTree* $psT(R, N)$ where $R = LCOA(u_1, \ldots, u_m)$ and $N = \bigcup_{u_i \in S} ObjNode(R \rightsquigarrow u_i)$.

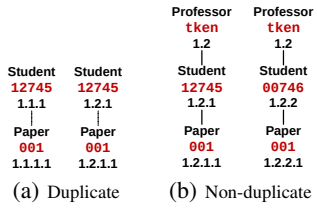
4.4.2 Output post-processing

Removing duplicated answers. Duplicated answers of a query are caused by (1) an PSTree is returned many times, or (2) different PSTrees provide the same information. While some works can filter duplicated answers caused by the first reason, to the best of our knowledge, no work yet deals with the second one because it requires discovering the same information appearing at different places in an XML document. In contrast, XComplete is capable of filtering out both types of duplicated answers, which are defined as follows.

Concept 4.2 **(Duplicated answers)** Two answers corresponding to two PSTrees $psT_1(R_1, N_1)$ and $psT_2(R_2, N_2)$ are duplicated if (1) $|N_1| = |N_2|$, and (2) for all pair of object nodes (u_1, u_2) where $u_1 \in N_1$ and $u_2 \in N_2$ (picked by the same order, e.g., DFS), u_1 and u_2 refer to the same object.

Fig. 12 shows examples of duplicated and non-duplicated PSTrees w.r.t. the data in Fig. 1.

Merging answers for the same query mapping. Intuitively, a query mapping provides a context of a query. This context 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. We do not merge answers from different mappings because these answers come from different contexts. For example, query {`XML, Database`} has a mapping containing two



(a) Duplicate (b) Non-duplicate

Fig. 12: Duplicated and non-duplicated PSTrees

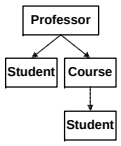


Fig. 13: Multiple class paths

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 meaning of the new answer is that both papers are written by the common authors, and both of them are accepted by the common conference. As can be seen, the merged answer provides more complete comprehension of the context rather than two separated ones. Note that we do not merge answers for *different* mappings. 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.

For illustration of the output presentation and post-processing, the final answers for all mappings of query {Clinton, Kennedy} after post-processed are given in Fig. 14.

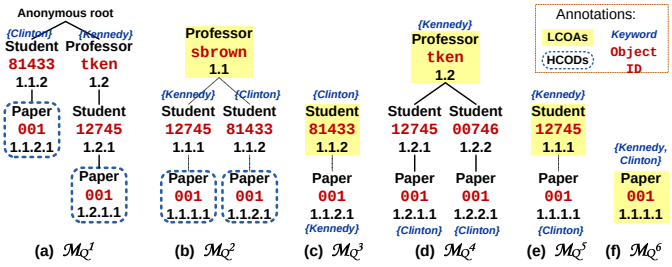


Fig. 14: Final answers of query {Clinton, Kennedy}

4.5 Ranking

The score of an answer PSTree $psT(R, N)$ for a keyword query mainly depends on following three properties. (1) The most important one is how close the query keywords are. Intuitively, if the size of psT is smaller, the query keywords are closer to each other. Let $close(psT)$ denote this property. $close(psT) = \frac{1}{N}$. (2) The way a query keyword matches a node in an XML document: full match or partial match; value match or tag name match; and object ID match or none object ID match. Intuitively, if a keyword matches the value of an object ID, the corresponding answer should be ranked higher. Moreover, full match, value match and Object ID match should have higher score than partial match, tag name match and non object ID match, respectively. (3) The possibility of matching objects: if a keyword, e.g., *Brown*, matches 10 students and 1 color, when a query, e.g., {*Brown, A*}, is issued, *Brown* should most likely be a person rather than a color.

Based on the above observations, we propose a scoring function for a PSTree $psT(R, N)$ of query $Q = \{k_1, \ldots, k_n\}$ as follows.

$$Score(psT) = \alpha\, close(psT) + \beta \sum_{i=1}^{n} match(k_i, v_i) + \theta \sum_{i=1}^{n} p(k_i, v_i)$$

where α, β, θ are tunable parameters, $close(psT)$ measures how close the query keywords are, $match(k_i, v_i)$ is the score of the match between node v_i and keyword k_i, and $p(k_i, v_i)$ is the score calculated by the probability discussed in the third property.

5. OPTIMIZATION AND OTHER ISSUES

In this section, we propose two optimization techniques to ensure that XComplete uses the reversed data to find the set of NCONs only in necessary cases, thereby improving the efficiency of query mapping. Specifically, we classify query mappings into *cases* and optimize the classification by leveraging some properties.

5.1 Classifying query mappings

Since not all query mappings require the reversed data in processing, we classify query mappings into several cases to reveal those in which the reversed XML document is not necessary to process a query mapping.

5.1.1 *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. 13 (extracted from an XML schema by keeping only object classes), object class Student has two occurrences corresponding to two object class paths: Root/Professor/Student and Root/Professor/Course/Student. An object is an instance of an object class in an XML document. We denote $ObjNode(o)$ as 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. 6, Student 12745 is an *ancestor object* of Paper 001.

Concept 5.1 **(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 5.2 **(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 mappings. We classify $\mathcal{M}_\mathcal{Q}$ into three 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) There exists a lowest object that has no descendant object.
  - (Case 3B) All lowest objects have descendant object(s).

The rationale of this classification is to determine whether the reversed XML document D_R provides any new answer for $\mathcal{M}_\mathcal{Q}$ from the original document D_O. For Case 1 ($\mathcal{M}_\mathcal{Q} = \{o\}$), each object node in $ObjNode(o)$ is the NCON of itself and D_R does not provide any additional answer from D_O. For Case 2, the chain among objects in $\mathcal{M}_\mathcal{Q}$ is an answer. If D_R is used, the answer is also this chain with the reversed order. Thus, D_R does not return any new answer. For Case 3A, if there exists a lowest object o_i in $\mathcal{M}_\mathcal{Q}$ that has no descendant object in D_O, then o_i has no ancestor object in D_R and thus D_R provides no answer for $\mathcal{M}_\mathcal{Q}$. Otherwise (Case 3B), D_R may provide new answers. We summarize these arguments in the following lemma.

LEMMA 5.1. *Given a query mapping $\mathcal{M}_\mathcal{Q}$, if $\mathcal{M}_\mathcal{Q}$ is classified into Case 1, Case 2, or Case 3A, the reversed XML document D_R provides no new answers. Thus, D_R is only used in processing $\mathcal{M}_\mathcal{Q}$ of Case 3B.*

PROOF.

Case 1: $\mathcal{M}_\mathcal{Q} = \{o\}$

By Defn. 2.2, we have

- $LCOA^O(\mathcal{M}_\mathcal{Q}) = \{u \mid u \in ObjNode(o)\}$ in D_O
- $LCOA^R(\mathcal{M}_\mathcal{Q}) = \{u \mid u \in ObjNode(o)\}$ in D_R

Therefore D_R provides no new answers.

Case 2: $\mathcal{M}_\mathcal{Q} = \{o_1, \ldots, o_m\}$ and $\{o_1, \ldots, o_m\}$ have AD chain

- W.r.t. D_O, $o_1 \prec_o o_i, i = 2..m$

Thus, $LCOA^O(\mathcal{M}_\mathcal{Q}) = \{u \mid u \in ObjNode(o_1)\}$ in D_O (Defn. 2.2)

- W.r.t. D_R, $o_m \prec_o o_i, i = 1..m-1$

Thus, $LCOA^R(\mathcal{M}_\mathcal{Q}) = \{u \mid u \in ObjNode(o_m)\}$ in D_R (Defn. 2.2)

Therefore, the answer for $(\mathcal{M}_\mathcal{Q})$ in D_R is the same that in D_O. (Although $LCOA^R(\mathcal{M}_\mathcal{Q}) \neq LCOA^O(\mathcal{M}_\mathcal{Q})$, the PSTrees rooted at these LCOAs provide the same information, only in reverse order, because the AD chains rooted at these LCOAs are the same.)

Case 3A: $\mathcal{M}_\mathcal{Q} = \{o_1, \ldots, o_m\}$ such that $\{o_1, \ldots, o_m\}$ has no AD chain, and o_m has no descendant object

If there exists no object o s.t. $o \succ_o o_m$ in D_O, then there exists no object o s.t. $o \prec_o o_m$ in D_R (Concept 3.3).

Moreover, in D_R, o_m is not the common ancestor of objects in $\mathcal{M}_\mathcal{Q}$ since $\{o_1, \ldots, o_m\}$ has no AD chain.

Therefore $LCOA^R(\mathcal{M}_\mathcal{Q}) = \emptyset$. □

The identifications of cases are described in Section 5.1.2 - Section 5.1.4.

5.1.2 Identifying Case 1

Identifying Case 1 requires removing duplicated objects in $\mathcal{M}_\mathcal{Q}$, which can be done by exploiting the following properties.

PROPERTY 5.1. *$ObjNode(o_1) \cap ObjNode(o_2) = \emptyset$ 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 5.1, we can figure out whether two objects are the same or not right after testing their *first object nodes*.

PROPERTY 5.2. *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 5.1). 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 5.2 enables XComplete to remove duplicated objects in $\mathcal{M}_\mathcal{Q}$ efficiently. The complexity of checking two Dewey labels are $O(n)$ where n is the length of the shorter label. However, it is less than $O(L)$ where L is the height of the XML data tree and is a constant for a given XML data tree. So $O(L) = O(1)$. 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 $|\mathcal{M}_\mathcal{Q}| = 1$.

5.1.3 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 Section 5.3. To speed up the checking of Case 2, we exploit some other properties as follows.

PROPERTY 5.3. *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 ($n \geq 3$).*

PROOF. Property 5.3 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 $u \in ObjNode(o_1)$, the number of nodes in $ObjNode(o_2)$ which are descendants of u are the same. Property 5.3 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 $u \in ObjNode(o_i)$, all nodes in $ObjNode(o_{i+1})$ are descendants of u. Thus, for all nodes $v_i \in ObjNode(o_1)$, 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. It is similar to the case where under one object node, there are descendant nodes of different objects. □

Property 5.3 is useful for testing AD chain of a set of objects since we only need to test the subtrees rooted at the *first node* of the highest object as further stated in Property 5.4.

PROPERTY 5.4. *Objects $o_1, \ldots, o_m$ (ordered decreasingly by hierarchical level of objects) have AD chain if there exists an AD chain among objects $u_1, \ldots, u_m$ 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 $(u_1, \ldots, u_m)$, $u_1 = ObjNode(o_1)[1]$, $u_i \in ObjNode(o_i)$, $i = 2..m$ s.t. $u_i \prec_a u_{i+1}$, $i = 1..(m-1)$, then there exists a chain $(u_1, \ldots, u_m)$, $u_i \in ObjNode(o_i)$, $i = 1..m$ s.t. $u_i \prec_a u_{i+1}$, $i = 1..(m-1)$. 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 5.2).

Phase 2: If for $u_1 = ObjNode(o_1)[1]$, there exists no chain $(u_1, \ldots, u_m)$, $u_i \in ObjNode(o_i)$, $i = 2..m$ s.t. $u_i \prec_a u_{i+1}$, $i = 1..(m-1)$, then for other node $u_1 \in ObjNode(o_1)$ there exist no such chain either (by Property 5.3). Therefore, objects $o_1, \ldots, o_m$ have no AD chain (by Concept 5.2). □

Checking AD chain. Algorithm 4 provides the checking of the AD chain of objects. By Property 5.4, 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.

Algorithm 4: Checking ancestor descendant (AD) chain

Input: $\{o_1, \ldots, o_m\}$ (ordered decreasingly by hierarchical level of objects)
1 **if** $\exists o_i, o_j, o_i \nsim o_j$ **then**
2 return FALSE;
3 $cur = ObjNode(o_1)[1]$ `//Property 5.4`
4 **for** $i = 2 \rightarrow m$ **do**
5 $S_i \leftarrow \{n \mid n \in ObjNode(o_i)$ s.t. $Dw(n)$ precedes $Dw(cur)\}$
6 **if** $\exists u \in S_i, cur \prec_a u$ **then**
7 $cur \leftarrow u$
8 **else**
9 return FALSE
10 return TRUE;

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 $O(mlog(|S_m|))$.

5.1.4 Identifying Case 3A and Case 3B

For a lowest object *o* in $\mathcal{M}_\mathcal{Q}$, if $Class(o)$ is a lowest object class in an XML schema, then *o* does not have any descendant object. This checking operation costs $O(1)$. The remaining case is Case 3B.

5.2 Processing a mapping with optimization

We now discuss how XComplete finds NCONs for each case of a query mapping $\mathcal{M}_\mathcal{Q}$, followed by Algorithm 5 for processing a query mapping by exploiting Lemma 5.1.

Case 1 ($\mathcal{M}_\mathcal{Q} = \{o\}$). By Lemma 5.1, D_R is not used. In D_O, any object node in $ObjNode(o)$ is the NCON of itself. However, XComplete 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 Defn. 2.2 and Concept 5.2, $LCOA^O(\mathcal{M}_\mathcal{Q}) = \{u \mid u \in ObjNode(o_1)\}$. Moreover, by Property 5.3, all PSTrees rooted at u's provide duplicated answers. Thus, $LCOA^O(\mathcal{M}_\mathcal{Q}) = \{u_1\}$, where $u_1 = ObjNode(o_1)[1]$. Therefore, $NCON(\mathcal{M}_\mathcal{Q}) = \{ObjNode(o_1)[1]\}$ because D_R is not used (by Lemma 5.1).

Case 3A (Objects in $\mathcal{M}_\mathcal{Q}$ have no AD chain and there exists a lowest object that has no descendant object). By Lemma 5.1, XComplete does not use D_R and only finds LCOAs of $\mathcal{M}_\mathcal{Q}$ w.r.t. the original document.

Case 3B (Objects in $\mathcal{M}_\mathcal{Q}$ have no AD chain and all lowest objects have decendent objects). By Lemma 5.1, the reversed document is necessary to find more answers to return the complete set of answers for $\mathcal{M}_\mathcal{Q}$. Thus, the process of $\mathcal{M}_\mathcal{Q}$ is the same with that for the general case studied in Section 4.3.

Complexity of Algorithm 5. Complexities for checking conditions and for finding NCONs in each case are given in Table 6.

$C_1 = O(mlog(|S_m|))$ 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 XComplete can return NCONs for query mappings of Case 1 and Case 2 immediately with the optimal $O(1)$ cost, it outperforms all existing systems in terms of efficiency when handling such cases.

Algorithm 5: The process of query mapping with optimization

Input: A query mapping $\mathcal{M}_\mathcal{Q} = \{o_1, \ldots, o_m\}$
Output: $NCON(\mathcal{M}_\mathcal{Q})$
1 $NCON(\mathcal{M}_\mathcal{Q}) \leftarrow \emptyset$
2 `//Case 1: MQ containing only one object`
3 **if** $|\mathcal{M}_\mathcal{Q}| = 1$ **then**
4 $NCON(\mathcal{M}_\mathcal{Q}) = \{ObjNode(o_1)[1]\}$
5 **else**
6 $isAD \leftarrow$ Check AD chain ($\{o_1, \ldots, o_m\}$)
7 `//Case 2: objects in MQ having AD chain`
8 **if** $isAD$ = *TRUE* **then**
9 $NCON(\mathcal{M}_\mathcal{Q}) = \{ObjNode(o_1)[1]\}$ `//Property 5.3`
10 `//Case 3: objects in MQ having no AD chain`
11 **else**
12 `//finding NCONs in the original XML document`
13 $LCOA^O(\mathcal{M}_\mathcal{Q}) \leftarrow$ find LCOAs ($\mathcal{M}_\mathcal{Q}$) w.r.t. D_O
14 $NCON(\mathcal{M}_\mathcal{Q})$. AddAll ($LCOAs^O(\mathcal{M}_\mathcal{Q})$)
15 `//finding NCONs in the reversed XML document`
16 **if** *all lowest objects having descendant objects* **then**
17 $LCOA^R(\mathcal{M}_\mathcal{Q}) \leftarrow$ find LCOAs ($\mathcal{M}_\mathcal{Q}$) w.r.t. D_R
18 $HCOD(\mathcal{M}_\mathcal{Q}) \leftarrow$ transfer from $LCOA^R(\mathcal{M}_\mathcal{Q})$
19 $NCON(\mathcal{M}_\mathcal{Q}) \leftarrow LCOA^O(\mathcal{M}_\mathcal{Q}) \bigcup HCOD(\mathcal{M}_\mathcal{Q}))$

Table 6: 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	$2C_2$

5.3 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 5.3 **(A query mapping - Ext)** Given a keyword query $Q = \{k_1, \ldots, k_n\}$, a mapping of query Q is $\mathcal{M}_\mathcal{Q} = \cup_{i=1}^n \{(o_i, p)\}$ where $(o_i, p) \in Obj(k_i, p)$.

By redefining the concept of a query mapping, the technique to handle each query mapping is the same as discussed in Section 5.

6. EXPERIMENT

In this section, we evaluate the performance of XComplete on three aspects including effectiveness, efficiency and the quality of the generated reversed XML document.

6.1 Experimental Setup

Implementation and Environment. XComplete system was implemented using Java and 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**[2], which contains all teams and players of the basketball leagues

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

during 1946 – 2004; **DBLP**[3], which includes all the conference papers during 1959 – 2010; and **SIGMOD Record**[4], which contains on-line issues of SIGMOD Record. Table 7 gives the statistics of the three datasets. We selected 25, 100 and 18 queries for NBA, DBLP and SIGMOD Record datasets, respectively.

Table 7: Statistics of datasets

Dataset	Number of nodes	Number of keywords	Number of distinct keywords	Data size
NBA	135,940	223,500	8,302	2.3M
DBLP	17,501,788	48,191,004	2,893,195	738M
SIGMOD	31,627	46,311	6,511	500KB

Compared Algorithms. We compared XComplete (with optimization) with the state-of-art algorithms, XKSearch [26], XSEarch [4] and VLCA [12]. We adopted Indexed Lookup Eager Algorithm of XKSearch and VLCAStack Algorithm of VLCA for computation since they achieve better performance.

Metrics. To evaluate the effectiveness, we used *Precision* and *Recall*. To compute precision and recall, we manually reformulated queries into schema-aware XQuery queries and used the results of these queries as the ground truth. 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 sharing the same properties and reported the average time. To test the quality of the reversed XML document, we computed the percentage of satisfied object nodes over the total generated object nodes in the reversed XML document.

6.2 Effectiveness Evaluation

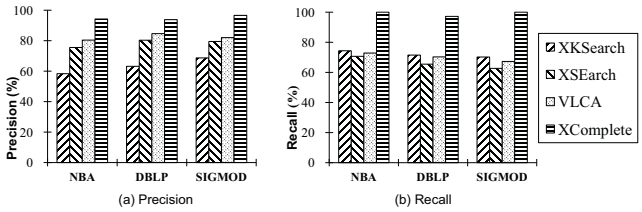


Fig. 15: Comparison on Precision and Recall

Precision. Fig. 15(a) shows the precision of all algorithms, among which, XComplete obtains the highest precision for all datasets (higher than 96%) for two reasons. First, 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 duplicated answers and irrelevant answers, and give more comprehensive answers to users, while the other methods do not focus on the post-processing phase.

Among the other algorithms, XKSearch has the lowest precision since it has false possitive problems and its outputs may contain a lot of irrelevant information under SLCA nodes. XSEarch achieves higher precision than XKSearch since it maintains all-pair connection semantics. VLCA has highest precision because it returns compact connected tree as answers, which is more meaningful for users.

Recall. Fig. 15(b) plots the recall of all algorithms, among which, XComplete achieves the highest recall (higher than 98%) for all datasets. The most important reason is the completeness of its answer set. When query mappings fall into case 3B, XComplete is the only system that is able to return common descendants. Moreover, XComplete improves the recall by filtering only LCOAs of the same query mapping to address false negative problems of LCA-based semantics. 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. Other algorithms achieve similar recall because XKSearch may filter out meaningful SLCAs while XSEarch and VLCA have low recall on several cases due to the requirement of a qualified pattern match.

Output presentation. Fig. 17 shows a snapshot of our XComplete system, which displays the first three results in PSTree form for query {`Clinton, Kennedy`} on a dataset containing similar data to the XML data in Fig. 1. The terms in red match query keywords. The terms in blue refer to object IDs of the same objects. The object nodes with purple and yellow background refer to LCOAs and HCODs, respectively.

6.3 Efficiency and Scalability Evaluation

Efficiency. The response time of algorithms is shown in Fig. 16, 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 three reasons. First, since XComplete operates at object level, the number of matching *objects* to be processed is much smaller than that of all matching nodes processed by the other algorithms. Second, processing each query mapping enables XComplete to filter duplicated and irrelevant answers on the fly. Third, 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.

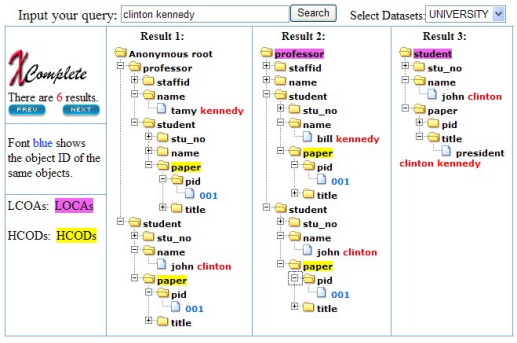


Fig. 17: A snapshot on the demonstration of XComplete

Executing time. Fig. 18 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. Let $Q(f, k)$ denote a query containing k keywords

[3] http://dblp.uni-trier.de/xml

[4] http://www.dia.uniroma3.it/Araneus/Sigmod/

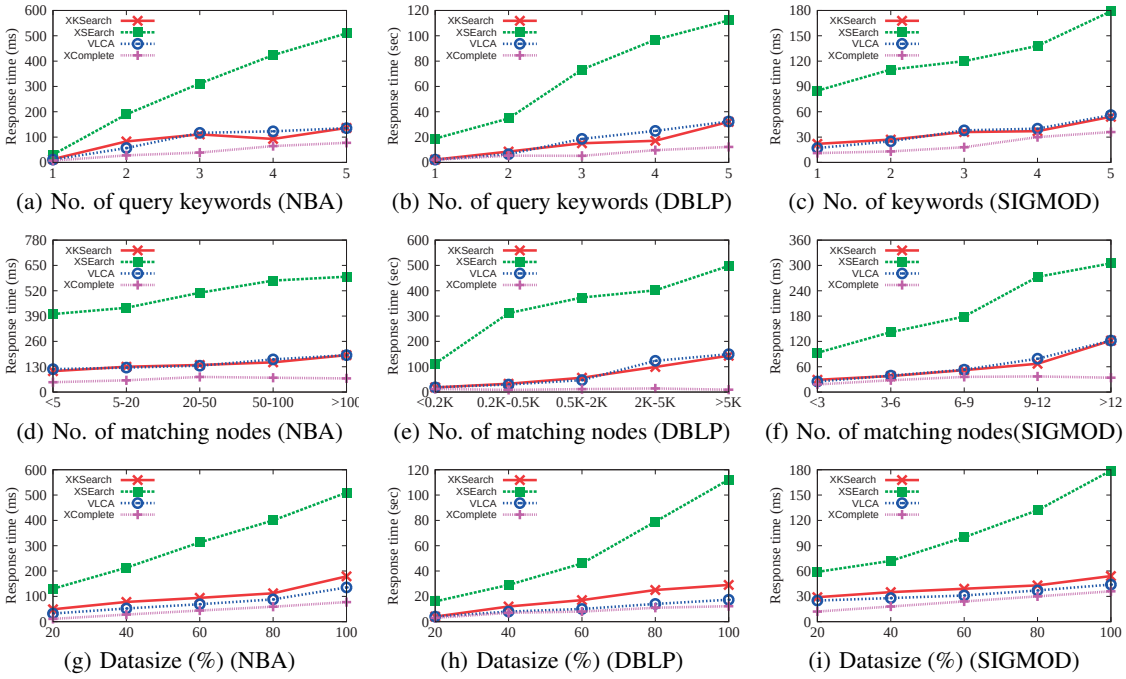


Fig. 16: Efficiency and scalability evaluation on NBA, DBLP and SIGMOD record datasets

with frequency f. As shown, the time for generating outputs is around 24.7% of the time for finding NCONs. Both of them increase with the number of matching objects due to more query mappings to be processed.

The performance with optimization. The running time of XComplete with and without optimization (abbreviated as XComplete and XComplete-NonOPT) is given in Fig. 19. Let $Q(x, y)$ denote a y-keyword query containing at least one query mapping which belongs to Case x. As shown, XComplete outperforms XComplete-NonOPT significantly when query mappings belong to Case 1, Case 2 and Case 3A. These results show that our optimization provides substantial improvement. Especially, for query mappings of Case 1 and 2, answers are returned immediately with the optimal cost $O(1)$.

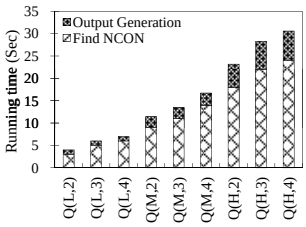


Fig. 18: Finding NCON and generating output

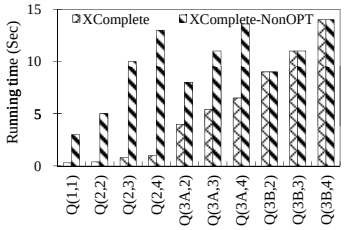


Fig. 19: XComplete with and without optimization

6.4 The quality of the reversed XML data

To generate the reversed XML document D_R from an XML document D_O, we first extract the object node tree OT_O from D_O and then generate D_R from OT_O. To test the quality of D_R (or OT_O), we computed the ratio of the number of satisfied object nodes over the total number of object nodes in D_R (OT_O). The satisfied nodes are those in D_R (OT_O) that satisfy the reversed schema (object class) tree which is manually generated. The quality and time of generating OT_O and D_R are given in Table 8. As can be seen, the quality of the whole process depends on the quality of OT_O, which is very high since our technique can discover object classes and object IDs with high accuracy. Once OT_O is extracted, D_R 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.

Table 8: Quality and time of generating the reversed data

	NBA	DBLP	SIGMOD
Quality of OT_O (%)	98.5	100	100
Quality of D_R (%)	98.5	100	100
Time to extract OT_R (seconds)	3.4	687.5	0.8
Time to generate D_R (minutes)	1.3	45.8	0.4

7. RELATED WORK

LCA-based approaches for XML keyword search. XRANK [6] proposes a stack based algorithm to efficiently compute LCAs. XKSearch [26] defines Smallest LCAs (SLCAs) to be the LCAs that do not contain other LCAs. Meaningful LCA (MLCA) [15] incorporates SLCA into XQuery. VLCA [12] introduces the concept of valuable LCA to improve the effectiveness of SLCA. MaxMatch [17] investigates an axiomatic framework that includes the properties of monotonicity and consistency. Recently, Chen et al. [3] propose join-based algorithms and Zhou et al. [27] use set intersection to provide efficiency for computation of SLCA and ELCA. 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.

Graph-based approaches for XML keyword search. BANKS [2] uses backward search to find group Steiner tree (GST) in labeled, directed graph. Bidirectional [10] improves BANKS by using bidirectional (backward and forward) search. BLINKS [7] proposes a bi-level index to prune and increase the speed of bidirectional search for top-k answers. EASE [13] introduces a unified graph index for keyword search on heterogeneous data. Most recently, [23] provides a fast solution for finding an approximate GST in real-time. However, if an XML document does not contain ID references, the above approaches cannot discover duplicated objects appearing at different places in the document. Thus, they face the same problems with LCA-based approaches, i.e., returning incomplete answer sets for a query.

Object-oriented approaches for XML keyword search. Bao et al. [1] and Wu et al. [25] proposed an object-oriented approach for XML keyword search. However, these works only consider object class but ignore object ID. Thus, they cannot discover duplicated objects and suffer the same problems as LCA-based approaches.

Output presentation and post-processing. eXtract [9] generates result snippets for XML keyword search to help users pick relevant results quickly. Liu et al. [19] propose techniques for result differentiation to investigate and compare multiple relevant results. Liu and Chen [18] cluster the results according to the roles of keywords, and then by the root of the subtree. XSeek [16] 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.

8. CONCLUSION

We introduced the NCON semantics for XML keyword search, by which an answer node should be an object node and the answer set includes not only common ancestors but also common descendants. We also proposed XComplete, an NCON-based approach to return a more complete answer set for a query. Our optimization techniques improve the efficiency of XComplete. The NCON semantics and post processing techniques provide faster browsing of answers due to the enhanced properties of the answer set: completeness, meaningfulness, no irrelevance, no duplicate and comprehension. 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 meaningful and comprehensive answers with minimal effect on computation time.

9. REFERENCES

[1] Z. Bao, J. Lu, T. W. Ling, L. Xu, and H. Wu. An effective object-level XML keyword search. In *DASFAA*, 2010.

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

[3] L. J. Chen and Y. Papakonstantinou. Supporting top-k keyword search in XML databases. In *ICDE*, 2010.

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

[5] M. Eki, T. Ozono, and T. Shintani. Extracting XML schema from multiple implicit XML documents based on inductive reasoning. In *WWW*, 2008.

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

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

[8] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava. Keyword proximity search in XML trees. In *TKDE*, 2006.

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

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

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

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

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

[20] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *SIGMOD*, 1998.

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

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

[23] Y. Tao, S. Papadopoulos, C. Sheng, and K. Stefanidis. Nearest keyword search in XML documents. In *SIGMOD*, 2011.

[24] Q. Y. Wang, J. X. Yu, and K.-F. Wong. Approximate graph schema extraction for semi-structured data. In *EDBT*, 2000.

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

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

[27] J. Zhou, Z. Bao, W. Wang, T. W. Ling, Z. Chen, X. Lin, and J. Guo. Fast slca and elca computation for XML keyword queries based on set intersection. In *ICDE*, 2012.

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