

# An Effective Object-level XML Keyword Search

Zhifeng Bao<sup>1</sup>, Jiaheng Lu<sup>2</sup>, Tok Wang Ling<sup>1</sup>, Liang Xu<sup>1</sup>, and Huayu Wu<sup>1</sup>

<sup>1</sup> School of Computing, National University of Singapore  
{baozhife, lingtw, xuliang, wuhuayu}@comp.nus.edu.sg

<sup>2</sup> School of Informatics, Renmin University of China, jiahenglu@gmail.com

**Abstract.** Keyword search is widely recognized as a convenient way to retrieve information from XML data. In order to precisely meet users' search concerns, we study how to effectively return the targets that users intend to search for. We model XML document as a set of interconnected object-trees, where each object contains a subtree to represent a concept in the real world. Based on this model, we propose object-level matching semantics called *Interested Single Object* (ISO) and *Interested Related Object* (IRO) to capture single object and multiple objects as user's search targets respectively, and design a novel relevance oriented ranking framework for the matching results. We propose efficient algorithms to compute and rank the query results in one phase. Finally, comprehensive experiments show the efficiency and effectiveness of our approach, and an online demo of our system on DBLP data is available at <http://xmldb.ddns.comp.nus.edu.sg>.

## 1 Introduction

With the presence of clean and well organized knowledge domains such as Wikipedia, World Factbook, IMDB etc, the future search technology should appropriately help users precisely finding explicit objects of interest. For example, when people search DBLP by a query "Jim Gray database", they likely intend to find the *publications* object about "database" written by the *people* object "Jim Gray". As XML is becoming a standard in data exchange and representation in the internet, in order to achieve the goal of "finding only the *meaningful* and *relevant* data fragments corresponding to the interested objects (that users really concern on)", search techniques over XML document need to exploit the matching semantics at *object-level* due to the following two reasons.

*First*, the information in XML document can be recognized as a set of real world objects [16], each of which has attributes and interacts with other objects through relationships. E.g. *Course* and *Lecturer* can be recognized as objects in the XML data of Fig. 1. *Second*, whenever people issue a keyword query, they would like to find information about specific objects of interest, along with their relationships. E.g. when people search DBLP by a query "Codd relational model", they most likely intend to find the *publications* object about "relational model" written by "Codd". Therefore, it is desired that the search engine is able to find and extract the data fragments corresponding to the real world objects.

### 1.1 Motivation

Early works on XML keyword search focus on LCA (which finds the Lowest Common Ancestor nodes that contain all keywords) or SLCA (smallest LCA)

semantics, which solve the problem by examining the data set to find the smallest common ancestors [16, 13, 9, 7, 20]. This method, while pioneering, has the drawback that its result may not be meaningful in many cases. Ideally, a practical solution should satisfy two requirements: (1) it can return the *meaningful* results, meaning that the result subtree describes the information at *object-level*; and (2) the result is *relevant* to the query, meaning that it captures users' search concerns. Despite the bulk of XML keyword search literature (See Section 2), the existing solutions violate at least one of the above requirements.

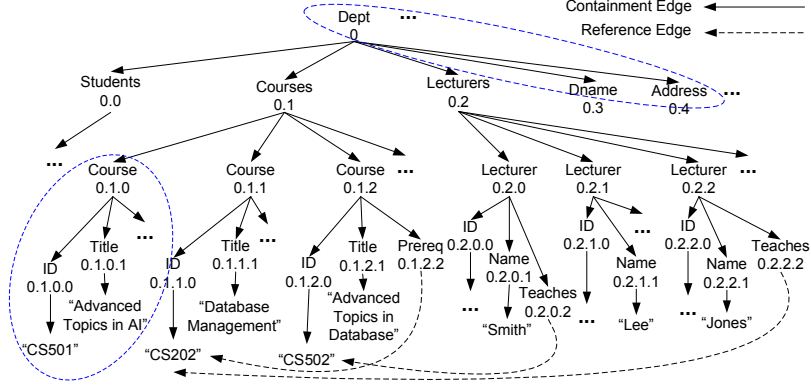


Fig. 1. Example XML data (with Dewey IDs)

Regarding to the *meaningfulness*, the query results should contain enough but non-overwhelming information. i.e. it should be of the same granularity as user's search concern. Unfortunately, the matching semantics proposed so far cannot achieve such goal. For example, for a query "Database" issued on Fig.1, both LCA and SLCA return title:0.1.1.1 and title:0.1.2.1 as results, while the desired result should be two subtrees rooted at Course:0.1.1 and Course:0.1.2, as they encapsulate enough information about a "Database" course. The recent competitors over SLCA include XSeek [18], CVLCA [13], MLCA [15] and MAXMATCH [17]. While those approaches propose some promising and improved matching semantics, the search target identification is still not clearly addressed. More importantly, their inability to exploit ID references in XML data causes some relevant results to be missed.

In order to complement the keyword search over tree model to find more relevant results, ID references in XML data are captured and matching semantics on *digraph data model* are designed. A widely adopted one is *reduced subtree*, which is the minimal subgraph containing all keywords. However, it suffers the same problem as those in tree model, as both of them exploit only the structure of XML data. Even worse, the problem of finding the results by increasing the sizes of reduced subtrees for keyword proximity is NP-hard [14], thus keyword search in digraph data model is heuristics-based and intrinsically expensive.

Regarding to the *relevance*, the query results should be relevant to user's search intention. However, the existing ranking strategies in both tree model [7] and digraph model [12, 8] are built at XML node level, which do not meet user's search concern at object level. Moreover, ranking functions in digraph model even do not distinguish the containment edge and reference edge in XML data.

## 1.2 Our approach

In this paper, we propose to model XML document as a set of object trees, where each real world object  $o$  (with its associated attributes) is encapsulated in an *object tree* whose root node is a representative node of  $o$ ; two object trees are interconnected via a containment or reference edge in XML data. E.g. The part enclosed by a dotted circle in Fig. 1 shows an object tree for Dept and Course.

We propose our object-level matching semantics based on an analysis of user’s search concern, namely ISO (*Interested Single Object*) and IRO (*Interested Related Object*). ISO is defined to capture user’s concern on a single object that contains all keywords, while IRO is defined to capture user’s concern on multiple objects. Compared to previous works, our *object-level* matching semantics have two main advantages. *First*, each object tree provides a more precise match with user’s search concern, so that meaningless results (which even though contain all keywords) are filtered. *Second*, it captures the reference edges missed in tree model, and meanwhile achieves better efficiency than those solutions in digraph model by distinguishing the reference and containment edge in XML.

We design a customized ranking scheme for ISO and IRO results. The ranking function *ISORank* designed for ISO result not only considers the *content* of result by extending the original TF\*IDF [19] to object tree level, but also captures the keyword co-occurrence and specificity of the matching elements. The *IRORank* designed for an IRO result considers both its self similarity score and the “bonus” score contributed from its interconnected objects. We design efficient algorithms and indices to dynamically compute and rank the matched ISO results and IRO results in one phase. Finally, we experimentally compare ISO and IRO algorithms to the best existing methods XSeek [16] and XReal [4] with real and synthetic data sets. The results reveal that our approach outperforms XReal by an order of magnitude in term of response time and is superior to XSeek in term of recall ratio, well confirming the advantage of our novel semantics and ranking strategies. A search engine prototype incorporating the above proposed techniques is implemented, and a demo of the system on DBLP data is available at <http://xml.db.ddns.comp.nus.edu.sg> [3].

## 2 Related Work

**XML tree model** In tree data model, LCA is first proposed to find the lowest common ancestor containing all the keywords in their subtrees. SLCA [20] is proposed to find the smallest LCA that doesn’t contain other LCA in its subtree. XSearch [6] is a variation of LCA, which claims two nodes  $n_1$  and  $n_2$  are related if there is no two distinct nodes with same tag name on the paths from their LCA to  $n_1$  and  $n_2$ . [15] incorporates SLCA into XQuery and proposes a Schema-Free XQuery where predicates in XQuery can be specified through SLCA concept. XSeek [16] studies how to infer the semantics of the search and identify the return nodes by recognizing possible entities and attributes inherently represented in XML data. The purpose of our research is also to maximize the possibility to understand user’s search semantics, while we take a novel perspective by studying new semantics based on ID reference and designing effective ranking strategy.

**XML graph model** The major matching semantics is to find a set of *reduced subtree*  $G'$  of database graph  $G$ , s.t. each  $G'$  is the smallest subgraph containing all keywords. However, the cost of finding all such  $G'$  ranked by size is intrinsically expensive due to its NP-hard nature[14]. Bidirectional expansion is proposed to find ranked reduced subtrees[12], but it requires the entire visited graph in memory, and suffers an inefficiency. BLINKS[8] improves it by designing a bi-level index for result pruning, with the tradeoffs in index size and maintenance cost. XKeyword[11] uses schema information to reduce search space, but its query evaluation is based on the method of DISCOVER [10] built on RDBMS, which cannot distinguish the containment and reference edges to further reduce search space. [5] builds a tree+IDRef model to capture ID references by avoiding the NP-hard complexity. However, this compromise may affect the results' meaningfulness and relevance, which are carefully investigated in this paper.

**Results ranking** In IR field, TF\*IDF similarity [19] is designed to measure the relevance of the keywords and the documents in keyword search over flat documents. XReal [4] addresses the keyword ambiguity problem by designing an XML TF\*IDF on tree model, which takes the structural information of XML into account. XRANK [7] generalizes PageRank to XML element and rank among LCA results, where the rank of each element is computed statically in data preprocessing. In contrast, the ranking functions in this paper are designed to rank on the object trees and are computed dynamically during query processing.

### 3 Data Model

**Definition 1 (Object Tree)** An object tree  $t$  in  $D$  is a subtree of the XML document, where its root node  $r$  is a representative node to denote a real world object  $o$ , and each attribute of  $o$  is represented as a child node of  $r$ .

In an XML document  $D$ , a real-world object  $o$  is stored in form of a subtree due to its hierarchical inherency. How to identify the object trees is orthogonal to this paper; here, we adopt the inference rules in XSeek [16] to help identify the object trees, as clarified in Definition 1. As we can see from Fig. 1, there are 7 object trees (3 Course, 3 Lecturer and 1 Dept), and the part enclosed by a dotted circle is an object tree for Course:0.1.0 and Dept:0 respectively. Note that nodes Students, Courses and Lecturers of Dept:0 are *connection* nodes, which connect the object "Dept" and multiple objects "Student" ("Course" and "Lecturer" ).

**Conceptual connection** reflects the relationship among object trees, which is either a reference-connection or containment-connection defined as below.

**Definition 2 (Reference-connection)** Two object trees  $u$  and  $v$  in an XML document  $D$  have a reference-connection (or are reference-connected) if there is an ID reference relationship between  $u$  and  $v$  in  $D$ .

**Definition 3 (Containment-connection)** Two object trees  $u$  and  $v$  in an XML document  $D$  have a containment-connection if there is a P-C relationship between the root node of  $u$  and  $v$  in  $D$ , regardless of the connection node.

**Definition 4 (Interconnected object-trees model)** models an XML document  $D$  as a set of object trees,  $D=(T,C)$ , where  $T$  is a set of object trees in  $D$ , and  $C$  is a set of conceptual connections between the object trees.

In contrast to the model in XSeek [16], ID references in XML data is considered in our model to find more meaningful results. From Fig. 1, we can find Dept:0 and Course:0.1.0 are interconnected via a containment connection, and Lecturer:0.2.0 and Course:0.1.2 are reference-connected.

## 4 Object Level Matching Semantics

When a user issues a keyword query, his/her concern is either on a single object, or a pair (or group) of objects connected via somehow meaningful relationships. Therefore, we propose *Interested Single Object (ISO)* and *Interested Related Object (IRO)* to capture the above types of users' search concerns.

### 4.1 ISO Matching Semantics

**Definition 5 (ISO)** Given a keyword query  $Q$ , an object tree  $o$  is the *Interested Single Object (ISO)* of  $Q$ , if  $o$  covers all keywords in  $Q$ .

ISO can be viewed as an extension of LCA, which is designed to capture user's interest on a single object. E.g. for a query "database, management" issued on Fig. 1, LCA returns two subtrees rooted at Title:0.1.1.1 and Courses:0.1, neither of which is an object tree; while ISO returns an object tree rooted at Course:0.1.1.

### 4.2 IRO Matching Semantics

Consider a query "CS502, lecturer" issued on Fig. 1. ISO cannot find any qualified answer as there is no single object qualified while user's search concern is on multiple objects. However, there is a Lecturer:0.2.0 called "Smith" who teaches Course "CS502" (via a reference connection), which should be a relevant result. This motivates us to design IRO (*Interested Related Object*).

As a first step to define IRO pair and IRO group, we give a formal definition on the connections among these multiple objects.

**Definition 6 (*n-hop-meaningful-connection*)** Two object trees  $u$  and  $v$  in an XML document have a *n-hop-meaningful-connection* (or are *n-hop-meaningfully-connected*) if there are  $n - 1$  distinct intermediate object trees  $t_1, \dots, t_{n-1}$ , s.t.

1. there is either a reference connection or a containment connection between each pair of adjacent objects;
2. no two objects are connected via a common-ancestor relationship.

**Definition 7 (IRO pair)** For a given keyword query  $Q$ , two object trees  $u$  and  $v$  form an *IRO pair* w.r.t.  $Q$  if the following two properties hold:

1. Each of  $u$  and  $v$  covers some, and  $u$  and  $v$  together cover all keywords in  $Q$ .
2.  $u$  and  $v$  are *n-hop-meaningfully-connected* (with an upper limit  $L$  for  $n$ ).

IRO pair is designed to capture user's concern on two objects that have a direct or indirect conceptual connection. E.g. for query "Smith, Advanced, Database", two object trees Lecturer:0.2.0 and Course:0.1.2 form an IRO pair, as there is a *reference connection* between them. Intuitively, the larger the upper limit  $L$  is, more results can be found, but the relevance of those results decay accordingly. Lastly, *IRO group* is introduced to capture the relationships among three or more connected objects.

**Definition 8 (IRO group)** For a given keyword query  $Q$ , a group  $G$  of object trees forms an *IRO group* if:

1. All the object trees in  $G$  collectively cover all keywords in  $Q$ .
2. There is an object tree  $h \in G$  (playing a role of hub) connecting all other object trees in  $G$  by a  $n$ -hop-meaningful-connection (with an upper limit  $L'$  for  $n$ ).
3. Each object tree in  $G$  is compulsory in the sense that, the removal of any object tree causes property (1) or (2) not to hold any more.

As an example, for query “Jones, Smith, Database” issued on Fig. 1, four objects Course:0.1.1, Course:0.1.2, Lecturer:0.2.0 and Lecturer:0.2.2 form an IRO group (with  $L' = 2$ ), where both Course:0.1.1 and Course:0.1.2 can be the hub. The connection is: Lecturer:0.2.2 “Jones” teaches Course:0.1.1, which is a prerequisite of a “Database” Course:0.1.2 taught by Lecturer:0.2.0 “Smith”.

An object involved in IRO semantics is called the *IRO object*; an ISO object  $o$  can form an IRO pair (or group) with an IRO object  $o'$ , but  $o$  is not double counted as an IRO object.

### 4.3 Separation of ISO & IRO results display

As ISO and IRO correspond to different user search concerns, we separate the results of ISO and IRO in our online demo\* [3], which is convenient for user to quickly recognize which category of results meet their search concern, thus a lot of user efforts are saved in result consumption.

## 5 Relevance Oriented Result Ranking

As another equally important part of this paper, a relevance oriented ranking scheme is designed. Since ISO and IRO reflect different user search concerns, customized ranking functions are designed for ISO and IRO results respectively.

### 5.1 Ranking for ISO

In this section, we first outline the desired properties in ISO result ranking; then we design the corresponding ranking factors; lastly we present the *ISORank* formula which takes both the content and structure of the result into account.

**Object-level TF\*IOF similarity ( $\rho(o, Q)$ )** Inspired by the extreme success of IR style keyword search over flat documents, we extend the traditional TF\*IDF (Term frequency\*Inverse document frequency) similarity [19] to our object-level XML data model, where flat document becomes the object tree. We call it as *TF\*IOF* (Term frequency\*Inverse object frequency) similarity. Such extension is adoptable since the object tree is an appropriate granularity for both query processing and result display in XML. Since TF\*IDF only takes the *content* of results into account, but cannot capture XML’s *hierarchical structure* we enforce the *structure* information for ranking in the following three factors.

**F1. Weight of matching elements in object tree** The elements directly nested in an object may have different weights related to the object. So we provide an optional weight factor for advanced user to specify, where the default weight is 1. Thus, the *TF\*IOF* similarity  $\rho(o, Q)$  of object  $o$  to query  $Q$  is:

$$\rho(o, Q) = \frac{\sum_{\forall k \in o \cap Q} W_{Q,k} * W_{o,k}}{W_Q * W_o}, \quad W_{Q,k} = \frac{N}{1 + f_k}, \quad W_{o,k} = \sum_{\forall e \in attr(o,k)} tf_{e,k} * W_e \quad (1)$$

\* Note: in our previous demo, ISO was named as ICA, while IRO was named as IRA.

where  $k \in o \cap Q$  means keyword  $k$  appears in both  $o$  and  $Q$ .  $W_{Q,k}$  represents the weight of keyword  $k$  in query  $Q$ , playing a role of inverse object frequency (*IOF*);  $N$  is the total number of objects in xml document, and  $f_k$  is number of objects containing  $k$ .  $W_{o,k}$  represents the weight of  $k$  in object  $o$ , counting the term frequency (*TF*) of  $k$  in  $o$ .  $attr(o, k)$  denotes a set of attributes of  $o$  that directly contain  $k$ ;  $tf_{e,k}$  represents the frequency of  $k$  in attribute  $e$ , and  $W_e$  is the adjustable weight of matching element  $e$  in  $o$ , whose value is no less than 1, and  $W_e$  is set to 1 for all the experiments conducted in section 8.

Normalization factor of  $TF*IOF$  should be designed in the way that: on one hand the relevance of an object tree  $o$  containing the query-relevant child nodes should not be affected too much by other query-irrelevant child nodes; on the other hand, it should not favor the object tree of large size (as the larger the size of the object tree is, the larger chance that it contains more keywords). Therefore, in order to achieve such goals, two normalization factors  $W_o$  and  $W_Q$  are designed:  $W_o$  is set as the number of query-relevant child nodes of object  $o$ , i.e.  $|attr(o, k)|$ , and  $W_Q$  is set to be proportional to the size of  $Q$ , i.e.  $|Q|$ .

**F2. Keyword co-occurrence ( $c(o, Q)$ )** Intuitively, the less number of elements (nested in an object tree  $o$ ) containing all keywords in  $Q$  is,  $o$  is likely to be more relevant, as keywords co-occur more closely. E.g. when finding papers in DBLP by a query “XML, database”, a paper whose title contains all keywords should be ranked higher than another paper in “database” conference with title “XML”.

Based on the above intuition, we present  $c(o, Q)$  in Equation 2 (denominator part), which is modeled as *inversely proportional to the minimal number of attributes that are nested in  $o$  and together contain all keywords in  $Q$* . Since this metric favors the single-keyword query, we put the number of query keywords (i.e.  $|Q|$  in nominator part) as a normalization factor.

$$c(o, Q) = \frac{|Q|}{\min(|\{E | E = attrSet(o) \text{ and } (\forall k \in Q, \exists e \in E \text{ s.t. } e.contains(k))\}|)} \quad (2)$$

**F3. Specificity of matching elements ( $s(o, Q)$ )** An attribute  $a$  of an object is fully (perfectly) specified by a keyword query  $Q$  if  $a$  only contains the keywords in  $Q$  (no matter whether all keywords are covered or not). Intuitively, *an object  $o$  with such fully specified attributes should be ranked higher; and the larger the number of such attribute is, the higher rank  $o$  is given.*

*Example 1.* When searching for a person by a query “David, Lee”, a person  $p_1$  with the exact name should be ranked higher than a person  $p_2$  named “David Lee Ming”, as  $p_1$ ’s name fully specifies the keywords in query, while  $p_2$  doesn’t. □

Thus, we model the specificity by measuring the *number of elements in the object tree that fully specify all query keywords*, namely  $s(o, Q)$ .

Note that  $s(o, Q)$  is similar to  $TF*IDF$  at attribute level. However, we enforce the importance of full-specificity by modeling it as a boolean function; thus partial specificity is not considered, while it is considered in original  $TF*IDF$ .

So far, we have exploited both the *structure* (i.e. factors F1,F2,F3) and *content* ( $TF*IOF$  similarity) of an object tree  $o$  for our ranking design. Since there

is no obvious comparability between structure score and content score, we use product instead of summation to combine them. Finally, the  $ISORank(o, Q)$  is:

$$ISORank(o, Q) = \rho(o, Q) * (c(o, Q) + s(o, Q)) \quad (3)$$

## 5.2 Ranking for IRO

IRO semantics is useful to find a pair or group of objects conceptually connected. As an IRO object does not contain all keywords, the relevance of an IRO object  $o$ , namely  $IRORank$ , should consist of two parts: its *self TF\*IOF similarity* score, and the bonus score contributed from its IRO counterparts (i.e. the objects that form IRO pair/group with  $o$ ). The overall formula is:

$$IRORank(o, Q) = \rho(o, Q) + Bonus(o, Q) \quad (4)$$

where  $\rho(o, Q)$  is the *TF\*IOF similarity* of object  $o$  to  $Q$  (Equation 1).  $Bonus(o, Q)$  is the extra contribution to  $o$  from all its IRO pair/group’s counterparts for  $Q$ , which can be used as a *relative* relevance metric for IRO objects to  $Q$ , especially when they have a comparable *TF\*IOF similarity* value. Regarding to the design of  $Bonus$  score to an IRO object  $o$  for  $Q$ , we present three guidelines first.

**Guideline 1: IRO Connection Count.** Intuitively, the more the IRO pair/group that connect with an IRO object  $o$  is, the more likely that  $o$  is relevant to  $Q$ ; and the closer the connections to  $o$  are, the more relevant  $o$  is. ♣

For example, consider a query “interest, painting, sculpture” issued on XMark [2]. Suppose two persons Alice and Bob have interest in “*painting*”; Alice has conceptual connections to many persons about “*sculpture*” (indicated by attending the same auction), while Bob has connections to only a few of such auctions. Thus, Alice is most likely to be more relevant to the query than Bob.

**Guideline 2: Distinction of different matching semantics.** The IRO connection count contributed from the IRO objects under different matching semantics should be distinguished from each other. ♣

Since IRO pair reflects a tighter relationship than IRO group, thus for a certain IRO object  $o$ , the connection count from its IRO pair’s counterpart should have a larger importance than that from its IRO group’s counterpart.

*Example 2.* Consider a query “XML, twig, query, processing” issued on DBLP. Suppose a paper  $p_0$  contains “XML” and “twig”;  $p_1$  contains “query” and “processing” and is cited by  $p_0$ ;  $p_2$  contains the same keywords as  $p_1$ ;  $p_3$  contains no keyword, but cites  $p_0$  and  $p_2$ ;  $p_4$  contains “query” and  $p_5$  contains “processing”, and both cite  $p_0$ . By Definition 7-8,  $p_1$  forms an IRO pair with  $p_0$ ;  $p_2, p_3$  and  $p_0$  form an IRO group;  $p_0, p_4$  and  $p_5$  form an IRO group. Therefore, in computing the rank of  $p_0$ , the influence from  $p_1$  should be greater than that of  $p_2$  and  $p_3$ , and further greater than  $p_4$  and  $p_5$ . □

According to the above two guidelines, the  $Bonus$  score to an IRO object  $o$  is presented in Equation 5.  $Bonus(o, Q)$  consists of the weighted connection counts from its IRO pair and group respectively, which manifests Guideline 1.  $w_1$  and  $w_2$  are designed to reflect the weights of the counterparts of  $o$ ’s IRO pair and group respectively, where  $w_1 > w_2$ , which manifests Guideline 2.

$$Bonus(o, Q) = w_1 * BS_{IRO\_P}(o, Q) + w_2 * BS_{IRO\_G}(o, Q) \quad (5)$$



**Guideline 3: Distinction of different connected object types.** The connection count coming from different conceptually related objects (under each matching semantics) should be distinguished from each other. ♣

*Example 3.* Consider a query  $Q$  “XML, query, processing” issued on DBLP. The bonus score to a “query processing” paper from a related “XML” conference inproceedings should be distinguished from the bonus score coming from a related book whose title contains “XML”, regardless of the self-similarity difference of this inproceedings and book. □

Although the distinction of contributions from different object types under a certain matching semantics helps distinguish the *IRORank* of an IRO object, it is preferable that we can distinguish the precise connection types to  $o$  to achieve a more exact *Bonus* score. However, it depends on a deeper analysis of the relationships among objects and more manual efforts. Therefore, in this paper we only enforce Guideline 1 and Guideline 2. As a result, the IRO bonus from the counterparts of  $o$ 's IRO pair and IRO group is presented in Equation 6-7:

$$BS_{IRO\_P}(o, Q) = \sum_{\forall o' | (o, o') \in IROPair(Q, L)} \rho(o', Q) \quad (6)$$

$$BS_{IRO\_G}(o, Q) = \frac{\sum_{\forall g \in IROGroup(Q, L') | o \in g} BF(o, Q, g)}{|IRO\_Group(o, Q)|} \quad (7)$$

In Equation 6,  $\rho(o', Q)$  is the *TF\*IOF similarity* of  $o'$  w.r.t.  $Q$ , which is adopted as the contribution from  $o'$  to  $o$ . Such adoption is based on the intuition that, if an object tree  $o_1$  connects to  $o'_1$  s.t.  $o'_1$  is closely relevant to  $Q$ , whereas object tree  $o_2$  connects to  $o'_2$  which is not as closely relevant to  $Q$  as  $o'_1$ , then it is likely that  $o_1$  is more relevant to  $Q$  than  $o_2$ . In Equation 7,  $BF(o, Q, g)$  can be set as the self similarity of the object in  $g$  containing the most number of keywords. As it is infeasible to design a one-fit-all bonus function, other alternatives may be adopted according to different application needs.  $L$  (in Equation 6) and  $L'$  (in Equation 7) is the upper limit of  $n$  in definition of *IRO pair* and *IRO group*.

## 6 Index Construction

As we model the XML document as the interconnected object-trees, the *first* index built is the *keyword inverted list*. An object tree  $o$  is in the corresponding list of a keyword  $k$  if  $o$  contains  $K$ . Each element in the list is in form of a tuple  $(Oid, DL, w_{o,k})$ , where  $Oid$  is the id of the object tree containing  $k$  (here we use the dewey label of the root node of object tree  $o$  as its *oid*, as it serves the purpose of unique identification) ;  $DL$  is a list of pairs containing the dewey labels of the exact locations of  $k$  and the associated attribute name;  $w_{o,k}$  is the term frequency in  $o$  (see Equation 1).  $c(o, Q)$  (in Equation 2) can be computed by investigating the list  $DL$ ;  $s(o, Q)$  is omitted in index building, algorithm design and experimental study later due to the high complexity to collect. Therefore, the *ISORank* of an object tree can be efficiently computed. A B+ tree index is built on top of each inverted list to facilitate fast probing of an object in the list.

The *second* index built is *connection table CT*, where for each object  $c$ , it maintains a list of objects that have direct *conceptual connection* to  $c$  in document order. B+ tree is built on top of object id for efficient probes. Since

it is similar to the adjacency list representation of graph, the task of finding the *n-hop-meaningfully-connected* objects of  $c$  (with an upper limit  $L$  for connection chain length) can be achieved through a depth limited (to  $L$ ) search from  $c$  in  $CT$ . The worst case size is  $O(|id|^2)$  if no restriction is enforced on  $L$ , where  $|id|$  is number of object trees in database. However, we argue that in practice the size is much smaller as an object may not connect to every other object in database.

## 7 Algorithms

In this section, we present algorithms to compute and rank the ISO and IRO results.

---

### Algorithm 1: KWSearch

---

**Input:** Keywords:  $KW[m]$ ; Keyword Inverted List:  $IL[m]$ ; Connection Table:  $CT$ ; upper limit:  $L, L'$  for IRO pair and group

**Output:** Ranked object list:  $RL$

```

1 let  $RL = ISO\_Result = IRO\_Result = \{\}$ ;
2 let  $HT$  be a hash table from object to its rank;
3 let  $IL_s$  be the shortest inverted list in  $IL[m]$ ;
4 for each object  $o \in IL_s$  do
5   let  $K_o = getKeywords(IL, o)$ ;
6   if ( $K_o == KW$ ) /*  $o$  is an ISO object */
7     initRank( $o, K_o, KW, HT$ );  $ISO\_Result.add(o)$ ;
8   else if ( $K_o \neq \emptyset$ )
9      $IRO\_Pair = getIROPairs(IL, o, o, CT, L)$  /* Algorithm 2 */
9      $IRO\_Group = getIROGroups(IL, o, o, CT, L', K_o)$  /* Algorithm 3 */
10   $RL = ISO\_Result \cup IRO\_Pair \cup IRO\_Group$ ;

Function initRank( $o, K_o, KW, HT$ )
1  if ( $o$  not in  $HT$ )
2     $HT.put(o.id, computeISORank(o, Q, KW))$ ;

Function computeRank( $o, oList$ )
1  foreach object  $o' \in oList$ 
2     $K_{o'} = getKeywords(IL, o')$ ;
3    if ( $K_{o'} == KW$ ) /*  $o'$  is an ISO object */
4      initRank( $o', K_o, KW, HT$ );  $ISO\_Result.add(o')$ ;
5    else if ( $K_{o'} \neq \emptyset$  AND ( $K_{o'} \cup K_o == KW$ )) /*  $o'$  is IRO object */
6      initRank( $o', K_o, KW, HT$ );
7       $IRO\_Pair.add(o, o')$ ;
8      initRank( $o, K_o, KW, HT$ ); /*  $o$  is an IRO object also */
9      updateIRORank( $o, o', oList, HT$ );

Function updateIRORank ( $o, o', oList, HT$ )
1  update the  $IRORank$  of  $o$  based on Equation 5–7;
2  put the updated ( $o, IRORank$ ) into  $HT$ ;
```

---

The backbone workflow is in Algorithm 1. Its main idea is to scan the shortest keyword inverted list  $IL_s$ , check the objects in the list and their connected objects, then compute and rank the ISO and IRO results. The details are: for each object tree  $o$  in  $IL_s$ , we find the keywords contained in  $o$  by calling function  $getKeywords()$  (line 5). If  $o$  contains all query keywords, then  $o$  is an ISO object, and we compute the  $ISORank$  for  $o$  by calling  $initRank()$ , then store  $o$  together

with its rank into hash table  $HT$ (line 6-7). If  $o$  contains some keywords, then  $o$  is an IRO object, and all its IRO pairs and groups are found by calling functions  $getIROPairs()$  (Algorithm 2) and  $getIROGroups$  (Algorithm 3) (line 8-10).

Function  $computeRank()$  is used to compute/update the ranks of objects  $o'$  in  $oList$ , each forming an IRO pair with  $o$ . For each such  $o'$ , it probes all inverted lists with  $o'$  to check three cases (line 1-2): (1) if  $o'$  is an ISO object containing all query keywords, then its  $ISORank$  is computed and it is added into  $ISO\_Result$  (line 3-4). (2) if both  $o$  and  $o'$  are IRO objects, their  $TF*IOF$  similarity are initialized (if not yet), and their  $IRO_Ranks$  are updated accordingly (line 5-9). Function  $initRank()$  computes the  $ISORank$  by Equation 3 if  $o$  is an ISO object, otherwise computes its  $TF*IOF$  similarity by Equation 1.

Algorithm 2 shows how to find all objects that form IRO pair with an IRO object  $src$ . It works in a recursive way, where input  $o$  is the current object visited, whose initial value is  $src$ . Since two objects are connected via either a reference or containment connection, line 2-3 deal with the counterparts of  $o$  via reference connection by calling  $getConnectedList()$ ; line 4-7 deal with containment connection. Then it recursively finds such counterparts connecting to  $src$  indirectly in a depth limited search(line 8-10).  $getIROGroups()$  in Algorithm 3 works in a similar way, the detail isn't shown due to space limit.

<b>Algorithm 2:</b> getIROPairs ( $IL[m]$ , $src$ , $o$ , $CT$ , $L$ )	<b>Algorithm 3:</b> getIROGroups ( $IL[m]$ , $o$ , $CT$ , $L'$ , $K_o$ )
<pre> /* find all counterparts of o captured   by IRO pair 1  if L == 0 then return ; 2  let oList = getConnectedList(o,CT) ; 3  computeRank(o, oList) ; 4  let ancList = getParent(o) ; 5  computeRank(o, ancList) ; 6  let desList = getChildren(o) ; 7  computeRank(o, desList(o)) ; 8  L = L - 1 ; 9  foreach o' ∈ (oList ∪ ancList ∪ desList)    s.t. o' is not IRO object yet 10  getIROPairs(IL, src, o', CT, L) ; </pre>	<pre> /* find all counterparts of o captured   by IRO group 1  let KS = ∅; count = 0; 2  cList = getConnectedList(o, CT, L'); 3  for n= 1 to L' do 4    foreach o' ∈ cList do 5      KS = getKeywords(IL, o') ∪ KS; 6      if (KS ⊂ KW) then 7        count++; continue; 8      elseif (count &gt; 2) then 9        initialize group g containing such o and            o'; 10     IRO_Group.add(o,g); </pre>

The time complexity of  $KWSearch$  algorithm is composed of three parts: (1) the cost of finding all IRO pairs is:  $O(\sum_{o \in L_s} \sum_{i=1}^L |cList_i(o)| * \sum_{j=1}^k \log |L_j|)$ , where  $L_s$ ,  $o$ ,  $|cList_i(o)|$ ,  $k$  and  $|L_j|$  represent the shortest inverted list of query keywords, an object ID in  $L_s$ , length of the list of objects forming an IRO pair with  $o$  with chain length =  $i$  (limited to  $L$ ), the number of query keywords, and the length of the  $j$ th keyword's inverted list respectively. (2) the cost of finding all IRO groups is:  $O(\sum_{o \in L_s} \sum_{i=1}^{L'} |Q_{L'}| * \sum_{j=1}^k \log |L_j|)$ , where the meaning of each parameter is same as part (1), and  $|Q_{L'}|$  denotes the maximal number of object trees reached from  $o$  by depth limited search with chain length limit to  $L'$ . (3) the cost of finding all ISO objects is:  $O(\sum_{o \in L_s} \sum_{j=1}^{k-1} \log |L_j|)$ . The formation of each cost can be easily derived by tracing Algorithm 1-3.

## 8 Experimental Evaluation

Experiments run on a PC with Core2Duo 2.33GHz CPU and 3GB memory, and all codes are implemented in Java. Both real dataset DBLP(420 MB) and synthetic dataset XMark(115 MB) [2] are used in experiments. The inverted lists and connection table are created and stored in the disk with Berkeley DB [1] B+ trees. An online demo [3] of our system on DBLP, namely ICRA, is available at <http://xmldb.ddns.comp.nus.edu.sg>.

### 8.1 Effectiveness of ISO and IRO Matching Semantics

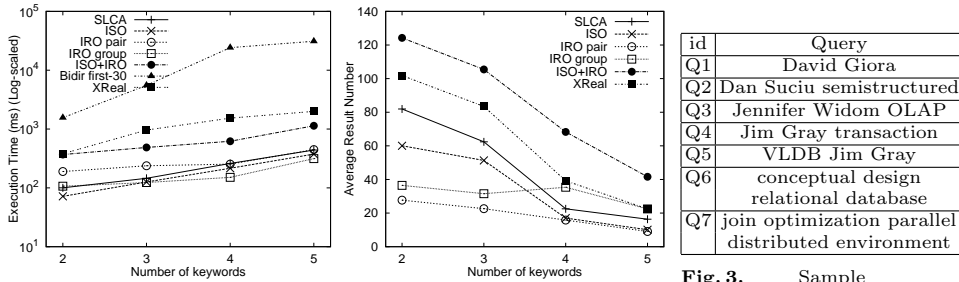
In order to evaluate the quality of our proposed ISO and IRO semantics, we investigate the overall recall of ISO, ISO+IRO with XSeek [16], XReal [4] and SLCA [20] on both DBLP and XMark. 20 queries are randomly generated for each dataset, and the result relevance is judged by five researchers in our database group. From the average recall shown in Table 1, we find: (1) ISO performs as well as XReal and XSeek, and is much better than SLCA. It is consistent with our conjecture that the search target of a user query is usually an object of interest, because the concept of object indeed is implicitly considered in the design of ISO, XReal and XSeek. (2) ISO+IRO has a higher recall than ISO alone, especially for queries on XMark, as there are more ID references in XMark that bring more relevant IRO results. In general, IRO semantics do help find more user-desired results while the other semantics designed for tree data model cannot.

Data	SLCA	XSeek	XReal	ISO	ISO+IRO
DBLP	75%	82.5%	84.1%	84.1%	90.5%
XMark	55.6%	63.8%	60.4%	62.2%	80.7%

Data	R-rank			MAP		
	XReal	ISO	IRO	XReal	ISO	IRO
DBLP	0.872	0.877	0.883	0.864	0.865	0.623
XMark	0.751	0.751	0.900	0.708	0.706	0.705

Table 1. Recall Comparison

Table 2. Ranking Performance Comparison



(a) Execution time (b) Total result number  
Fig. 2. Efficiency and scalability tests on DBLP

id	Query
Q1	David Giora
Q2	Dan Suciu semistructured
Q3	Jennifer Widom OLAP
Q4	Jim Gray transaction
Q5	VLDB Jim Gray
Q6	conceptual design relational database
Q7	join optimization parallel distributed environment

Fig. 3. Sample queries on DBLP

### 8.2 Efficiency & Scalability test

Next, we compare the efficiency of our approach with SLCA and XReal [4] in tree model, and Bidirectional expansion [12] (Bidir for short) in digraph model. For each dataset, 40 random queries whose lengths vary from 2 to 5 words are generated, with 10 queries for each query size. The upper limit of connection chain length is set to 2 for IRO pair and 1 for IRO group, and accordingly we modify Bidir to not expand to a node of more than 2-hops away from a keyword node for a fair comparison. Besides, since Bidir searches as small portion of a graph as possible and generates the result during expansion, we only measure

its time to find the first 30 results. The average response time on cold cache and the number of results returned by each approach are recorded in Fig. 2 and 4.

The log-scaled response time on DBLP is shown in Fig. 2(a), and we find: (1) Both SLCA and ISO+IRO are about one order of magnitude faster than XReal and Bidir for queries of all sizes. SLCA is twice faster than ISO+IRO, but considering the fact that ISO+IRO captures much more relevant results than SLCA (as evident from Table 1), such extra cost is worthwhile and ignorable. (2) ISO+IRO scales as well as SLCA w.r.t the number of query keywords, and ISO alone even has a better scalability than SLCA.

From Fig. 2(b), we find the result number of ISO is a bit smaller than that of SLCA, as ISO defines qualified result on (more restrictive) object level. Besides, ISO+IRO finds more results than SLCA and XReal, because many results that are connected by ID references can be identified by IRO. The result for XMark (see Fig. 4) is similar to DBLP, and the discussion is omitted due to space limit.

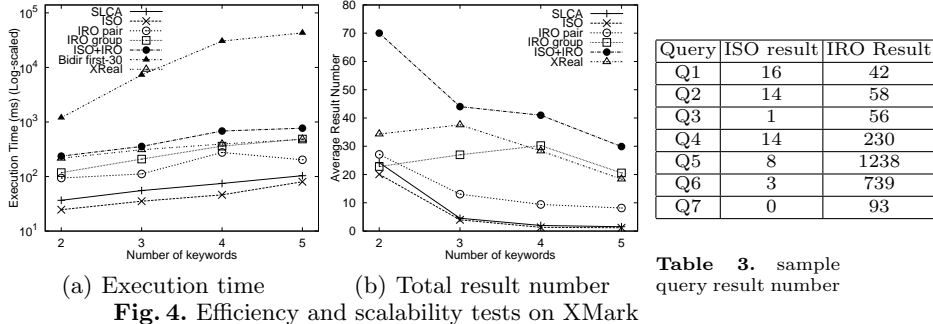


Fig. 4. Efficiency and scalability tests on XMark

### 8.3 Effectiveness of the Ranking Schemes

To evaluate the effectiveness of our ranking scheme on ISO and IRO results, we use two widely adopted metrics in IR: (1) Reciprocal rank (R-rank), which is 1 divided by the rank at which the first relevant result is returned. (2) Mean Average Precision (MAP). A precision is computed after each relevant one is identified when checking the ranked query results, and MAP is the average value of such precisions. R-Rank measures how good a search engine returns the first relevant result, while MAP measures the overall effectiveness for top-k results.

Here, we compute the R-rank and MAP for top-30 results returned by ISO, IRO and XReal, by issuing the same 20 random queries as describe in section 8.1 for each dataset. Specificity factor  $s(o, Q)$  is ignored in computing *ISORank*; in computing the *IRORank*,  $w_1 = 1$  and  $w_2 = 0.7$  are chosen as the weights in Equation 5. The result is shown in Table 2. As ISO and XReal do not take into account the reference connection in XML data, it is fair to compare ISO with XReal. We find ISO is as good as XReal in term of both R-rank and MAP, and even better on DBLP’s testing. The ranking strategy for IRO result also works very well, whose average R-rank is over 0.88.

Besides the random queries, we choose 7 typical sample queries as shown in Fig. 3: Q2-Q4 intend to find publications on a certain topic by a certain author; Q5 intends to find publications of a particular author on a certain conference.

In particular, we compare our system [3] with some academic search engines such as Bidir in digraph model [12], XKSearch employing SLCA [20] in tree model, with commercial search engines, i.e. Google Scholar and Libra<sup>†</sup>. Since both Scholar and Libra can utilize abundant of web data to find more results than ours whose data source only comes from DBLP, it is infeasible and unfair to compare the total number of relevant results. Therefore, we only measure the number of top-k relevant results, where k=10, 20 and 30.

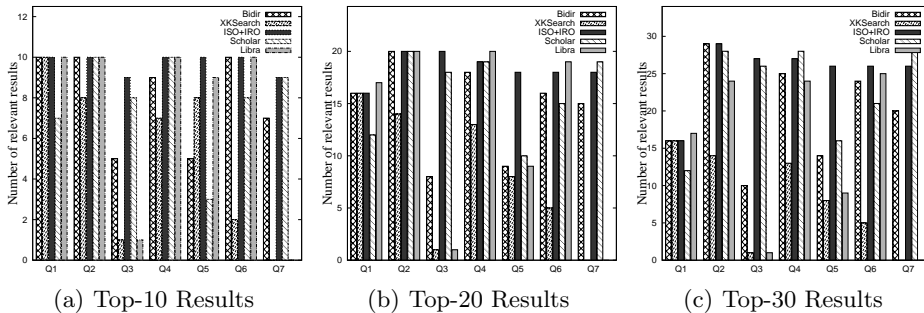


Fig. 5. Result quality comparison

Since our system separates ISO results and IRO results (as mentioned in section 4.3), top-k results are collected in the way that, all ISO results are ordered before the IRO results. The total number of ISO results and IRO results are shown in Table 3, and the comparison for the top-30 results is shown in Fig. 5.

*First*, we compare ISO+IRO with Bidir and XKSearch. For queries that have both ISO and IRO results (e.g. Q1-Q6), our approach can find more relevant results, and rank them in most of the top-30 results. There is no ISO result for Q7, XKSearch also returns nothing; but 26 IRO results are actually relevant.

*Second*, we compare ISO+IRO with Libra and Scholar. From Fig. 5, we find our approach is comparable with Scholar and Libra for all sample queries. In particular, ISO+IRO is able to rank the most relevant ones in top-10 results for most queries, because its top-10 precision is nearly 100% for most queries, as evident in Figure 5(a). In addition, as Libra only supports keyword conjunction (similar to our ISO semantics), it does not work well for Q3 and Q7, as there is only 1 and 0 result containing all keywords for Q3 and Q7. As shown in Fig. 5(a), Scholar only finds 3 relevant results for Q5 in its top-10 answers, probably because keywords “Jim” and “Gray” appear in many web pages causes many results that don’t contain “VLDB” to still have a high rank, which is undesired.

Thirdly, as shown in Fig. 5, the average recall for each query generated by our ISO+IRO is above 80% at each of the three top-k levels, which confirms its advantage over any other approach.

## 9 Conclusion and Future Work

In this paper, we build a preliminary framework for object-level keyword search over XML data. In particular, we model XML data as the interconnected object-trees, based on which we propose two main matching semantics, namely ISO

<sup>†</sup> Google Scholar: <http://scholar.google.com>. Microsoft Libra: <http://libra.msra.cn>

(*Interested Single Object*) and IRO (*Interested Related Object*), to capture different user search concerns. A customized ranking scheme is proposed by taking both the structure and content of the results into account. Efficient algorithms are designed to compute and rank the query results in one phase, and extensive experiments have been conducted to show the effectiveness and efficiency of our approach. In future, we plan to investigate how to distinguish the relationship types among objects and utilize them to define more precise matching semantics.

## 10 Acknowledgement

Jiaheng Lu was partially supported by 863 National High-Tech Research Plan of China (No: 2009AA01Z133, 2009AA01Z149), National Science Foundation of China (NSFC) (No.60903056, 60773217), Key Project in Ministry of Education (No: 109004) and SRFDP Fund for the Doctoral Program(No.20090004120002).

## References

1. Berkeley DB. <http://www.sleepycat.com/>.
2. <http://www.xml-benchmark.org/>.
3. Z. Bao, B. Chen, T. W. Ling, and J. Lu. Demonstrating effective ranked xml keyword search with meaningful result display. In *DASFAA*, 2009.
4. Z. Bao, T. Ling, B. Chen, and J. Lu. Effective xml keyword search with relevance oriented ranking. In *ICDE*, 2009.
5. B. Chen, J. Lu, and T. Ling. Exploiting id references for effective keyword search in xml documents. In *DASFAA*, 2008.
6. S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSearch: A semantic search engine for XML. In *VLDB*, 2003.
7. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK:ranked keyword search over XML documents. In *SIGMOD*, 2003.
8. H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD*, 2007.
9. V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava. Keyword proximity search in XML trees. In *TKDE*, pages 525–539, 2006.
10. V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, 2002.
11. V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *ICDE*, 2003.
12. V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, and R. Desai. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.
13. G. Li, J. Feng, J. Wang, and L. Zhou. Effective keyword search for valuable lcas over xml documents. In *CIKM*, 2007.
14. W. Li, K. Candan, Q. Vu, and D. Agrawal. Retrieving and organizing web pages by information unit. In *WWW*, 2001.
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. volume 1, pages 921–932, 2008.
18. Z. Liu, P. Sun, Y. Huang, Y. Cai, and Y. Chen. Challenges, techniques and directions in building xseek: an xml search engine. volume 32, pages 36–43, 2009.
19. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*.
20. Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD*, 2005.