THE NATIONAL UNIVERSITY
of SINGAPORE

School of Computing
Lower Kent Ridge Road, Singapore 119260

TRC5/07

ICRA: Effective Semantics for Ranked XML Keyword Search

Bo CHEN, Jiaheng LU and Tok Wang LING

May 2007

Technical Report

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.

JAFFAR, Joxan
Dean of School

ICRA: Effective Semantics for Ranked XML Keyword Search

Bo Chen Jiaheng Lu Tok Wang Ling
School of Computing, National University of Singapore
3 Science Drive 2, Singapore 117543
{chenbo,lujiahen,lingtw}@comp.nus.edu.sg

ABSTRACT

Keyword search is a user-friendly way to query XML databases. Most previous efforts in this area focus on keyword proximity search in XML based on either tree data model or graph (or digraph) data model. Tree data model for XML is generally simple and efficient for keyword proximity search. However, it cannot capture connections such as ID references in XML databases. In the contrast, techniques based on graph (or digraph) data model capture connections, but are generally inefficient to compute. In this paper, we propose *interconnected object trees* model for keyword search to achieve the efficiency of tree model and meanwhile to capture the connections such as ID references in XML by fully exploiting the property and schema information of XML databases. In particular, we propose ICA (Interested Common Ancestor) semantics to find all predefined interested objects that contain all query keywords. We also introduce novel IRA (Interested Related Ancestors) semantics to capture the conceptual connections between interested objects and include more objects that only contain some query keywords. Then, a novel ranking metric, *RelevanceRank*, is studied to dynamically assign higher ranks to objects that are more relevant to a given keyword query according to the conceptual connections in IRAs. We design and analyze efficient algorithms for keyword search based on our data model; and experiment results show our approach is efficient and outperforms most existing systems in terms of result quality. A prototype of our ICRA system ($ICRA = ICA + IRA$) on the updated 321M DBLP data is available at http://xmldb.ddns.comp.nus.edu.sg/.

1. INTRODUCTION

Keyword search is a proven user-friendly way of querying HTML documents in the World Wide Web. Keyword search is also well-suited to XML documents since it allows users to easily pose queries without the knowledge of complex query languages and/or the structure of the underlying data.

Over the recent years, extensive research efforts have been conducted on efficient and effective processing of keyword search in XML databases. Majority of them focus on keyword proximity search, which generally assumes closer co-occurrence of all keywords of a query indicates a better result. In XML context, the problem of keyword proximity search can be reduced to the problem of finding the smallest sub-structures of XML that contain all the query keywords.

There are two main existing data model for keyword proximity search in XML documents, *tree data model* and *general graph* (or *digraph*) *data model*. In tree data model ([18, 9, 21, 16, 10]), the Smallest LCA (Smallest Lowest Common Ancestor) [21, 16][1] is an effective representation of the smallest sub-structure for keyword proximity search. We say an XML node n *contains* a keyword k if k appears in PCDATA or CDATA (or tag names depending on whether or not the application wants to include tag names in keyword search) of n or descendants of n. A subtree rooted at n is in the Smallest LCA results of a list of keywords if n contains all query keywords and no descendant of n contains all query keywords. For example, consider the XML data in Fig. 1, with Dewey (or called prefix) labeling scheme. The answer to the keyword query "Widom Lorel" is the first "inproceeding" subtree, as Smallest LCA(1.1.1, 1.2.1)=1. Smallest LCAs are efficient to evaluate by simply computing the longest prefix of the Dewey labels as we can see in the example. However, the tree data model cannot exploit the ID references in XML data which usually indicate relevance and preference between objects in real world. For example, the important citation information as ID references in bibliographic data cannot be captured in tree model. On the other hand, XML documents can also be modeled as digraphs when ID reference edges are taken into account ([8, 13, 4, 12, 11]). With digraph model, a keyword search engine captures a richer semantics than that based on tree model. The key concept in existing semantics can be viewed as *reduced subtrees* ([8, 13]). Given an XML graph G and a list of keywords K, a subgraph T of G is a reduced tree with respect to K if T contains all keywords in K, but no proper subtree of T contains all these keywords. For example, "$1.1.1 \leftarrow 1.1 \leftarrow 1 \rightarrow 1.2 \rightarrow 1.2.1$" is a reduced subtree for query "Widom Lorel" in Figure 1. However, the problem of finding and enumerating results by increasing sizes of reduce-trees for keyword proximity is NP-hard; the well-known *Steiner tree* problem [7] for graph can be reduced to it (see reduction approach in [15]). Thus keyword proximity search based on digraph model are usually intrinsically expensive, heuristics-based and less efficient than that based

[1] In [16], the Smallest LCA is referred as Meaningful LCA.

VLDB '07, September 23-28, 2007, Vienna, Austria.

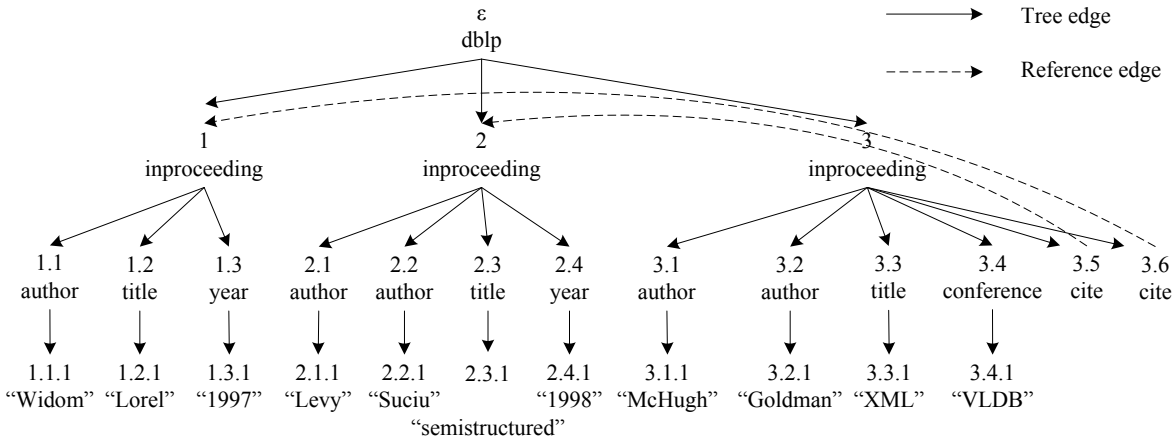


Figure 1: Example XML document (with Dewey numbers)

on tree data model.

Despite the pros and cons for tree or digraph model for keyword proximity search in XML, most of those existing efforts do not exploit schema information of the underlining XML data and application.Although schema-independent keyword proximity search implementation is designed to be general for all data instances (thus simplifying the job of developers from tuning the implementation to fit different application), implementations without schema have two limitations. First, schema-independent systems may have difficulties in representing results as also addressed in [11]. For example, in DBLP bibliography where authors are subelements of papers, given a keyword query of one author's names, the Smallest LCAs or reduced-trees would be a list of author elements of the same author. However, it is not quite useful to return such elements containing only the author's names since the user already know the name and it makes no sense to repeat the query itself as the results. Notice the navigation feature of user interface does not solve the problem as users will not know from which author name to start navigation. This problem gets even worse for queries of only one keyword (the result of Smallest LCA or reduced trees would be a list of such keywords). Thus it is important for the system to present enough information in each single result rather than the smallest possible element (or sub-structure).

Second, schema-independent implementations may return many irrelevant results. For example, Smallest LCA semantics based on tree model may return a large tree including many irrelevant results. Let us consider the keyword query "Suciu XML" in DBLP data, which most likely looks for XML papers written by Suciu. Suppose there was no Suciu's paper with "XML" in the title, then the LCA of the two keywords would be the root of whole bibliography tree which overwhelmingly includes all papers. [10] tries to solve this problem by introducing *MCTs (minimum connecting trees)* to exclude the subtrees rooted at the LCAs (or Smallest LCAs) that do not contain query keywords. For example, MCT for previous query "Suciu XML" would include all papers whose authors are named "Suciu" or titles include "XML" instead of the LCA result of whole bibliography tree. However, MCT semantics may still return many irrelevant results since neither all Suciu's papers are related to XML nor all XML papers are written by Suciu. Therefore, the LCA semantics and its variation, based on tree model may return many irrelevant results. Notice that the ranking methods proposed in XRANK ([9]) are not adequate to solve the problem of overwhelming information since they focus on ranking among LCAs and cannot rank information inside a single overwhelming LCA. In digraph model, this problem can get even worse for reduced-tree semantics since the keyword proximity results of reduced-trees form a superset of MCT results. BANKS ([12]) ranks results in approximate order of result generation by heuristic bi-directional expansion from "important" nodes via "important" edges in digraphs. Given a good estimation of "important" nodes and edges, BANKS seems to work well for a small number of results. However, it still has the inherited limitation of slow query response when a large number of results are required.

We believe schema information is usually present in many XML databases regardless their semi-structured nature. For example, most of the well studied databases in the literature have schema information, such as DBLP bibliography and XMark auction XML databases. When schema information is available, it can be used for better keyword search as the focus of this paper; or when schema information is not available, the previous Smallest LCA or reduced tree semantics can be adopted.

In this paper, we study a novel data model, *interconnected object trees*, to exploit the property and schema information of XML databases for better keyword proximity search when schema information is available. Based on this model, we propose a framework of keyword proximity search system with novel semantics for ranked result evaluation.

In particular, the database administrator or domain expert first exploits the schema information of XML documents to define a set of interested objects and conceptual connections between interested objects. The interested objects are modeled as rooted trees containing enough but not overwhelming information, which can be either physical or conceptual subtrees in XML database. The conceptual connection between two interested objects can be (but not restricted to) ID references between the objects indicating their relevance relationships. For example, in the auction XML database, one interested object can be physical subtrees rooted at "person" and two persons can have a con-

nection if they share common interests (e.g. they attended many auctions together); whereas in the DBLP XML database, the interested objects can be publications and two publications have a connection if one cites another. Note that end users need not know the schema except that they may choose an interested object if there are multiple such objects. In case users do not know their interested object, the system can choose a default one for them or output results categorized into different interested objects.

With the concept of interested objects, the problem of overwhelming Smallest LCAs is automatically solved since results are restricted to the interested objects which contain all the keywords. We call such interested objects containing all query keywords as *ICAs (Interested Common Ancestors)* of the keywords in contrast to Smallest LCAs. An immediate drawback of ICAs is that ICAs may miss relevant results since it is not necessary that a relevant result must contain all keywords. For example, consider the query "Suciu XML" in Figure 1 again, where the interested objects are defined as publications. There would be no results since no single publication contains the two words as an ICA. However, with the bibliography semantics, a domain expert would argue the second "inproceeding" may be relevant to the query since it is written by Suciu and related to XML as evidenced from the citation from an XML paper ("inproceeding" 3). Therefore, we propose novel IRA (Interested Related Ancestors) semantics to find relevant objects which do not contain all keywords. IRA captures conceptual connections which indicate relevance between objects in real world. Informally, each result of IRA of a list of keywords is a pair of conceptually connected interested objects such that each object in the pair contains some keyword and the pair together contain all keywords. Then, we use conceptual connections in IRA results to compute our novel ranking metric *RelevanceRank* to rank interested objects according to their scores of relevance to the query for output. Our RelevanceRank is different from existing ranking metrics such as PageRank [5], HITS [14] and ObjectRank [3] which also exploits connections between objects. We will explain the differences in Section 6. Notice that We consider IRA as logical results to discover more relevant interest objects, but we choose not to directly display IRA results to users. For clarity, the output is a list of ranked interested objects. However, a user can click on the objects to see its connections with other objects if s/he is interested in the connections.

The contributions of this paper are summarized as follows:

- We introduce the *interconnected object trees* data model to fully exploit XML property and schema information for ranked keyword search in XML databases. Based on this model, we propose *ICA* and novel *IRA* semantics to find relevant interested objects (with conceptual connections for IRA) which contain all or some query keywords. Then, we propose novel *RelevanceRank* to rank results based on their relevance to the keyword query according to the conceptual connections in IRA results. Also, a framework of ranking is discussed which combines RelevanceRank and other existing ranking metrics.
- We design and analyze an efficient algorithm to compute the ranked results according to our ICA and IRA semantics in one phase without keeping or materializing the connections in logical IRA pair results.
- We conduct extensive experiments to study our keyword search approach. Results show our approach considerably outperforms existing systems in terms of result quality and/or query response. A prototype of our system, named ICRA ($ICRA = ICA + IRA$), over updated 321M DBLP data is available at http://xmldb.ddns.comp.nus.edu.sg/.

In the rest of this paper, we first introduce our data model in Section 2. Then, we discuss ICA/IRA semantics and RelevanceRank to find and rank results of keyword queries based on our data model in Section 3. Section 4 presents the data structure and algorithms for our keyword search. Experimental results are given in Section 5. We also review related works in Section 6 before we finally conclude in Section 7.

2. DATA MODEL AND SEMANTICS

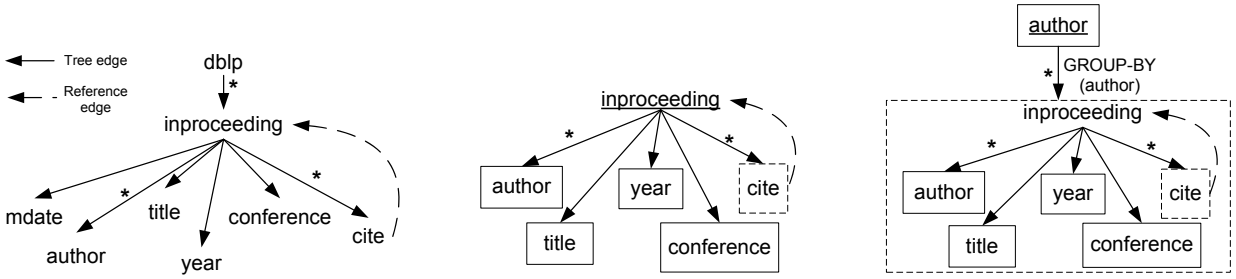
In this paper, we model an XML database D as interconnected object trees, $D=(R_D, T, C)$, to exploit XML property and schema semantics. In the model, R_D is the root of D; T is a set of physical or conceptual rooted object trees in D; and C is a set of conceptual connections between two object trees. Each rooted object tree $o \in T$ belongs to a certain interested object class which is defined according to the schema semantics by database administrators or domain experts. An object tree contains all necessary information for the object in tree pattern. In the following, we will use *object* and *object tree* interchangeably to refer to objects modeled as subtrees in XML database D. Object trees can be either physical subtrees of D or conceptual subtrees defined according to the schema semantics. We discuss possible specifications of interested objects as object trees in Section 2.1. Each conceptual connection $c_i \in C$ is a pair of object trees (o, o'), representing the conceptual relevance between o and o'. An overview of conceptual connections and how they can be defined and mined from XML database D is covered in Section 2.2.

2.1 Interested Object Tree

Each XML database may have multiple types of interested objects. For example, the types of interested objects may be (but not restricted to) *publication* and *author* in bibliography database; whereas in auction database, there may be three types of interested objects, *person*, *item* and *auction*. We model these interested objects as object trees which can be either physical or conceptual subtrees of XML database D. It is not a difficult task for administrators to specify of all types of interested objects according to the schema semantics. However, specification of the contents of each class of interested objects as object trees is a bit involved as it is equivalent to defining views in XML databases. In the following, we illustrate three cases of object tree specification for an interested object to give some insights of how this task can be performed. However, these illustrations are not meant to cover all cases of application requirements.

- **Object tree as physical subtree of D**

The object tree of interested objects can simply be physical subtrees or partial subtrees rooted at elements of certain tags. For example, given the schema graph of bibliography



(a) Bibliography schema graph (b) Object contents including subtrees (c) Object contents with group-by

Figure 2: Bibliography schema graph and object content specification examples

XML database in Figure 2(a), the object trees of *publication* objects are simply subtrees rooted at "inproceeding"[2] elements (shown in Figure 2(b)), where the root of the object tree is underlined. However, it is not necessary that all descendants of the "inproceeding" subtree are included in the object. For example, the "mdate" which is a system attribute indicating the last modified date is not considered as *publication* object's information. By default, we assume an element's descendants in database D must be explicitly included in the object tree. However, we can also use solid rectangles to indicate the whole subtree rooted at an element is indeed included in the object tree, e.g. "author" in Figure 2(b). Similarly, not all information of an object tree is considered equal for output. For example, we may not want to display all citations of each *publication* object to mess up the output. Thus, as shown in Figure 2(b), we can use dashed rectangle to indicate the information of "cite" is only displayed at users' request (e.g. by clicking the citations in result display).

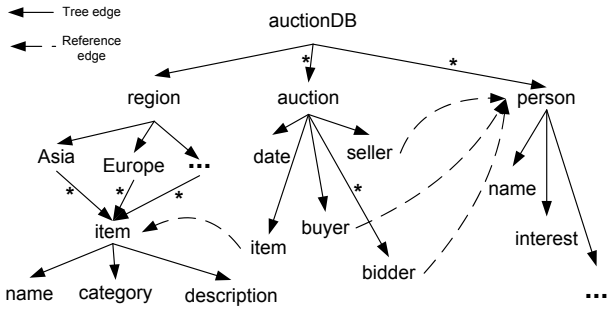
- **Object tree including ancestors of certain tags**

Object trees of interested objects can also include the ancestors of certain tags in XML database D as object information. For example, consider Figure 3(a), where "item"s are grouped according to "region"s and names of "region"s (e.g. Asia, Europe, etc.) are designed to be element tags. In this case, when we specify the contents of *item* object tree, we can also include the ancestor information of "item" into the object tree as shown in Figure 3(b), where "_anscestor_" indicates we do not care the tag name of the ancestor.

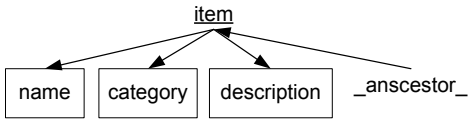
- **Object tree with group-by information**

We can also compute group-by information for object trees whose information is not properly nested as their descendants in the schema graph. For example, in DBLP XML database, each "author" element contains only the information of author name, which is usually not enough for *author* object tree. Thus as shown in Figure 2(c), we can group *publication* information for each author so that when users select *author* as the interested object and issue a query of an author's name, they can not only see the author's name, but also they can click to see all the author's publications. Note that DBLP database currently does not have ID for authors and two authors may have the same name. This raises the difficulty of grouping by author (grouping by the person rather than the name), but how to solve this problem is beyond the scope of this work.

[2]We use italics for object names (e.g. *publication*); whereas element tags in XML are indicated by quotations (e.g. "inproceeding")



(a) Auction DB schema graph



(b) Object contents including subtrees and ancestors

Figure 3: Auction DB schema graph and object contents specification examples

Finally, we address that we model interested objects as subtree trees in order to benefit from the simplicity and efficiency of LCA semantics of tree model for keyword proximity search. However, object trees are not restricted to physical subtrees in XML databases. For example, the last two cases of object trees discussed above are actually conceptual subtrees in XML databases. With the notion of conceptual object trees, our model offers much more flexibility and generality than original physical tree model.

2.2 Conceptual Connection

Conceptual connections in XML databases indicate the conceptual relevance between objects. Remind our model D=(R_D, T, C). Each conceptual connection $c_i \in C$ is a pair of object trees (or objects) (o, o'). Depending on the application semantics, connection can be directed, e.g. citations in bibliography, or undirected, e.g. common interest between two persons in auction DB. Conceptual connections can also be defined between two different types of objects. However, we only consider conceptual connections between objects of the same type in this paper and leave conceptual connec-

tions between objects of different types as a future work. Similar to interested objects, the specification of conceptual connections is application and schema dependent. In the following, we show some cases of conceptual connections defined or mined from physical database D, which is not meant to cover all cases.

- **ID references**

ID references in XML databases is a natural way to specify conceptual connections in most applications. For example, the citation relationship between two *publication* objects is usually represented by ID reference as in Figure 2(a).

- **Common ancestor or descendant**

In some cases, two objects may have a conceptual connection if they share the same ancestor or descendant in physical XML database D. For example, two *author* objects can be considered related (co-author) if they share the same "inproceeding" (publication) ancestor in D. Similarly, two *publication* objects may be related if they have the same "author" descendant; or in auction DB, two *person*s can be business-related if they have matching "business" descendants. Notice that it is not easy to capture connections through common descendants without schema semantics.

- **Conceptual connection by intermediate object**

Two objects may also be conceptually connected through intermediate objects. For example, two persons may share common interest if they attended the same auction; two papers may be related if they cite or are cited by the same paper. There may be even a chain of intermediate objects where the administrator can set the length of the chain to a meaningful limit. Notice that administrators can also set a threshold of the minimal number of intermediate objects required for conceptual connections. For example, we may specify that two persons share common interests only if they attended at least three auctions together.

Similar to object trees, conceptual connections provides more flexibility and generality from administrator's definition according to the schema, in contrast to most existing works only based on physical connections. Moreover, we clearly distinguish tree edges within an object and conceptual connections between objects. In this way, we effectively reduce the search space as compared to keyword proximity search in general graph model since keyword proximity search based on tree model (object tree edges) is usually simple and efficient; meanwhile we captures the connections between objects that tree model cannot capture. We will discuss how the objects and conceptual connections can be used for keyword search in next Section.

3. ICA/IRA SEMANTICS AND RANKING

In this section, we first define the result semantics of a keyword query based on interconnected object trees model. Then, we present the novel RelevanceRank and how it can be incorporated in a general ranking model.

3.1 ICA/IRA Semantics

Given the notion of interested object in our model, the most intuitive result of a keyword query is a list of interested object trees that each contains all query keywords (conjunction of keywords). We call these interested objects as *ICAs (Interested Common Ancestors)* in contrast to well-known LCA (Lowest Common Ancestor) semantics in tree model. Notice we use "Ancestor" instead of "Object" in ICA to emphasize the fact that interested objects are modeled as (physical or conceptual) trees in XML databases to benefit from simplicity and efficiency of tree model.

DEFINITION 3.1. ***(ICA)*** *Given a query Q of a list of keywords, the* Interested Common Ancestors (ICAs) *of Q, denoted as ICA(Q), is a list L of interested objects such that each $o_i \in L$ contains all query keywords.*

ICA semantics can be viewed as an extension from LCA semantics. However, as discussed, ICA offers more flexibility based on the interested object concept whose conceptual subtree may include ancestors or group-by information in the physical XML databases.

One obvious limitation of ICA semantics is it is too restrictive. An object which does not contain all query keywords can also be related to the concept of the keyword query. However, those objects are excluded from ICA results. For example, given a keyword query "XML query processing", a paper with title "XML query evaluation" is usually highly related to the query, but ICA semantics does not consider it as a result. A first and natural attempt to eliminate the limitation is to include interested objects containing some keywords (disjunction of keywords) and use traditional TF*IDF cosine similarity to rank this objects. However, keyword disjunction with TF*IDF cosine similarity sometimes cannot enforce the precise concept expressed as the conjunction of keywords. For example, there are many relational query processing papers that are not really related to "XML query processing". However, TF*IDF cosine similarity has a good chance to give high ranks to those relational "query processing" papers given the fact "XML" is also a very frequent word. Therefore, we propose novel IRA (Interested Related Ancestors) to include interested objects containing some keywords, and meanwhile enforce the precise concept of keyword conjunction in conceptually related objects.

DEFINITION 3.2. ***(IRA)*** *Given a query Q of a list of keywords, the* Interested Related Ancestors (IRA) *of Q, denoted as IRA(Q), is a list L of conceptually related interested object pairs such that each object of a pair contains some query keywords and the pair together contain all query keywords.*

We also say that an interested object is in IRA results or is an IRA object *for Q if it is a member of some IRA pair and it is not in ICA results for Q.*

Note that an ICA object o can also form an IRA pair with an object o' containing some query keywords to include o' as an IRA object. However, the ICA result o is not double counted as IRA object.

Let us see the intuition of IRA semantics with examples. Consider in auction database, a user looks for persons who are interested in paintings and sculptures by setting the interested object as *person* and issuing query "painting sculpture" (the user can enforce the query as "interest painting sculpture" if the tag name "interest" is also indexed in the system). Suppose there is a person, Alice, who only indicated one of her interest as "painting" during registration for

some reason (e.g. to save registration time or to protect privacy). However, if Alice has interest-connection (indicated by attending the same auction) to persons whose interests include "sculpture" but not "painting", it is reasonable to consider Alice as a candidate result by IRA semantics. At least, the IRA result, Alice, is a better result than other persons whose interests include only "painting" and are not interest-connected to persons whose interests include "sculpture". Similarly, for query "XML query processing" in bibliography database, a paper that has "query processing" in the title and cites or is cited by "XML" papers is considered related to the query by IRA semantics. Note in the following discussions, we do not consider objects that contain some keywords but are not IRA objects in our search result although those objects can also be included and ranked lower than IRA objects.

Note that we currently restrict our discussions to conceptually connected *object pairs* in IRA semantics instead of the more general *object groups* for three reasons. First, a large portion of keyword queries consist of only a few keywords. Thus, a pair of objects are usually enough to contain all keywords. For example, we do not need three objects to together contain only two keywords. Second, in cases of queries with many keywords, the benefit of extending IRA pair to IRA group is not certain in terms of result quality. According to the general assumption in keyword proximity search that a smaller structure is usually a better and more precise result, we think the less the number of related objects that together contain all query keywords is, the more likely each object is still relevant to the precise concept of the query. In other words, if only a large number of conceptually connected objects can contain all query keywords, then it is doubtful that each object is still concisely relevant to the query. Third, unlike the simple pattern of conceptual connection in a pair, the pattern of connection in a group of objects can be very complicated, which can result in poor performance in terms of query processing time. Therefore, we leave the general IRA group semantics with possible long query response yet doubtful gain in result quality as a future work when there are suitable applications.

Also note that IRA semantics may not be meaningful in some cases. For example, given a query of a particular person's name, e.g. "Alice Bush", the possible IRA pair of two persons Alice and Bush that have common interests are not relevant to the query at all. However, this is generally not avoidable for systems that handles keyword disjunction and include objects containing only some keywords. We will explain shortly that, in our model, persons with name "Alice" or "Bush" (but not "Alice Bush") of the IRA pair will not be ranked higher than persons with name "Alice Bush".

3.2 Relevance Rank

IRA semantics is useful to find more interested objects that contain some of query keywords. However, each IRA result in the origin is a pair of conceptually connected objects, which may not be a good choice for direct result display due to two reasons. First, most of the time, users are interested in the objects rather than the relationships between objects. Second and more importantly, IRA pairs can have many duplicated objects. For example, if a "query processing" paper P is cited by many "XML" papers, then P will appear in many IRA pairs for query "XML query processing". Such duplication may frustrate the users. Thus, we present our ranking metric *RelevanceRank* to rank objects in IRA pairs according to their relevance to the query so that the output is a list of objects instead of object pairs. However, upon users' request, the system can provide the information of all IRA pairs for each object that does not contain all keywords. This information can serve as a reason of why and how the object is included and ranked in the results.

In IR (Information Retrieval) community, the traditional TF*IDF cosine similarity is good way to measure the similarity (or relevance) of a document to a keyword query. The formula of the cosine similarity measure can be expressed as the following:

$$\rho(q,d) = \frac{\sum_{t \in q \cap d} W_{q,t} * W_{d,t}}{W_q * W_d}$$

where t is a term or keyword; $W_{q,t}$ and $W_{d,t}$ represent the weights of t in query q and document d respectively; while W_q and W_d are the weights of query q and document d which are usually proportional to the length of q and d respectively. $W_{q,t}$ is normally inverse proportional to the number of documents containing t (Inverse Document Frequency or IDF) or $W_{q,t}$ can also be defined in user queries; whereas $W_{d,t}$ is normally proportional to the number of occurrences of t in d (Term Frequency or TF). This cosine similarity models both d and q as vectors in n-dimension space where each term represents one dimension; and $\rho(q,d)$ is the cosine value of the angle between the vectors of d and q. Thus, a larger cosine value $\rho(q,d)$ indicates the vectors of d and q have similar orientation, and consequentially, d and q are more relevant. There is also existing work to refine the original TF*IDF to explore the structure of XML for XML specific cosine similarity (e.g. [6]). However, we use the original TF*IDF cosine similarity in the following discussion for simplicity since [6] is orthogonal to our ranking based on conceptual connections and can be easily incorporated in our approach.

The TF*IDF cosine similarity or its XML specific variation [6] does not consider the conceptual connections in IRA semantics. Generally speaking, we can consider the conceptual connections in IRA semantics as additional bonus to an object's relevance to a given query besides the TF*IDF cosine similarity. We call this bonus as *IRA bonus*. Therefore, the relevance of an IRA object o to query q can be viewed as the following formula:

$$RelevanceRank(q,o) = \rho(q,o) + Bonus(q,o)$$

where $\rho(q,o)$ is the cosine similarity of o to q and $Bonus(q,o)$ is the IRA bonus of o from all its IRA pairs for q.

Therefore, our task is to find the IRA bonus $Bonus(q,o)$ of o from all o's IRA pairs for q. Logically, given an IRA object o containing only some keywords, the maximum possible IRA bonus o can receive from all its IRA pairs is the additional value to make o as relevant as if o contains all query keywords[3]. Thus we can model $Bonus(q,o)$ as a fraction of the maximum possible IRA bonus that o can receive. Note that our interest is in the relative relevance of an IRA object to a query as compared to other IRA objects (especially when they have similar cosine similarity) rather than the absolute IRA bonus value.

[3]In this paper, we enforce the policy that an object with only some query keywords should not be ranked higher than an object containing all query keywords. However, this policy can be relaxed by adjusting the upper bound of IRA bonus according to application semantics.

To get the maximum possible IRA bonus of object o to query q, imagine there is an object o' such that o' is the same as o except that o' also contains all the keywords that o does not contain for q. Then the difference between the cosine similarity of o' to q and o to q can be viewed as the maximum possible IRA bonus that o can receive from all its IRA pairs. Thus, we present the maximum possible IRA bonus of o given q as the following formula:

$$MaxB(q, o) = \rho(q, o \cup q) - \rho(q, o)$$

where $o \cup q$ can be viewed as the object o' discussed above. With the upper bound $MaxB(q, o)$, we can avoid the situation where two persons "Alice" and "Bush" with common interests is ranked higher than the person "Alice Bush" for query "Alice Bush" in general.

With the maximum possible IRA bonus of IRA object o to q, $MaxB(q, o)$, we now need to model the fraction of $MaxB(q, o)$ for o's IRA bonus, $Bonus(q, o)$, from all its IRA pairs. Intuitively, the more IRA pairs an object participates in, the more likely the object is relevant to the overall concept of the keyword query. We call the number of IRA pairs that an object o participates in for a query q as the *IRA participation count* of o for q. For example, Consider keyword query "interest painting sculpture" in auction DB. Given two persons, Alice and Bob, who have interest in "painting". Suppose Alice has conceptual connections (indicated by attending the same auction) to many persons who are interested in "sculpture" but not "painting"; while Bob has connections to only a few such persons. It is reasonable to guess Alice is more likely to be interested in "sculpture" thus more relevant to the query than Bob. Similarly, for keyword query "XML query processing" in bibliography, if a "query processing" paper cites or is cited by many "XML" papers, it is likely this paper is more relevant to the keyword query as compared to another "query processing" paper citing or cited by only one or two "XML" papers. Therefore, we can use the IRA participation count of o for q normalized by the total number of IRA pairs of q as possible fraction of $MaxB(q, o)$ for $Bonus(q, o)$.

However, the IRA participation count does not distinguish the bonus from different conceptually related objects. For example, for keyword query "XML query processing" in bibliography, the IRA bonus to a "query processing" paper from a conceptually related "XML" paper may be different from the bonus from a conceptually related "XML query processing" paper which is more relevant to the query. In other words, a "query processing" paper citing or cited by an "XML query processing" paper may be more relevant to the query "XML query processing" than another "query processing" paper citing or cited by an "XML" paper. Similarly, for query "interest painting sculpture" in auction database, if both Alice and David are interested in "painting", but Alice attended same auctions with people who are interested in "sculpture" but not "painting" whereas David attended same auctions with people who are interested in both "sculpture" and "painting", then it is likely that Alice is more relevant to query than David. So, we present the following IRA bonus formula to to distinguish the IRA bonus for an object from its different IRA pairs:

$$Bonus(q, o) = \frac{\sum_{\forall o' | (o,o') \in IRA(q)} BF(q, o, o')}{NFactor} * MaxB(q, o)$$

where $Bonus(q, o)$ is the IRA bonus of o from all its IRA pairs of q; $BF(q, o, o')$ is an application specific bonus function for the IRA bonus of o from a particular IRA pair o'; and $NFactor$ is the normalization factor which can be the total number of IRA pairs of q. In bibliography, the function $BF(q, o, o')$ can be direct proportional to $\rho(q, o')$ based on the intuition that if paper P_1 cites (or is cited by) paper P_1' such that P_1' is closely related to a given query q whereas another paper P_2 cites (or is cited by) paper P_2' such that P_2' is not as closely related to query q as P_1', then it is likedly P_1 is more related to q than P_2. However, in auction database with keyword queries matching person's interest, $BF(q, o, o')$ may not always be proportional to $\rho(q, o')$ as discussed for the previous case that Alice is likely more relevant to the query than David although David's IRA pair is closely related to the query. Instead, $BF(q, o, o')$ should return zero or a very low value if o and o' have overlaps in their "interest" elements. So, $BF(q, o, o')$ should be carefully designed according to application semantics.

Note that using the number of IRA pair of q as the normalization factor may be biased to objects which have many conceptual connected objects. For example, in bibliography, an old famous paper can be cited by many query relevant papers even the old paper may not be very relevant to the query. An alternative is to use the number of conceptually connected objects of object o as the normalization factor for $Bonus(q, o)$. However, this choice may discriminate against objects that have many conceptually related objects. For example, if object o_1 has only one conceptual related object and this pair is an IRA of q whereas another object o_2 has 100 related objects and 99 out of them form IRA pairs with o_2 for q, then o_2 is still likely to be ranked lower than o_1. Other choices are also possible, but it is difficult to judge which one is superior in all cases. In this paper, we simply adopt the first choice for DBLP data to favor highly cited papers and the experiments show it is a reasonably good choice for DBLP data.

All the above discussions focus on the IRA bonus of only one type of conceptual connection. However, there may be several different conceptual connections between the same objects. For example, in bibliography, besides direct reference and citation, two papers can also be considered relevant if they are cited or they cite the same paper. Similarly, persons in auction database may also be conceptual related if their addresses are near to each other (geographical connections may be useful when someone wants to find all persons that are interested in painting and stay in or nearby the city of an upcoming painting auction so that those persons may be potential bidders). In general, when we consider several conceptual connections, the IRA bonus of different connections may be given different weights. Therefore, we now take the final step to introduce our RelevanceRank formula which incorporates TF*IDF cosine similarity and IRA bonus of multiple conceptual connection for an object o and a given query q as following:

$$RelevanceRank(q, o) = \rho(q, o) + \frac{\sum_{i=1}^{k} w_i * Bonus_i(q, o)}{\sum_{i=1}^{k} w_i}$$

where k is the number of different conceptual connections, $Bonus_i(q, o)$ is the IRA bonus for the i^{th} connection and w_i is the weight given to the i^{th} connection. Notice that w_i can be set by the administrator or dynamically adjusted by the system or chosen by experienced users at run time. For example, in bibliography database, the administrator may set

the weight of IRA bonus for direct citations/references as 1 and the weight of IRA bonus for indirect citations (through a hub) as (say) 0.5; whereas in auction database, the system can dynamically give more weight to IRA bonus based on geographical connections if most query keywords match text values in "address" rather than "interest" element of *person* object and vice versa. Note in some cases, system can even automatically choose not to use IRA bonus based on a particular conceptual connection for RelevanceRank if that conceptual connection does not apply for a given query. For example, given query "Alice Bush" which does not match text values in "interest" elements, then the system can choose not to use interest-related connections for RelevanceRank.

3.3 Overall Ranking Framework

RelevanceRank ranks an object based on its relevance to a given query according to the traditional TF*IDF cosine similarity or its XML specific variant and additional relevance (IRA bonus) discovered from all its IRA pairs. In this part, we briefly list some other existing possible ranking metrics for keyword search in XML databases and discuss how our RelevanceRank can be incorporated with these metrics to compute an overall ranking for each objects according to both XML properties and application semantics.

3.3.1 Keyword Proximity Rank: $ProxRank(q, o)$

The general assumption of keyword proximity search in XML tree or digraph model that prefers smaller substructures can also be addressed in our interconnected object tree model to rank objects with closer co-occurrence of keywords higher. Since we model interested objects as trees, most tree model based existing keyword proximity ranking methods (e.g. XRANK [9]) can be directly used except special considerations may be needed for IRA objects. In this work, we adopt an easy way to estimate the keyword proximity by counting the minimum number of elements[4] inside an object tree that directly contain query keywords in their text values. Smaller number of elements indicates closer co-occurrence of keywords, thus a better result. For example, an *publication* object with one author element containing two query keywords (names of a single author) is more precise (better) than an *publication* with two authors containing the two keywords (co-author). For IRA results, where not all objects contain the same number of query keywords, we can simply estimate the proximity by per keyword minimum number of elements that contain all keywords in the object.

3.3.2 Application Specific Rank: $ApRank(o)$

We can also incorporate application specific ranking metrics in the overall rank framework to reflect the importance of an object in the application. For example, in bibliography database, a recent paper may be considered "important" to reflect the general fact that most publications get out-of-date quickly and most researcher are interested in new information. The PageRank like metrics can also be a candidate in bibliography to favor highly cited papers. However, PageRank may not be useful to rank *person* objects in auction database. In this case, other application specific metrics may be used, e.g. salary or wealth, etc. Notice that we current model application specific rank as a query independent metric. However, it can be extended to query-dependent metrics according to application requirements.

In summary, we show an overall ranking framework, $R(q, o)$, for keyword search in interconnect object trees model to incorporate keyword proximity, application dependent metrics and novel RelevanceRank as the following formula.

$$R(q, o) = RelevanceRank(q, o) * ProxRank(q, o) * ApRank(o)$$

4. DATA STRUCTURE AND ALGORITHM

4.1 Data Structure

4.1.1 Inverted Lists

Similar to most keyword search systems, the inverted list is the fundamental structure in our system. Since there may be several interested object classes, each keyword can also has several inverted lists, one for each object class. An object is in the corresponding inverted list of a keyword if the conceptual object tree contains the keyword in the subtree. Without loss of generality, we assume there is only one object class and one inverted list for each keyword in the following since keyword query processing of multiple interested object class can be performed independently.

Each element in the inverted list is a triplet (Oid, DL, Imp), where Oid is the ID of the object containing the keyword; DL is a list of Dewey numbers of the exact locations of the keyword in the object for keyword proximity and Imp is the application dependent "importance" of the object. Objects in inverted lists can be ordered according to Imp so that the query processing can start from those "important" objects and scan only prefixes of the inverted lists for the top few results. Finally, a B+ tree index is built on top of each inverted list to facilitate fast probing of an object in the list.

4.1.2 Conceptual Connection Tables

Conceptual connection tables are used to store the conceptual relationships between objects. Since there may be different conceptual connections for the same object class, there may be also several conceptual connection tables for the same object class. Each conceptual table can be modeled as an adjacency list representation of the conceptual connection graph. In other words, for each object, the connection table maintain a list of its conceptually connected objects.

In the worst case, the size of each connection table for object class c can be $|c|^2$, where $|c|$ is the number of objects of class c. However, we argue that in most practical situations, the size is significantly smaller than the worst case upper bound. For example, in bibliography, each paper usually cites only a few papers (in the order of tens), which is almost a negligible portion to the total number of papers (in the order of millions). For connections through intermediate nodes, such as common interests indicated by attending the same auction, there is a potential for the connection table to grow huge. Even in such cases, the administrator can effectively reduce the table size by setting thresholds of minimum number of intermediate nodes as discussed in Section 2.2. Or when storage is not expensive, the system can store the huge table in the way such that each connection list is in the same order as inverted lists. Then during query process-

[4]We focus on keyword proximity at XML element level; while the keyword proximity within an element can be solved with traditional document keyword proximity techniques.

ing, the system can scan only a prefix of the connection list of each object for fast retrieval of top results

4.2 Algorithm

In this part, we design and analyze an efficient algorithm for keyword search for ICA and IRA results based on interconnected object trees model.

There are many existing algorithms in the literature to efficiently compute LCAs or Smallest LCAs results based on inverted lists with Dewey numbers, such as *Stack* algorithm [16], *Indexed Lookup* and *Scan Eager* algorithms [21]. These algorithms can be directly modified to compute our ICA results. Since [21] analyzes and reports *Indexed Lookup* is superior when one keyword inverted list is much shorter than others and *Indexed Lookup* is comparable to other algorithms in general, we adopt an modified version of *Indexed Lookup* algorithm for ICA. Indeed, we will discuss shortly that we also modify *Indexed Lookup* for our IRA semantics since objects' conceptual connected object lists are usually much shorter than keyword inverted lists. Notice that in LCA problem, the algorithms have to check the longest common prefix of the Dewey numbers in each keyword inverted list. However, for ICA results, we only need to check the object ID's to see if there exists an interested object to contain all keywords; and we only need to use Dewey numbers inside an object o for keyword proximity rank when o is indeed an ICA result. In this way, ICA computation can effectively save expensive Dewey number comparisons by using integer (object ID) comparisons.

Next, we discuss an efficient algorithm to compute and rank results based on IRA semantics and RelevanceRank. A brute force approach would first compute all IRA pairs. Then, it ranks each object o based on RelevanceRank by inspiring the keywords contained in o and all IRA pairs of o. However, this approach may generate large intermediate results since each object can be duplicated in many IRA pairs. Also, it requires a second phase to compute the RelevanceRank after IRA pairs are computed. With the following observations of the properties of IRA semantics and RelevanceRank, a better algorithm is possible.

Observation 1. Given a keyword query q of n keywords $(k_1, ..., k_n)$, for any query keyword k_i, an conceptually connected object pair (o, o') is one IRA pair of q if at least one of o or o' is in the inverted list of k_i (contains k_i).

Proof Sketch: (Proof by contradication) If both o and o' are not in the inversted list of one query keyword k_i, then o and o' cannot together contain all query keywords, which implies o and o' cannot be an IRA pair. Thus if o and o' form an IRA pair at least one of o or o' is in the inverted list of k_i (contains k_i).

Observation 2. Given an IRA pair (o, o') of IRA object o, the IRA bonus (Section 3.2) from o' to o is independent from other IRA pairs of o in current RelevanceRank metric.

Proof Sketch: From the formula of RelevanceRank, it is clear that we do not need other IRA pairs of o to compute the IRA bonus of o for a particular IRA pair o'. Note that the normalization factor may depend on the number of IRA pairs that o participates in. However, we can increase both o's participation count and IRA bonus without normalization independent of each particular IRA pair of o; then normalize IRA bonus of o after we find all IRA pairs of a given query or right before the algorithm (discussed shortly) decides to terminate.

```
   Input: Keywords: K_q; Indexed inverted lists for K_q: L_q;
          Conceptual connection tables: CTs;
   Output: Ranked result object list: RL
 1 let ICA_Result = {} and IRA_Result = {} ;
 2 let H be a hash table from objects to their ranks;
 3 let L_s be the shortest inverted list in L_q ;
 4 for each object ID o ∈ L_s do
 5     let K_o = probeLists (L_q, o) ;
 6     if (K_o == K_q)      /* o is in ICA                    */
 7         initializeRelevanceRank(o,K_o,K_q,H);
 8         ICA_Result = ICA_Result ∪ {o} ;
 9     for each connection table t_i ∈ CTs do
10         let c be the connected object list of o in t_i ;
11         for each object ID o' ∈ c do
12             let K_o' = probeLists (L_q, o') ;
13             if (K_o' == K_q)      /* o' is in ICA        */
14                 initializeRelevanceRank(o',K_o,K_q,H);
15                 ICA_Result = ICA_Result ∪ {o'} ;
16             else if (K_o' ≠ ∅ AND K_o' ∪ K_o == K_q)
               /* o' is in IRA                               */
17                 initializeRelevanceRank(o',K_o,K_q,H);
18                 IRA_result = IRA_result ∪ o';
19                 updateRelevanceRank(o', o, t_i, H);
20             if (K_o ≠ K_q AND K_o' ∪ K_o == K_q)
               /* o is in IRA                                */
21                 initializeRelevanceRank(o,K_o,K_q,H);
22                 IRA_result = IRA_result ∪ o ;
23                 updateRelevanceRank(o, o', t_i, H);
         end
     end
   end
27 let RL = ICA_result ∪ IRA_result ;
28 compute the overall rank for each object in RL and sort
   RL based on ranks for output ;

Function probeLists (L_q, o)
 /* returns all query keywords contained in o              */
 1  let KS = {} ;
 2  for each L_i ∈ L_q do
 3      put L_i's corresponding keyword in KS if o ∈ L_i ;
 4  return KS ;

Function initializeRelevanceRank (o, K_o, K_q, H)
 1 if (o is not in H)
 2     initialize the RelevanceRank of o as cosine
       similarity and put (o, RelevanceRank) into H ;

Function updateRelevanceRank (o, o', t_i, H)
 1 update the RelevanceRank of o based on keyword
   similarity of o' and the connection type of t_i ;
 2 put the updated (o, RelevanceRank) into H ;
```

Algorithm 1: Indexed-Probing

Based on Observation 1, it is sufficient to scan all object ID o in one keyword's inverted list and probe the other keyword's inverted lists with o and o' conceptually connected objects to find all IRA pairs. With Observation 2, when we find an IRA pair (o, o'), we only need to keep each IRA object o, o' and their IRA bonus independently. Then when we find another IRA pair for o (or o'), we can simply incremental update the IRA bonus for o (or o') accordingly. Therefore, we propose a one phase algorithm, *Indexed-Probing*, to compute all IRA objects and their RelevanceRanks without keeping intermediate IRA pairs. Notice an additional phase of sorting results by their ranks is still required. But this sorting is common in almost all ranking algorithms.

The pseudo-code of the *Indexed-Probing* algorithm is shown in Algorithm 1. The input to the algorithm includes a set of query keywords K_q, the B+-Tree indexed inverted lists L_q of K_q and a set of conceptual connection tables CTs for the interested object. The output is a list of ranked object IDs including both ICA and IRA results. The main idea of the algorithm is to scan the shortest inverted list of query keywords and check the objects in the list and their conceptually connected objects for ICA and IRA results. Notice that for the minimum cost purpose, we should scan the inverted list L such that the sum of number of objects o in L and number of all conceptually connected objects of o is minimum. However, when such information is not available, we can scan the shortest inverted list as a reasonable estimate.

The details of the algorithm are the followings. For each object ID o in the shortest inverted list, it probes all the other inverted lists to get the set of query keywords contained in o (line 5). With this information, we can decide whether o belongs to ICA results and initialize the RelevanceRank of o as consine similarity if it is not already initialized (lines 6-8). Then for each conceptually connected objects o' of o, the algorithm probes all inverted lists with o' to see if o' alone and/or o and o' together contain all query keywords to check for the following cases: 1) o' belongs to ICA results (lines 13-15); 2) o' belongs to IRA results (lines 16-19); 3) o belongs to IRA results (lines 20-23). In any case, the algorithm initializes and/or updates the the RelevanceRank of o' or o. Finally, after scanning the shorted inverted list, the algorithm combines the ICA and IRA results and sorts them for output (lines 28-29). The Function probeLists(L_q, o) probes all inverted lists in L_q with o to find the query keywords contained in o. Since each inverted list is indexed by a B+-tree, this process usually does not cost much especially when the indexed inverted lists of query keywords fit into memory. The Functions initializeRelevanceRank and updateRelevanceRank initialize or update the RelevanceRank of an object according to Section 3.2. Notice initializeRelevanceRank simply does nothing if the input object's rank is already initialized.

The CPU complexity of Indexed-Probing algorithm without sorting results for output is $O(\sum_{o \in L_s} \sum_{i=1}^{|CTs|} |t_i(o)| * \sum_{j=1}^{k} \log |L_j|)$, where L_s, o, $|CTs|$, $|t_i(o)|$, k and $|L_j|$ represent the shortest inverted list of the query keywords, an object ID in L_s, the number of different conceptual connection for the interested object, the length of the conceptual connection list for object o in its i's connection table, the number of query keywords and the length of the j's query keyword inverted list. For an intuition of the complexity, the first part, $\sum_{o \in L_s} \sum_{i=1}^{|CTs|} |t_i(o)|$, represents the total number of objects in L_s and all their conceptually connected objects in Big-O notation; whereas the second part, $\sum_{j=1}^{k} \log |L_j|$ represents the cost of inverted list probing for each object in the first part. However, we can add the complexity in proximity ranking and sorting results for output which is $O(kd|RS| + |RS|^2)$, to above formula if reader is interested in the overall complexity, where d is the length of dewey numbers for the interested object and $|RS|$ is the result size.

For the number of disk accesses, if the inverted lists of query keywords fit into memory, the worst-case is to load all these lists into memory. However, when the inverted lists of query keywords do not fit in memory for very huge XML databases, list probing for objects in the connection tables may incur slow random disk accesses. In such case, the algorithm can be easily modified to probe only the prefixes of inverted lists that fit in memory for top results. Given tens or hundreds of MBs of object IDs in the prefixes of inverted lists in memory, we believe the number of top results are usually large enough for interactive users to consume. Moreover, in case the computation of IRA is slow, the algorithm can also be modified to compute all ICA results first for fast response of top ranked results.

In the following, we state the correctness of Indexed-Probing algorithm with Theorem 1.

THEOREM 1. *The Indexed-Probing algorithm correctly finds and ranks all query answers based on ICA/IRA semantics and RelevanceRank.*

Proof Sketch: With Observation 1, the algorithm correctly finds all ICA and IRA results by probing all query keyword inverted lists with all objects o in the shortest inverted lists and the conceptual connected objects o' of o. With Observation 2, the algorithm correctly initializes and updates the RelevanceRanks for each ICA and IRA results (independent of other IRA pairs).

5. EXPERIMENTAL EVALUATION

5.1 Experimental Settings

5.1.1 Hardware and implementation

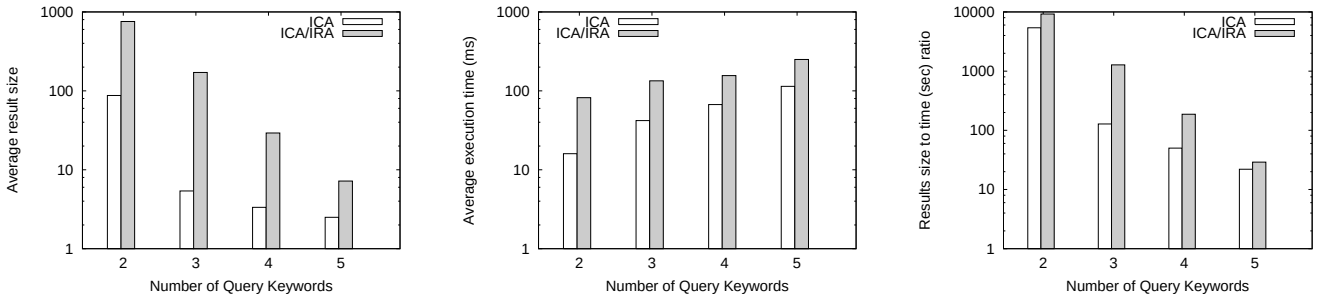
We use a normal PC with Pentium 2.6GHz CPU and 1GB memory for our experiment.

We choose DBLP datasets for our experiment, which has been widely studied for XML keyword search (e.g. [12, 21, 3]). The reality of DBLP data makes it possible to study the quality of search results besides search efficiency. Currently, we only implemented one object class, *publication*. We define two types of conceptual connections between two *publication* objects in DBLP, which are strong connection (direct reference/citation) and weak connection (e.g. cited by or citing the same *publication* object). The size of strong and weak connection tables are 3.9M and 19.1M respectively. They are stored in Disk with B+-trees and their entries are cached in memory only after the corresponding entries are used in searching.

All implementations are written in java. An online demo, which enables keyword search in the same DBLP data, is provided at http://xmldb.ddns.comp.nus.edu.sg. The prototype runs as a Java Servlet using Jakarta Tomcat server.

5.1.2 Queries and performance measures

We use both random and sample queries in our experiments. We generate sample queries of 2 to 5 keywords long, which are categorized into four sets such that there are 100 queries of same length in each set. Query keywords are random selected from all indexed keywords in the data, except that selected queries must have at least one ICA result. This is to ensure the random queries are not too "random" from reality since most users expect some results by issuing meaningful queries instead of random sequence of keywords. We use these random queries to measure the effectiveness of ICA/IRA in terms of result size and execution time tradeoff as compared to ICA alone.



(a) Result size (b) Execution time (c) Size to time ratio

Figure 4: Result size and execution time tradeoff of ICA v.s. ICA/IRA (log scale) for random queries

Table 1: Tested queries

ID	Query	Meaning
Q1	Giora Fernández	Co-author
Q2	Jim Gray transaction	Topic by author
Q3	Dan Suciu semistructured	Topic by author
Q4	Conceptual design relational database	Topic
Q5	Join optimization parallel distributed environment	Topic

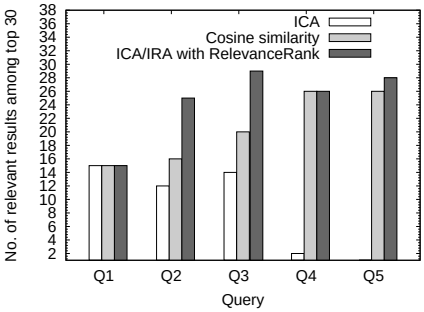


Figure 5: Result Quality of ICA v.s. ICA/IRA

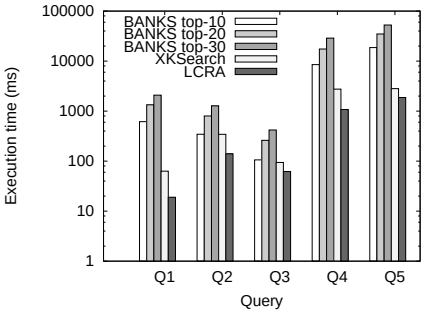


Figure 6: Performance of three systems

We also use sample queries of length 2-5 with wide range of meanings (as shown in Table 1) to measure the effectiveness of our ICA/IRA semantics and RelevanceRank in term of result quality. Note that we use both "Fernández" and "Fernandez" in other systems to try to get rid of the character encoding problem for Q1. Besides comparing ICA and ICA/IRA combined with RelvanceRank, we also report the comparisons of our system with other five existing systems, including three academic systems and two commercial systems. Our metric for result quality is the number of relevant answers among top-10, 20 and 30 results (precision of top-k results). Answer relevance of the queries are judged manually by reading the title, abstract, introduction and/or conclusion of the results (it is beyond our resource to read whole papers of all results for perfect judgement).

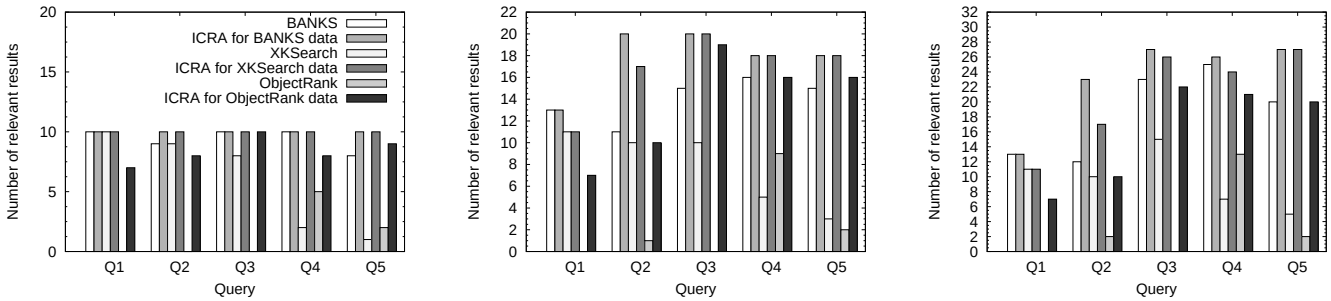
5.2 Results of Random Queries

We show result size, execution time and ratio of result size per unit time of ICA alone and ICA/IRA in Figure 4. As expected, the result size decreases and execution time increases as the number of keywords increases. Also, it is clear that ICA/IRA (IRA on top of ICA) semantics can find much more results (up to 30 times more results for queries of 3 keywords in Figure 4(a)) than ICA alone at the cost of additional execution time (2-3 times more in Figure 4(b)). Moreover, in Figure 4(c), we can see that ICA/IRA is more efficient than ICA alone in terms of the ratio of result size to execution time (up to 10 times more results per unit time). This is because both ICA and ICA/IRA need to load an inverted list block when they probe that block for the first time (the block is then cached in memory) and the total amount of loading is similar in two cases. Note that the computation of IRA needs to load the conceptually connected objects from conceptual connection tables for each object in the shortest inverted list. But there is a better chance for those entries to be cached since in both real search engines and our experiment, several consecutive queries may have completely different keywords (especially from different users for real cases) while the inverted lists of those keywords may have large overlaps.

5.3 Results of Sample Queries

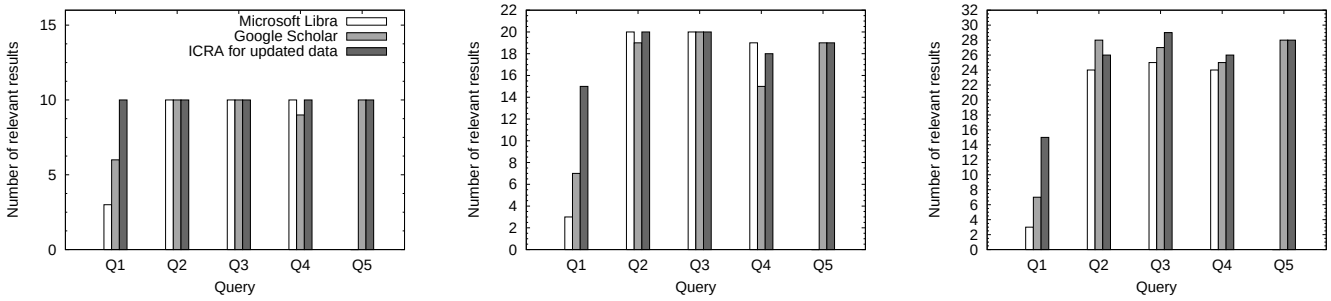
5.3.1 Result quality of ICA/IRA with RelevanceRank

We have shown that ICA/IRA can identify more results than ICA with reasonable more time for random queries. Now, in Figure 5, we show with sample queries that most of the top-30 ranked results of ICA/IRA semantics are indeed relevant to the queries with our ranking (RelevanceRank) methods. We omit the execution time for these sample queries to save space since it is similar to random queries. It is clear ICA/IRA with RelevanceRank is able to find more relevant results than ICA alone and ranks them in most of



(a) top-10 answer (b) top-20 answer (c) top-30 answer

Figure 7: Comparisons of answer quality with other academic systems



(a) top-10 answer (b) top-20 answer (c) top-30 answer

Figure 8: Comparisons of answer quality with commercial systems

the top 30 results for Q2-Q5 where the IRA semantics applies. Note there is no ICA results for Q5, but there are 28 relevant results due to IRA and RelevanceRank. ICA/IRA with RelevanceRank is also better than Keyword disjunction with cosine similarity in general. Note that the precision of Q4 & Q5 based on cosine similarity is comparable to ICA/IRA with RelevanceRank for top 30 results. However, for top 40 results of Q5, ICA/IRA with RelevanceRank shows its advantage over cosine similarity as it identifies 37 relevant results compared to 29 by cosine similarity. It is also interesting to find out that RelevanceRank is slightly worse than cosine similarity for top 40 results of Q4 (34:35). The reason is there are many paper containing "conceptual database design" which is a disjunction of Q4, but most of these papers are indeed about relational databases. So, when keyword disjunction still carries the meaning as conjunction, RelevanceRank and cosine similarity are comparable. However, when keyword disjunctions breaks the concise meaning as conjunction (e.g. Q2 & Q3), RelevanceRank is much better than cosine similarity. In case that IRA semantics does not apply (Q1), ICA/IRA does not lose in terms of result quality. ICA/IRA also does not lose in terms of execution time for Q1 if the system chooses to compute ICA first for fast response of top results as discussed in Section 4.2.

5.3.2 *ICA/IRA v.s. other academic demos*

Now, we report the comparison among other academic demo systems for keyword search in DBLP (BANKS [4, 12], XKSearch [21], ObjectRank [3]) and our ICA/IRA incorporated with RelevanceRank. We choose these three systems for comparison since other known related systems were not available at the time of our experiments.

Figure 6 compares the query execution time of BANKS, XKSearch and our ICA/IRA system. We get the execution time directly from BANKS and XKSearch online demo, thus the network delay does not affect the comparison. We cannot provide the execution time of ObjectRank since it is not shown in their demo. It is clear BANKS is much slower than XKSearch and our system since it is a heuristic of NP-hard problem. For example, BANKS's computations for only top 10 results of all sample queries are still slower than XKSearch and our system to compute all results. It is also interesting to see that our system is faster than XKSearch for all queries. One possible reason is the Dewey number comparison in LCA semantics of XKSearch is slower than our integer comparison for Object IDs. Another reason is XKSearch also includes results that contain only some query keywords (keyword disjunction) besides Smallest LCAs. This process may also slow down their execution.

The comparisons for result quality are shown in Figure 7. Since four systems use different datasets, for fair comparison, we show the results of ICA/IRA with RelevanceRank based on other systems' data. For example, "ICRA for BANKS data" in Figure 7 means that we run our system on data used by BANKS. Note that BANKS outputs results in the format of reduced trees (containing publication IDs) instead of lists of publications; we assume there is a middleware to transfer BANKS results to publication lists. From Figure 7, we can see the result quality of our system is superior than existing academic demo in general. ObjectRank is good at ranking results for single keywords. However, its result quality drops significantly as the number of keywords goes beyond three (e.g. Q4 & Q5). ObjectRank cannot handle Q1-3 possibly because it does not well maintain in-

formation for author names. The same as ICA, the LCA semantics in XKSearch is too restrictive when it limits the results to publications; and its results for keyword disjunction is not very useful without ranking. Despite the slow response of BANKS, our results are considerably better due to RelevanceRank. For Q4, BANKS results are comparable to ours since they also captures ID references in XML.

5.3.3 ICA/IRA v.s. commercial systems

Finally, we show the comparisons of our system with excising commercial systems, Microsoft Libra [1] and Google Scholar [2]. We consider them as commercial systems since they are products of commercial companies. However, readers may regard them as non-commercial system at their choice. The possible significant difference in machine power among Libra, Scholar and ours makes it unfair to compare execution time. Also, our limited resource prohibits us from comparing the overall usefulness such as their wonderful interfaces, the ability to get pdf files etc. Thus we focus on comparison of the relevance of top-k results. Figure 8 shows our system is comparable to (if not better than) Libra and Scholar for all sample queries even they are able to search in significantly more web data as compared to our DBLP data. Our result is much better for for Q1 than Libra and Scholar. Libra outputs only three results for Q1 possibly due to the encoding problem; whereas Scholar's results include papers where the two authors do not appear as coauthors. For Q5 our result is comparable to Scholar's and much better than Libra's. Libra cannot find any results for Q5 possibly since they only consider results containing all keywords, whereas cosine similarity possibly helps Scholar to find relevant results in large amount of web data. Note that the large amount of web data is positive for Scholar for Q5, but negative for Q1 as noises. However, from the anecdotal evidence of sample queries, our system, especially IRA and RelevanceRank, is able to achieve the positive facts of large amount of web data with only 321M DBLP data; and meanwhile our system is not affected by the noises.

5.4 Experiment Summary

From the experimental results, we conclude that IRA semantics on top of ICA can identify significantly more results than ICA alone; and ICA/IRA semantics is also efficient to evaluate. Based on the results of sample queries, ICA/IRA with RelevanceRank is able to place most relevant results in top-k of all ICA/IRA results. Moreover, Our system based on ICA/IRA and RelevanceRank is superior to most existing academic demos in terms of both execution time and, more importantly, result quality. When compared to commercial systems, our system also demonstrates superior or comparable power in finding relevant results with limited 321M DBLP data.

6. RELATED WORK

Extensive research efforts have been conducted on efficient and effective processing of keyword search, especially keyword proximity search in XML databases. Based on the assumption that a smaller sub-structure is more precise to the query, keyword proximity search in XML is usually reduced to the problem of finding the smallest sub-structures containing all query keywords. There are two main data models for XML keyword proximity search, *tree* and *digraph* data model.

In tree data model ([18, 9, 16, 21, 10, 20]), LCA is the natural semantics for keyword proximity search, which looks for the lowest common ancestor containing all the keyword in the subtrees. Schmidt et al. [18] introduce the "meet" operator to compute LCAs based on relational-style joins and indices. XRANK [9] presents a ranking method to rank among subtrees rooted at LCAs. The rank is a combination of keyword proximity (distance among query keywords) inside the subtree and refined PageRank [5] of the subtree root element. Li et al. [16] and XKSearch [21] further define Smallest LCAs (or called Meaningful LCAs in [16]) to be LCAs that do not contain other LCAs. Li et al. [16] incorporate Meaningful LCA search in XQuery; whereas XKSearch [21] focuses on studying efficient algorithms to compute Smallest LCA and All LCAs. Sun et al. [19] study a multiway approach to further optimize the computation of Smallest LCAs. Hristidis et al. [10] introduce *MCTs (minimum connecting trees)* to exclude the subtrees rooted at the LCAs that do not contain query keywords. Large XML document are partitioned into XML fragments in [20] to avoid returning meaningless subtrees. XML keyword proximity search techniques based on tree model are generally efficient with Dewey labels. However, they cannot capture ID references and other conceptual connections between elements and they may return many irrelevant results as explained in Section 1. Note that ranking method proposed in XRANK [9] only computes ranks among LCAs, thus it is not adequate when a single LCA is overwhelmingly large.

Most previous algorithms on XML digraph (or graph) model are intrinsically expensive, heuristics-based, because the *reduced tree* problem on graph may be as hard as NP-hard. Li et al [15] show the reduction from minimal reduced tree problem to the NP-hard *Group Steiner Tree* problem on graphs. Cohen et al [8] study the computing complexity of *interconnection semantics* when the the semantical relation among elements can be explicitly specified in XML documents. BANKS [12] uses *Bidirectional* expansion heuristic algorithms to search as small portion of graph as possible and ranks result reduced-trees in approximate order of result generation during the expansion. Given a good estimation of "important" nodes and edges where the expansion starts with, BANKS seems to work well for a small number of results. However, the heuristics for NP-hard problem still has the inherited limitation of slow query response when a large number of results are required. XKeyword [11] is similar to our approach in that both use schema information to reduce search space and present results. However, our approach, based on interconnected objects trees model, clearly distinguishes tree edges (parent-child edges) from conceptual connections to further reduce search space. Also, we discuss a ranking framework on top our searching model which is missing in XKeyword.

Existing ranking methods in IR community are also relevant to our ranking framework. The well-studied TF*IDF cosine similarity [17] or its XML specific variation [6], which takes structural information in XML into consideration for ranking, can be directed incorporated into our ranking framework to indicate the relevance of an object to a keyword query. When the connections among objects are considered for ranking, static PageRank [5] can also be used to compute the importance of an objects according to the application semantics. Notice our novel RelevanceRank is different from PageRank since RelevanceRank is computed dynamically at

query processing phase and RelevanceRank emphasizes on relevance of a object to the query rather than the importance of the objects. RelevanceRank is also different from existing dynamic ranking metrics that exploits connection at query processing, such as HITS [14] and ObjectRank [3]. The reason is RelevanceRank can enforce the co-occurrence of all query keywords in a single logical IRA result while HITS and ObjectRank cannot. As a result, the relevance ranks computed by HITS and ObjectRank may be biased to keywords which are frequent among objects, especially when there are three or more keywords. For example, for query "XML query processing" in ObjectRank, there are only five XML related papers in the top twenty results and the top ranked XML related paper is only the tenth. However, ObjectRank is effective to find objects that are relevant to a single keyword without containing the keyword, which can be incorporated in our model to include objects into the keyword inverted list even they do not contain the keyword.

7. CONCLUSION AND FUTURE WORK

We propose *interconnected object trees* model for keyword search in XML to achieve the efficiency of tree model and meanwhile to capture the connections such as ID references by fully exploiting the property and schema information of XML databases. Based our model, we propose ICA and IRA semantics to identify result objects for keyword queries. We further propose novel RelevanceRank and a ranking framework to incorporate RelevanceRank and other ranking metrics to rank ICA and IRA objects according to their relevance to the query and other application specific requirements. With the conceptual object trees and connections defined bases on schema information, our model offers more flexibility and generality as compared to most existing approaches that focus on only physical sub-structure and connection without schema information. Experimental evaluation show that our approach and algorithm is efficient and outperforms most existing work in terms of result quality.

As a part of future work, we would like to relax the assumption on predefined interested objects and study how our approach, especially RelevanceRank, can be extended to the general situation where experienced users can identify interested objects with path expressions at query time.

8. REFERENCES

[1] Microsoft Libra: http://libra.msra.cn/.

[2] Google Scholar: http://scholar.google.com/.

[3] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proc. of ICDE Conference*, pages 431–440, 2002.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[6] D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching xml documents via xml fragments. In *SIGIR*, pages 151–158, 2003.

[7] M. Charikar, C. Chekuri, T.-Y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed steiner problems. In *SODA Conference*, pages 192–200, 1998.

[8] S. Cohen, Y. Kanza, B. Kimelfeld, and Y. Sagiv. Interconnection semantics for keyword search in xml. In *Proc. of CIKM Conference*, pages 389–396, 2005.

[9] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD*, pages 16–27, 2003.

[10] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava. Keyword proximity search in XML trees. In *TKDE Journal*, pages 525–539, 2006.

[11] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *Proc. of ICDE Conference*, pages 367–378, 2003.

[12] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proc. of VLDB Conference*, pages 505–516, 2005.

[13] B. Kimelfeld and Y. Sagiv. Efficiently enumerating results of keyword search. In *Proc. of DBPL Conference*, pages 58–73, 2005.

[14] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[15] W. S. Li, K. S. Candan, Q. Vu, and D. Agrawal. Retrieving and organizing web pages by information unit. In *Proc. of WWW Conference*, pages 230–244, 2001.

[16] Y. Li, C. Yu, and H. V. Jagadish. Schema-free XQuery. In *VLDB*, pages 72–83, 2004.

[17] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[18] A. Schmidt, M. L. Kersten, and M. Windhouwer. Querying xml documents made easy: Nearest concept queries. In *ICDE*, pages 321–329, 2001.

[19] C. Sun, C.-Y. Chan, and A. K. Goenka. Multiway slca-based keyword search in xml data. In *WWW conference*, 2007.

[20] J. Xu, J. Lu, W. Wang, and B. Shi. Effective keyword search in XML documents based on MIU. In *Proc. of DASFAA Conference*, pages 702–716, 2006.

[21] Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *Proc. of SIGMOD Conference*, pages 537–538, 2005.