EFFICIENT AND EFFECTIVE KEYWORD SEARCH IN XML DATABASE

CHEN BO

(B.Sc.(Hons.), NUS)

A THESIS SUBMITTED FOR

THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2008

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Prof. Ling Tok Wang, for his guidance, support, advice and patience throughout my master studies. His technical, editorial and other advice was essential to the completion of this thesis and he has taught me innumerable lessons and insights that will also benefit my future career.

I would also like to thank Department of Computer Science of National University of Singapore for the strong support for my research work.

My thanks go to Dr. Gillian Dobbie for her valuable comments and suggestions that are of great help to me during the thesis preparation.

My thanks also go to Bao Zhifeng, Lu Jiaheng, Wu Huayu, Wu Wei, Yangfei, Zhu Zhenzhou and all the other previous and current database group members. Their personal and academic helps are of great value to me and the friendships with them have made my graduate life joyful and exciting.

Lastly, I would like to thank my wife, Kang Xueyan and my family. Their dedicated love, support, encouragement and understanding was in the end what made this thesis possible.

Contents

Summary

XML emerges as the standard for representing and exchanging electronic data on the Internet. With increasing volumes of XML data transferred over the Internet, retrieving relevant XML fragments in XML documents and databases is particularly important. Among several XML query languages, keyword search is a proven user-friendly approach since it allows users to issue their search needs without the knowledge of complex query languages and/or the structures of underlying XML databases.

Most prior XML Keyword search techniques are based on either tree or graph (or digraph) data models. In the tree data model, SLCA (Smallest Lowest Common Ancestor) semantics is generally simple and efficient for XML keyword search. However, SLCA results may not be a good choice for direct result display without using application semantic information. Moreover, it cannot capture the important information residing in ID references which is usually present in XML databases. In contrast, keyword search approaches based on the general graph or directed graph (digraph) model of XML capture ID references, but they are computationally expensive (NP-hard).

In this thesis, we propose Tree+IDREF data model for keyword search in XML. Our data model effectively captures XML ID references while also leveraging the efficiency gain of the tree data model. In this model, we propose novel

Lowest Referred Ancestor (LRA) pair, Extended LRA (ELRA) pair and ELRA group semantics as complements of SLCA. We also present algorithms to efficiently compute the search results based on our semantics.

Then, we adopt ORA-SS to exploit underlying schema information in identifying meaningful units of result display. We study and propose rules based on object classes and relationship types captured in ORA-SS to formulate result display for SLCA, ELRA pair and ELRA group results.

We also developed a keyword search demo system based on our approach with DBLP real-world XML database for the research community to search for publications and authors. Some intuitive result ranking is implemented in the demo system. The demo prototype is available at:
http://xmldb.ddns.comp.nus.edu.sg

Experimental evaluation shows keyword search based on our approach in Tree+IDREF data model achieves much better result quality than that based on SLCA semantics in the tree model; and much faster execution time with comparable or better result quality in terms of precision of top-k answers than that based on the digraph model.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to XML

XML (eXtensible Markup Language) is a markup language for documents containing nested structured information. Nowadays, XML emerges as the standard for representing and exchanging electronic data on the Internet.

An XML document consists of nested XML elements starting with the root element. Each element can have attributes and values in addition to nested subelements. In this thesis, unless otherwise specified, we do not make explicit distinction between XML elements and attributes; and we use XML structural nodes or simply nodes to refer to both types. In many XML databases, besides nested relationships, there are also IDs (identifiers) and ID references, represented as IDREFs, to capture node relationships.

Due to the nested structure, XML documents are usually modeled as rooted, labeled trees. In most contexts, a labeling scheme is adopted to assign a numerical label to uniquely identify each node in an XML tree structure. With focus on XML keyword search, we adopt Dewey number labeling scheme [4, 12] since it is

commonly used for XML keyword search applications (i.e. [35, 42, 46] etc).

For example, Figure 1.1 shows an XML document modeled as a rooted tree for a Computer Science department in a university that maintains information about Students, Courses, Lecturers, etc. We include Dewey labels in the figure for later illustration. Besides the nested hierarchical structure, the XML document of Figure 1.1 also includes ID references (i.e. IDREF edges) denoted by dashed lines to indicate the Lecturer-Teaching relationship between lecturers and the courses they are teaching. Each ID reference is captured by a value link from an XML IDREF attribute to an XML element with ID attribute such that the IDREF and ID attributes have the same text value. For example, there is an IDREF edge from node @Course:0.2.0.2.0 to Course:0.1.2 since the text value of @Course:0.2.0.2.0[1] is the same as the identifier (i.e. @id) of Course:0.1.2, which is "CS502". Note we show the reference pointer from @Course:0.2.0.2.0 to Course:0.1.2 directly instead of @id:0.1.2.0 simply because @id:0.1.2.0 is an identifier of Course:0.1.2. We will explain more details about how ID (identifier) and ID references can be represented with XML schema languages in Chapter 3.

1.2 Keyword search and motivation

With increasing volumes of XML data transferred over the Internet, retrieving relevant XML fragments in XML documents and databases is particularly important. Several query languages have been proposed, such as XPath [9] and XQuery [11]; and researchers have devoted a great amount of work ([8, 14, 16, 19, 29, 37, 38, 43], etc) to efficient processing of these query languages.

However, XPath and XQuery are usually too complex for novice users to

[1] We show the link without text values of XML IDREF attributes (i.e. @Course) for simplicity.

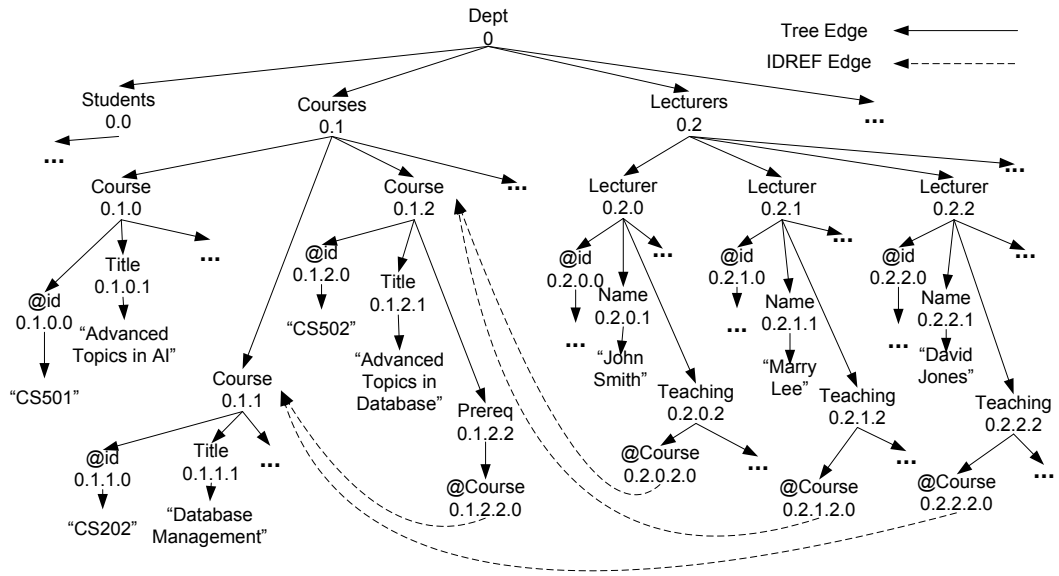


Figure 1.1: Example XML document of computer science department with Dewey labels (Nodes prefixed with @ are XML attributes instead of XML elements)

master. Moreover, they require users to have a clear understanding of the underlying schema information, which potentially prohibits even experienced database people from issuing queries against an unfamiliar XML database. As a result, keyword search in XML recently drawn the attention of many researchers due to its proven user-friendliness that allows users to issue their search needs without the knowledge of complex query languages and/or the structures of underlying XML databases.

The majority of the research efforts in XML keyword search focus on keyword proximity search in either the *tree model* or the *general graph (or digraph) model*. Both approaches generally assume a smaller sub-structure of the XML document that includes all query keywords indicates a better result.

1.2.1 Tree model for XML keyword search

In the tree model, *SLCA (Smallest Lowest Common Ancestor)* ([35, 42, 46]) is a simple and effective semantics for XML keyword proximity search. Each SLCA result of a keyword query is an XML subtree rooted at one XML node[2] that satisfies two conditions. First, the node covers all keywords in its subtree; second, it has no single proper descendant subtree to cover all query keywords. For example, in Figure 1.1, the SLCA result of keyword query "CS202 Database Management" is the Course:0.1.1 node (i.e. Course node with Dewey label 0.1.1).

However, the SLCA semantics based on the tree model does not capture ID reference information which is usually present and important in XML databases. As a result, SLCA is insufficient to answer keyword queries that require the information in XML ID references and may return a large tree including irrelevant information for those cases. For example, in Figure 1.1, consider a search intention that a searcher wants to look for whether lecturer Smith teaches some Database course and also the information of the course and/or Smith if so. In this case, "Smith Database" is a reasonable keyword query. However, the SLCA result for this query without considering ID references is the root of the whole XML database, which is overwhelming and will frustrate the searcher.

Moreover, SLCA results may not be a good choice for direct result display without using application semantic information. For example, the SLCA result for query "Database Management" in Figure 1.1 is Title:0.1.1.1 of a course. However, it is not informative to display just the title without other information of the course. In this case, it is better to display the information of the course (i.e. Course:0.1.1) with the matching title.

[2]In the following, we use the term *subtree* and *node* interchangeably to refer to a subtree rooted at the corresponding node when there is no ambiguity.

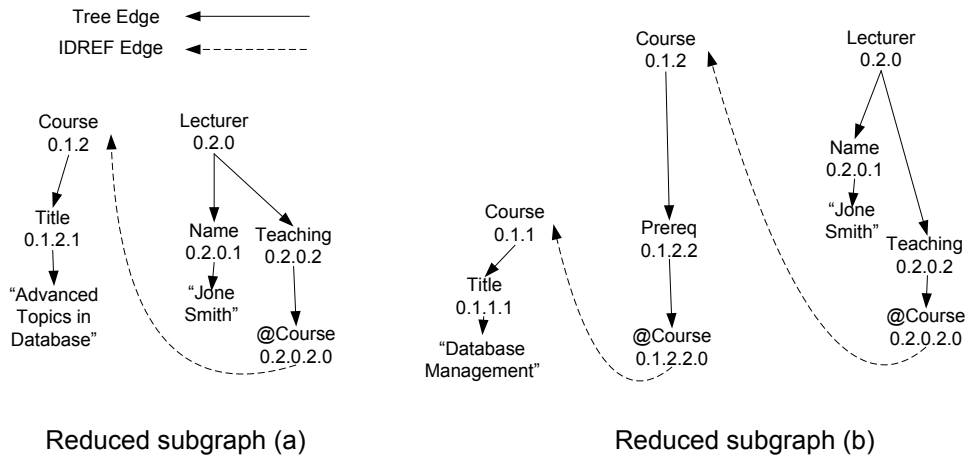


Figure 1.2: Example reduced subgraph results for query "Smith Database" in Figure 1.1

1.2.2 Graph model for XML keyword search

On the other hand, XML documents can be modeled as graphs (or digraphs) when ID reference edges are taken into account. With the graph (or digraph) model, a keyword search engine captures a richer semantics than that based on the tree model. The key concept in the existing semantics is called *reduced subgraph* ([20]). Given an XML graph G and a list of keywords K , a connected subgraph G' of G is a reduced subgraph with respect to K if G' contains all keywords of K, but no proper subgraph of G' contains all these keywords.

For example, with the XML document shown in Figure 1.1, some possible reduced subgraph results for query "Smith Database" are shown in Figure 1.2.

Note, following [30], when there is a forward edge from node u to v in the digraph model, we also consider there is a backward edge from v to u in this thesis. This is to admit more interesting sub-structures in the results. For example, in Figure 1.1, both Lecturers John Smith and Marry Lee teach Course "CS502 Advanced Topics in Database" shown in Figure 1.3. If we do not consider the backward edges from Course nodes to (the subtrees of) Lecturer nodes, we will

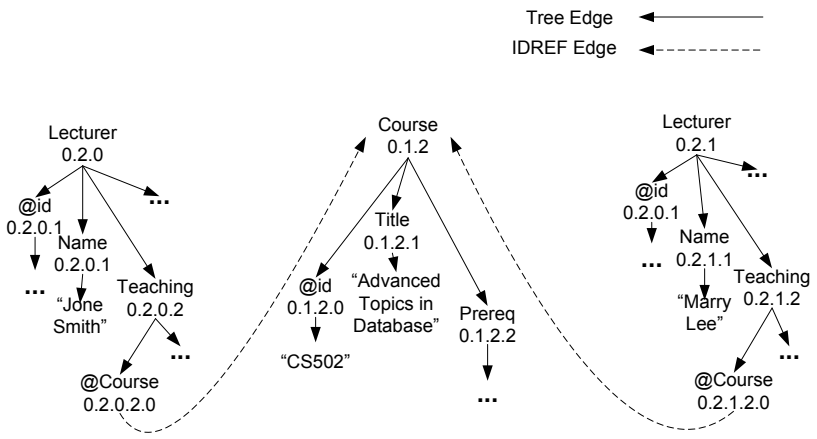


Figure 1.3: Abstract connection of two lecturers teaching the same course

not be able to find the meaningful connection pattern that Smith and Lee teach the same course for keyword query "Smith Lee" since we cannot reach Lecturer nodes from Course nodes.

Although there exist very efficient algorithms on SLCA with the tree model (e.g. [23, 42, 46]), unfortunately, to our knowledge, there is no efficient algorithm for reduced subgraphs. The reason is twofold. Firstly, the number of all reduced subgraphs may be exponential in the size of G. In contrast, the number of LCA subtrees is bounded by the size of the given XML tree. Note that different reduced subgraphs present different connected relationships in the real world; and most of them cannot be easily considered as redundant results. Secondly, if we consider enumerating results by increasing sizes of reduced subgraphs for ranking purposes according to the general assumption of XML keyword proximity search, this problem can be NP-hard; the well-known *Group Steiner tree* problem [15] for graph can be reduced to it (see reduction approach in [34]). Although there are a multitude of polynomial time approximation approaches (e.g. [15, 22]) that can produce solutions with bounded errors for *minimal Steiner* problem, they require an examination of the entire graph. These algorithms are not desirable

since the overall graph of XML keyword search is often very large.

1.3 Contribution

Motivated by the limitations of the tree and general graph (or digraph) models for XML keyword search, in this thesis, we study a novel special graph, *Tree + IDREF* model, to capture ID references which are missed in the tree model; and meanwhile to achieve better efficiency than the general graph model by distinguishing reference edges from tree edges in XML to leverage the efficiency benefit of the tree model.

In particular, we propose novel *LRA pair* (*Lowest Referred Ancestor pair*) semantics. Informally, LRA pair semantics returns a set of lowest ancestor node pairs such that each node pair (and their subtrees) in the set are connected by ID references and the pair together cover all keywords in their subtrees. Since ID references in XML documents usually indicate relevance between XML nodes, it is reasonable to speculate that such connected and relevant pairs covering all keywords are likely to be relevant to the keyword query. For example, consider the query "Smith Database" in Figure 1.1 again. The result of LRA pair semantics is the pair of nodes Lecturer:0.2.0 and Course:0.1.2 that are connected by ID reference and together cover all keywords in their subtrees, which can be understood as Smith teaches the course indicated by the ID reference. Then, we extend LRA pairs that are directly connected by ID references to node pairs that are connected via intermediate node hops by a chain of ID references; which we call *ELRA pair* (*Extended Lowest Referred Ancestor pair*) semantics. Finally, we further extend ELRA pair to *ELRA group* to define the relationships among two or more nodes which together cover all keywords and are connected with ID

references.

The contributions of this thesis are summarized as follows:

(1) We introduce *Tree + IDREF* data model for keyword proximity search in XML databases. In this model, we propose novel LRA pair, ELRA pair and ELRA group semantics as complements of well-known SLCA to find relevant results for keyword proximity search. The data model and search semantics are general and applicable to most XML databases that maintain ID references.

(2) We study and analyze efficient polynomial algorithms to evaluate keyword queries based on the proposed semantics.

(3) We further discuss some guidelines for result display based on application schema semantics which can be captured in ORA-SS [44] so that we can provide more meaningful search results when information of schema semantics is available.

(4) We developed ICRA keyword search prototype for DBLP dataset to provide keyword search service to research community to search for publications and authors. Our ICRA system is available at: http://xmldb.ddns.comp.nus.edu.sg.

(5) We conduct extensive experiments with our keyword search semantics. The results prove the superiority of the proposed model and search semantics over existing approaches.

1.4 Thesis organization

In the rest of the paper, we first review related work in Chapter 2.

In Chapter 3, we discuss the background and data model of this work. It includes a brief introduction to XML, two existing XML schema languages (DTD and ORA-SS) and Dewey labeling scheme. We also emphasize the existence of ID references in XML, and propose our Tree + IDREF data model.

In Chapter 4, we introduce proposed keyword search semantics, including LRA pair, ELRA pair and ELRA group semantics. We also address their applicability to general XML databases. A detailed study of data structures and algorithms to compute results based on our search semantic are also presented in this chapter.

In Chapter 5, we discuss some guidelines for result display in XML keyword search based on semantic information of underlying XML database which can be captured in ORA-SS. We also present descriptions of the features of our online keyword search demo prototype for DBLP bibliography.

In Chapter 6, we experimentally compare our Tree + IDREF data model with the tree and digraph models for keyword search. We also show the effectiveness of our online demo system in terms of search result quality.

Finally, we conclude this thesis and propose the future work in Chapter 7.

Some of the material in this thesis appears in our papers [18], [17] and [7].

Chapter 2

Related Work

2.1 XML keyword search with the tree model

Extensive research efforts have been conducted for XML keyword search in the tree data model ([23, 26, 35, 40, 42, 45, 46]) based on LCA (Lowest Common Ancestors), SLCA (Smallest Lowest Common Ancestors) semantics and their variations.

The first area of research relevant to this work is the computation of LCAs (Lowest Common Ancestors) of a set of nodes based on the XML tree model .

Schmidt et al. [40] introduce the "meet" operator to compute LCAs based on relational-style joins. The semantics of the meet operator is the nearest concept (i.e. lowest ancestor) of XML nodes. It operates on multiple sets (i.e. relations) where all nodes in the same set have the same prefix path. The meet operator of two nodes v_1 and v_2 is implemented efficiently using joins on relations, where the number of joins is the number of edges of the shorter one of the paths from v_1 and v_2 to their LCA.

XRANK [23] presents a ranking method to rank subtrees rooted at LCAs.

XRANK extends the well-known Google's PageRank [13] to assign each node u in the whole XML tree a pre-computed ranking score, which is computed based on the connectivity of u in the way that u is given a high ranking score if u is connected to more nodes in the XML tree by either parent-child or ID reference edges. Note the pre-computed ranking scores are independent of queries. Then, for each LCA result with descendants $u_1, ...u_n$ to contain query keywords, XRANK computes its rank as an aggregation of the pre-computed ranking scores of each u_i decayed by the depth distance between u_i and the LCA result. XRANK also proposes a stack-based algorithm to utilize inverted lists of Dewey labels. A inverted list of a keyword is a list of Dewey labels whose corresponding nodes directly contains the keyword. The algorithm maintains a result heap and a Dewey stack. The result heap keeps track of the top k results seen so far. The Dewey stack keeps the ID and rank of the current dewey ID, and also keeps track of the longest common prefixes computed during the merge of the inverted lists. The stack algorithm merges all keyword lists and computes the longest common prefix of the node with the smallest Dewey number from the input lists and the node denoted by the top entry of the stack. Then it pops out all top entries containing Dewey components that are not part of the common prefix. If a popped entry n contains all keywords, then n is the result node. Otherwise, the information about which keywords n contains is used to update its parent entry's keywords array. Also, a stack entry is created for each Dewey component of the smallest node which is not part of the common prefix, to push the smallest node onto the stack. The action is repeated for every node from the sort merged input lists.

XSearch [21] proposes a variation of LCA to find meaningfully related nodes as search results, called interconnection semantics. According to interconnection semantics, two nodes are considered to be semantically related if and only if

there are no two distinct nodes with the same tag name on the paths from the LCA of the two nodes to the two nodes (excluding the two nodes themselves). Several examples are provided to justify the usefulness and meaningfulness of the proposed interconnection semantics. For example, in Figure 1.1, id:0.1.2.0 and Title:0.1.2.1 are considered semantically related since there are no two nodes of the same tag on the paths from their LCA (Course:0.1.2) to the two nodes. However, it is obvious interconnection semantics does not work for all cases. For example, Course:0.1.0 and Course:0.1.2 are not so semantically related, but they are considered related by interconnection semantics.

As LCA semantics is defined on a set of nodes instead of a set of node lists, LCA itself is not well suited for keyword search applications where each query keyword usually has a list of XML nodes that contain it. For example, in Figure 1.1, keyword "Advanced" matches two nodes Title:0.1.0.1 and Title:0.1.2.1; while "Database" also matches two nodes Title:0.1.1.1 and Title:0.1.2.1. As a result, the LCAs of query "Advanced Database" include both Courses:0.1 (due to Title:0.1.0.1 containing "Advanced" and Title:0.1.2.1 containing "Database") and Title:0.1.2.1 (containing both query keywords). It is obvious the first LCA (i.e. Courses:0.1) is not meaningful for this query. Both [35] and [46] address the problem. In [35], Li et al. propose Meaningful LCA and XKSearch [46] proposes Smallest LCA. Both Meaningful LCA and Smallest LCA (SLCA) are essentially similar to LCAs that do not contain other LCAs[1]. In other words, the SLCA result of a keyword query is the set of nodes that each satisfies two conditions. First, each node in the set covers all query keywords in its subtree. Second, each node in the set does not have a single descendant to cover all query keywords.

Li et al. [35] incorporates SLCA (which they call Meaningful LCA) in XQuery

[1]In this thesis, we unify the two terms (i.e. Meaningful LCA and Smallest LCA) as Smallest LCA (or SLCA)

and proposes Schema-Free XQuery where predicates in an XQuery can be specified through the concept of SLCA. With Schema-Free XQuery, users are able to query an XML document without full knowledge of the underlying schema. When users know more about the schema, they can issue more precise XQueries. However, when users have no ideas of the schema, they can still use keyword queries with Schema-Free XQuery. [35] also proposes a stack based sort merge algorithm to compute SLCA results with Dewey labels, which is similar to the stack algorithm in XRANK [23].

XKSearch [46] focuses on efficient algorithms to compute SLCAs. It also maintains a sorted inverted list of Dewey labels in document order for each keyword. XKSearch addresses an important property of SLCA search, which is, given two keywords k_1 and k_2 and a node v containing k_1, only two nodes in the inverted list of k_2 that directly proceeds and follows v in document order are able to form a potential SLCA solution with v. Based on this property, XKSearch proposes two algorithms: Indexed Lookup Eager and Scan Eager algorithms. Indexed Lookup Eager scans the shortest inverted list of all query keywords and probes other inverted lists for SLCA results. During the probing process, nodes in other inverted lists that do not contribute to the final results can be effectively skipped. In contrast, Scan Eager algorithm scans all inverted lists for cases when all query keyword inverted lists have similar sizes. Experimental evaluation shows the two algorithms are superior than the stack based algorithm in [35]. Indexed Lookup Eager is better than Scan Eager when the shortest list is significantly shorter than other lists of query keywords; or slightly slower but comparable to Scan Eager when all inverted lists of query keywords have similar lengths.

Sun et al. [42] make a further effort to improve the efficiency of computing SLCAs. It discovers the fact that we may not need to completely scan the short-

est keyword list for certain data instances to find all SLCA results. Instead, some Dewey labels in the shortest keyword list can be skipped for faster processing. As a result, Sun et al. propose Multiway-based algorithms to compute SLCAs. In particular, Multiway SLCA computes each potential SLCA by taking one keyword node from each kewyord list in a single step instead of breaking the SLCA computation to a series of intermediate binary SLCA computations. As compared to XKSearch [46] where the algorithm can be viewed as driven by nodes in the shortest inverted list; Multiway SLCA picks an "anchor" node from all query keyword inverted lists to drive the SLCA computation. In this way, it is able to skip more nodes than XKSearch [46] during SLCA computation. Though algorithms in Multiway SLCA [42] have the same theoretical time complexity as Indexed Lookup Eager algorithm in [46], experimental results show the superiority of Multiway-based algorithms. In [42], Sun et al. also generalizes the SLCA semantics to support keyword search to include both AND and OR boolean operators, by transferring queries to disjunctive normal forms and/or conjunctive normal forms.

Besides LCA and SLCA, Hristidis et al. [26] propose Grouped Distance Minimum Connecting Trees (GDMCT) and Lowest GDMCT as variations of LCA and SLCA for XML keyword search. The main difference between GDMCT and LCA is that GDMCT identifies not only the LCA nodes, but also the paths from LCA nodes to their descendants that directly contain query keywords. Similarly, Lowest GDMCT identifies not only the SLCA nodes, but also the paths from SLCA nodes to descendants containing query keywords. GDMCT is useful to show how query keywords are connected to the LCA (or SLCA) nodes in result display, which is classified as path return (in contrast to subtree return in LCA and SLCA) in [36].

XSeek [36] addresses the search intention of keyword queries to find meaningful return information based on the concept of object classes (which they call entities) and the pattern of query matching. It proposes heuristics to infer the set of object classes in an XML document and also heuristics to infer the search intentions of keyword queries based on keyword match patterns. Its main idea is if an SLCA result is an object or a part of an object, we should consider the whole object subtree or some attribute of the object specified in the query that is not the SLCA for result display.

Recently, Li et al. [33] propose Valuable LCA semantics, which is another variation of LCA and SLCA. Its main idea is that an LCA of m nodes $n_1, n_2, ..., n_m$ is valuable if and only if there are no nodes of the same tag name along the paths from the LCA to $n_1, n_2, ..., n_m$, except nodes in $n_1, n_2, ..., n_m$ may have the same tag. This is similar to the idea of interconnection semantics in [21]. It further proposes a variation of Dewey labeling, called MDC to infer the tag names in the path, which is essentially similar to Extended Dewey in [38].

XML keyword proximity search techniques based on the tree model are generally efficient. However, they cannot capture important information in ID references which are indications of node relevance in XML and they may return overwhelming (or not informative) information as explained in Chapter 1. Note that the ranking method proposed in XRANK [23] only computes ranks among LCAs, thus it is not adequate when a single LCA is overwhelmingly large. GDMCT in [26] identifies how query keywords are connected in each LCA or SLCA result, which is useful in result display to enable the searcher to understand the inclusion of each result However, without considering ID references, GDMCT is similar to search by keyword disjunction when the root of a GDMCT is overwhelmingly large. XSeek [36] based on the concept of objects is able to identify meaningful

result units and to avoid returning overwhelming information. However, it considers neither ID references nor relationships between objects. As a result, XSeek may miss meaningful results of query relevant object relationships that contain all keywords.

2.2 Keyword search with the graph model

XML databases can also be modeled as graphs (or digraphs) when ID references edges are taken into account. In this part, we first present the overall search and result semantics in the graph (or digraph) model. Then, we review some related work of keyword search in relational databases and/or XML databases with the graph (or digraph) model.

Keyword search in databases with the graph (or digraph) model was first addressed for relational databases in [5,10,27], etc. They view a relational database as a graph G where tuples of relations are modeled as nodes N and relationships such as foreign-key are modeled as edges E (i.e. $G = (N, E)$). Similarly, XML databases can also be modeled as graph G for keyword search ([10, 28], etc) in the way that XML elements/attributes are viewed as nodes N and relationships such as node containment (i.e. parent-child relationship) and ID references are modeled as edges E.

In the graph model, answers to a keyword query $k_1, k_2, ..., k_n$ in a (either relational or XML) database graph G are usually modeled as connected subgraphs of G such that 1) each answer subgraph G' contains all keywords of query $k_1, k_2, ..., k_n$ in its nodes (i.e. tuples in relational database or elements/attributes in XML context) and 2) no nodes in G' can be removed from G' to form another smaller subgraph G'' to contain all query keywords. Each answer subgraph G'

is usually referred to as a *reduced subgraph* of query $k_1, k_2, ..., k_n$ in G[2] [20]. Reduced subgraphs of a query are ranked according to their sizes (e.g. [5, 27, 28], etc.) with the intuition that a smaller reduced subgraph usually indicates a closer connection between query keywords, thus a more meaningful result.

However, searching all reduced subgraphs ranked by size for a keyword query is NP-hard. Li et al [34] show the translation between minimal (or ordered-by-size) reduced subgraphs problem and the NP-hard *Group Steiner Tree* problem on graphs. The *Steiner tree* problem [24] is known as the problem of finding the minimum weighted connected subgraph, G', of a given graph G, such that G' includes all vertices in a given subset of R of G. *Group Steiner tree* problem is an extension of *Steiner tree* problem, where we are given a set $\{R_1, ..., R_n\}$ of sets of vertices such that the subgraph has to contain at least one vertex from each group $R_i \in \{R_1, ..., R_n\}$. Both *Steiner Tree* and *Group Steiner Tree* problems are proven NP-hard. Therefore, most previous algorithms for keyword search with the graph (or digraph) model are intrinsically expensive, heuristics-based.

Banks [10] adopts backward expanding search heuristics to find ranked reduced subgraphs of query keywords in digraphs. Each node in the graph is assigned a weight which depends on the prestige of the node; and each edge is also given a weight based on schema to reflect the strength of the relationship between two nodes. It computes, ranks and outputs results incrementally in approximate order of result generation. Given a set of keywords $\{k_1, ..., k_n\}$, their inverted lists $\{l_1, ..., l_n\}$ and the union $L = \bigcup l_i \in \{l_1, ..., l_n\}$ of query keyword inverted lists, backward expanding algorithm in [10] concurrently runs $|L|$ copies of Dijkstra's single source shortest path algorithm, one of each keyword node $n \in L$, with n as the source. Each copy of the single source shortest path algorithm traverses the

[2]Some people call G' a reduced subtree since G' can be also viewed as a tree.

graph edges in the reverse direction in order to find a common vertex from which a forward path exists to one keyword node in each inverted list $l_i \in \{l_1, ..., l_n\}$. Once a common vertex is found, it is identified as the root of a connection tree, thus a search result.

A subsequent work [30] of Banks proposes bidirectional search to improve on backward expanding search by allowing forward search from potential roots towards leaves. During bidirectional search, each node is assigned an activation score, reflecting how "active" it is to be expanded next. The initial activation value of a keyword node in one inverted list is inversely proportional to the size of the inverted list so that nodes containing a rare keyword will be expanded (backward) first. It maintains two priority queues, one for backward expanding Q_b and one for forward expanding Q_f. All nodes in inverted lists are initially kept in backward expanding queue Q_b. Once a node u with highest activation in Q_b is expanded backward, it transfers its partial activation value to other nodes that are expanded to from u and puts those nodes into Q_b; now u is put into Q_f from Q_b with remaining activation value. Similarly, once a node u with highest activation in Q_f is expanded, it also transfers its activation value to other nodes and puts them into Q_f. Search results are identified during the expanding when a node is found to be able to connect all keywords. Experimental results in [30] shows bi-directional expanding is more efficient than backward expanding.

Bidirectional expanding approach in Banks is random in nature and suffers poor worst-case performance. Moreover, Bidirectional expanding approach requires the entire visited graph in memory which is infeasible for large databases. Blinks [25] address these problems by using a bi-level index for pruning and accelerating the search. Its main idea is to maintain indexes to keep the shortest distance from each keyword to all nodes in the entire database graph. To reduce

the space of such indexes, Blinks partitions a data graph into blocks: the bi-level index stores summary information at the block level to initiate and guide search among blocks, and more detailed information for each block to accelerate search within blocks. Experiments of Blinks [25] show its benefit in improving search efficiency. However, index maintenance is an inherent drawback of Blinks, since adding or deleting an edge has global impact on shortest distances between nodes.

DBXplorer [5] and Discover [27] exploit relational schema to reduce search space for keyword search in relational databases.

Given a set of query keywords, DBXplorer returns all rows (either from single tables, or by joining tables connected by foreign-key relationships) such that each row contains all query keywords (which is a relaxed form of reduced subgraphs). DBXplorer has two steps to enable keyword search in an existing database, *Publish* (pre-process) and *Search* (query processing). In the publish step, a symbol table is created, which is similar to inverted lists to determine the locations of query keywords in the database. The location granularity of the symbol table can be either cell level or column level, depending on several measures, such as, the existence or not of a column index, space and time tradeoff during symbol table creation and query processing. In the search step, the symbol table is first looked up to identify the tables containing query keywords. Then, according to schema graph where each node is a relation and each edge is a foreign-key, a set of subgraphs are enumerated to build join trees. Each such join tree represents a join of relations such that the join result contains rows that potentially contain all query keywords. Finally, a join SQL statement is executed for each enumerated join tree and rows with all query keywords are selected from join results.

Discover [27] improves over DBXplorer to consider solutions that include two tuples from the same relation and to exploit the reusability of join trees for

better efficiency. Result semantics in Discover is reduced subgraphs of query keywords, which they call *Minimal Total Join Network (MTJNT)*. Discover uses master index (also similar to inverted lists) to identify all tuples that contain a given keyword for each relation. During query processing for a given query $K = \{k_1, k_2, ..., k_3\}$, Discover first identifies relations that contain some keywords in K. Each such relation R_i is partitioned horizontally into tuple sets $R_i^{K'}$ for all subsets $K' \subset K$ such that $R_i^{K'}$ contains tuples of R_i that contain all keywords of K' and no other keywords in K. Then, with schema graph, Discover generates all candidate networks, each of which is a graph of tuple sets $R_i^{K'}$ such that the join result of all tuple sets in a candidate network 1) potentially contains reduced subgraphs of all query keywords 2) but does not contain subgraph with all keywords that is not a reduced subgraph. Finally, a plan of joining tuple sets for each candidate network is generated and executed to exploit the reusability of intermediate join results for better efficiency. Discover propose a greedy algorithm to choose intermediate results for reuse; while the selection of the optimal execution plan is NP-complete as shown in Discover [27].

A recent work [47] studies the problem of finding all records in a relation such that each result record contains all query keywords. The problem addressed in [47] can be viewed as a sub-problem of DBXplore and Discover since [47] does not explore foreign key relationships. Moreover, [47] does not use Inverted Lists for keyword search. As a result, the technique proposed in [47] is over complicated and very inefficient. In contrast, DBXplore and Discover can handle the problem more efficiently with Inverted Lists.

Since DBXplorer [5] and Discover [27] require relational schema during query processing, they cannot be directly applied for XML keyword search if the XML databases cannot be mapped to a rigid relational schema.

XKeyword [28] extends the work of Discover to handle keyword search in XML databases with the graph model. It requires database administrator to manually split the schema graph into minimal self-contained information pieces, which are called *Target Schema Segments (TSS)*. The edges connecting the data instances of TSSs in schema graph are stored in the connection tables. Besides, redundant connection relations connecting several TSSs based on decomposition of TSS graph are materialized and used to improve the performance of the search. During query processing, XKeyword first retrieves the schema nodes from the inverted index, such that instances of those schema nodes in XML data contain query keywords. Then, it exploits the schema graph to generate a complete and non-redundant set of connection trees (similar to candidate networks in Discover [27]) between them. Similar to Discover, each candidate network may produce a number of answers to the keyword query, when evaluated on the XML graph. However, XKeyword is laborious in that database administrator's knowledge is necessary in all stages of indexing, presenting results and query processing. Moreover, redundant materialization of connection relations imposes problems in updating the connection relations, in addition to space overheads.

In summary, keyword search approach in the graph (or digraph) models are inherently expensive due to its NP-hard nature. DBXplorer [5], Discover [27] and XKeyword [28] exploit schema information to reduce search space during query processing. The former two are designed for relational databases and cannot be directly used for XML; while the last one (i.e. XKeyword [28]) is designed for XML databases. However, XKeyword [28] is laborious and requires specification from DBA for each individual application whereas our approach does not require DBA's efforts during query processing though their optional efforts can be useful in our case. Techniques in Banks project [10, 30] can be directly used for XML

databases. However, our experimental results show they are significantly inefficient as compared to our approach in Tree+IDREF model. Blinks [25] improves the efficiency over techniques in Banks with tradeoffs in index size and ease of maintenance. It is orthogonal to our indexing approach and can be extended and incorporated to improve our search efficiency with the same tradeoffs in index size and ease of maintenance.

Chapter 3

Background and Data Model

3.1 XML data

XML stands for eXtensible Markup Language, which is a markup language for documents containing structured information. Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.

Tags are basic markups in XML, which are enclosed in angle brackets. An XML document consists of nested XML elements starting with the root element. An XML element is everything from (including) the element's start tag to (including) the element's end tag. Each element can have attributes and text values in addition to nested subelements. Each attribute has further text values. In many XML databases, there are also IDs and ID references represented as IDREFs to indicate relationships between XML elements.

Example 1 *Figure 3.1 shows an example XML data document fragment that maintains information for a Computer Science department in one university. The document has one root element,* Dept. *In the inside rectangle, we highlight*

```
<Dept>
    <Students> ... </Students>
    <Courses>
        <Course id="CS501">
            <Title> Advanced Topics in AI </Title>
            ...
        </Course>
        <Course id="CS202">
            <Title> Database Management </Title>
            ...
        </Course>
        <Course id="CS502">
            <Title> Advanced Topics in Database </Title>
            <Prereq Course="CS202"/>
            ...
        </Course>
        ...
    </Courses>
    <Lecturers>
        <Lecturer id="L01">
            <Name> John Smith </Name>
            <Teaches Course="CS502">
                <Year> 2007 </Year>
                ...
            </Teaches>
            ...
        </Lecturer>
        <Lecturer id="L02">
            <Name> David Lee </Name>
            <Teaches Course="CS502"/>
            ...
        </Lecturer>
        <Lecturer id="L03">
            <Name> Marry Jones </Name>
            <Teaches Course="CS202"/>
        </Lecturer>
    </Lecturers>
</Dept>
```

Figure 3.1: Example XML data fragment

```
<!ELEMENT Dept          (Students, Courses, Lecturers)>

<!ELEMENT Students      (...)>
...

<!ELEMENT Courses       (Course+)>
<!ELEMENT Course        (Title, Prereq*, Description)>
<!ATTLIST Course        id ID #REQUIRED>
<!ELEMENT Title         (#PCDATA)>
<!ELEMENT Prereq        EMPTY >
<!ATTLIST Prereq        Course IDREF #REQUIRED>
<!ELEMENT Description   (#PCDATA)>

<!ELEMENT Lecturers     (Lecturer+)>
<!ELEMENT Lecturer      (Name, Teaching+, Address?
                              Hobby*)>
<!ATTLIST Lecturer      id ID #REQUIRED>
<!ELEMENT Name          (#PCDATA)>
<!ELEMENT Teaching      (Year, Semester) >
<!ATTLIST Teaching      Course IDREF #REQUIRED>
<!ELEMENT Address       (#PCDATA)>
<!ELEMENT Hobby         (#PCDATA)>
```

Figure 3.2: Example DTD for XML data in Figure 3.1

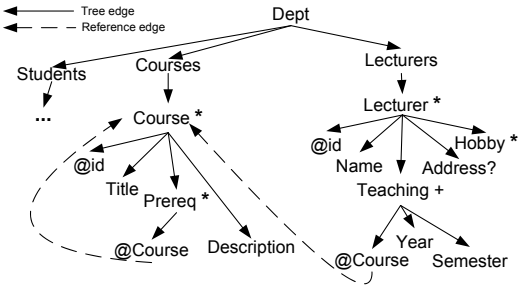


Figure 3.3: Graph representation of DTD in Figure 3.2 (@ denotes attributes)

one XML element, Course*. The information of this element includes everything between its start tag <*Course*> and end tag <*/Course*>. A* course *element has further nested attribute* id *and nested elements* Title *and* Prereq*. Finally, attribute* id *has text value "CS502" while* Title *has text value "Advanced Topics in Database". With the help of DTD or other schema languages which we will discuss shortly,* id *attribute of each* Course *can be recognized as the identifier of the* Course *element while* Course *attribute of each* Prereq *element can be recognized as an ID reference to a particular* Course *element with the specified* id *value.*

3.2 Schema languages for XML

There are several existing languages to specify the schema of an XML database. In this thesis, we present a brief description of two schema languages: XML DTD (Document Type Description) and ORA-SS (Object-Relationship-Attribute model for SemiStructured data).

3.2.1 XML DTD

Document Type Description (DTD) is a commonly used simple schema language to describe the structure of an XML document. A very basic description of DTD is given here.

From the DTD point of view, the building blocks of XML documents of interest are *element*, *attribute*, *#PCDATA* and *#CDATA*. For each XML element, DTD specifies its tag name. An element can either be empty or contain further information in forms of sub-elements, attributes and text values. For empty elements, DTD specifies them as *EMPTY* together with their tag names. For elements with further information, DTD specifies its nested information as #PCDATA (i.e. text values) or attributes or the tag names of sub-elements using regular expressions with operators * (a set of zero or more elements), + (a set of one or more elements), ? (optional) and | (or). Sub-elements without operators are mandatory (one and only one element) by default. Text values nested in elements are specified as #PCDATA; while text values of XML attributes are usually specified as #CDATA. Attributes can have further predefined types in DTD. Some particular attribute types of interest are "ID" and "IDREF". "ID" type indicates the attribute value is an identifier of the attribute's parent element (i.e. unique, non-nullable and always present); while "IDREF" type indicates the

attribute value is a reference to an element with specified identifier (ID) value.

Example 2 *Figure 3.2 shows the DTD for our example department XML data. The root element* Dept *has three mandatory sub-elements* Students, Courses *and* Lecturers *and each has one and only one occurrence under* Dept. Courses *element has more than one nested* Course *element while each* Course *in turn has* Title, Prereq *and* Description *sub-elements.* Title *and* Description *are mandatory for each* Course *and they contain only text values (i.e.* #PCDATA*) but no further nested sub-elements.* Prereq *can have zero or more occurrences nested in each* Course. *Each* Prereq *has one IDREF typed attribute* Course, *but has neither sub-elements nor text values indicated by* EMPTY. *The value of each IDREF typed attribute* Course *under* Prereq *is the identifier of some other element to represent an ID reference from* Prereq *(to a* Course *element in this case evidenced from XML data). Finally,* Address *nested in* Lecturer *is marked with ?, indicating each* Lecturer *can have zero or one* Address *in the XML document.*

Since DTD also has inherited hierarchical structure, we can use graphs to represent DTDs for easy illustration. For example, Figure 3.3 shows the graph representation of DTD in Figure 3.2, where XML attributes are annotated by @.

3.2.2 ORA-SS

The ORA-SS (Object-Relationship-Attribute model for SemiStructured data) is a semantic rich schema language for XML documents. It can capture useful semantic information which is missed in other schema languages. In this part, we first present a brief introduction to ORA-SS; then we highlight two kinds of semantic information that are important to meaningful keyword search but cannot be captured by DTD.

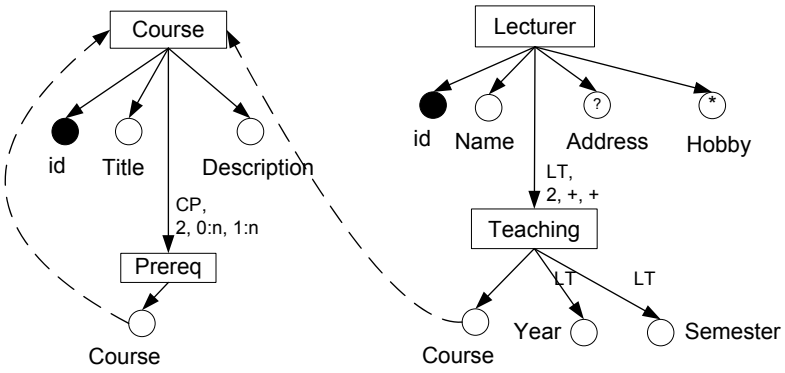


Figure 3.4: Example ORS-SS schema diagram fraction for XML data in Figure 3.1

ORA-SS data model has three basic concepts: *object class*, *relationship type* and *attribute*. An *object class* is similar to an entity type in an ER diagram. A *relationship type* describes a relationship among object classes. *Attributes* are properties belonging to an object class or a relationship type. A full description of the data model can be found in [44].

An ORA-SS *schema* represents an object class as a labeled rectangle, an attribute as a labeled circle. All attributes are assumed to be mandatory and single valued, unless the circle contains a "?" indicating it is optional and single valued, "+" indicating it is mandatory and multi-valued, and "*" indicating it is optional and multi-valued. Identifier of an object class is a filled circle.

The relationship type between object classes is assumed on any edge between two objects, and described by a label in the form of "*name, n, p, c*" in ORA-SS. Here, *name* denotes the name of relationship type; n indicates the degree of the relationship type. A relationship of degree 2 (i.e. a binary relationship) is between two objects, parent and child of the relationship. A relationship of degree 3 (i.e. a ternary relationship) relates three objects. In a tertiary relationship, there is a binary relationship between two objects and a relationship between this binary relationship and the other object. The parent, in this case, is the

binary relationship and child is the other object. In the label of a relationship, p indicates the participation constraints of the parent of the relationship, and c is the participation constraints of the child of the relationship. p and c are defined using the min:max notation, with shorthand of ?(0:1), *(0:n) and +(1:n). A relationship type can also have attributes. The attribute of a relationship type has the name of the relationship type to which it belongs on its incoming edge, while the attribute of an object class has no edge label.

Finally, solid edge in ORA-SS represents nested relationship of XML while dashed edge represent references. A reference depicts an object referencing another object, and we say a *reference* object references a *referenced* object. The reference and referenced objects can have different labels and relationships. References are also used to model recursive and symmetric relationships.

Example 3 *Figure 3.4 shows the ORA-SS schema diagram for the XML data in Figure 3.1. The rectangles labeled* Course, Lecturer, Teaching *and* Prereq *are four object classes, and attributes* id *of* Course *and* id *of* Lecturer, *are the identifiers of* Course *and* Lecturer *respectively. For each* Lecturer, Name *is a mandatory single valued attribute,* Address *is an optional single valued attribute, and* hobby *is an optional multi-valued attribute.*

There are two binary relationship types, namely CP *and* LT. CP *is a recursive relationship type between* Course *and* Prereq *(prerequisite), and* LT *is a relationship type between* Lecturer *and* Teaching. *Both* CP *and* LT *are many-to-many relationships, where each* Course *can have zero or more* Prereqs, *each* Prereq *(or* Lecturer *or* Teaching*) has one or more* Courses *(or* Teachings *or* Lecturers *respectively).*

The label LT *on the edge between* Teaching *and* Year *indicates that* Year *is a single valued attribute of the relationship type* LT.

Finally, Teaching *and* Prereq *are reference objects and their information are captured in their referenced objects (i.e.* Course *in this case).*

ORA-SS captures significantly more semantic information of underlying XML database applications. In this thesis, we highlight two kinds of important semantic information that can be captured in ORA-SS, but not in DTD or other schema languages.

- Object class v.s. attribute: Data can be represented in XML documents either as attributes or elements. So, it is difficult to tell from the XML document whether an element is in fact an object or attribute of some object. DTD and other schema languages cannot specify whether an element represents an object in the real world or is an attribute of some object.

  For example, from the DTD graph in Figure 3.3, it is difficult to tell Lecturer is an object class while Hobby is not an object class, but an attribute of Lecturer object class.

- Attribute of object class vs. attribute of relationship type: As DTD and and other schema languages do not have the concept of object classes and relationship types (they only represent the hierarchical structure of elements and attributes), there is no way to specify whether an attribute is the attribute of one object class or the attribute of some relationship type.

  For example, Year is considered as an attribute of *LT* relationship between Lecturer and Teaching. However, from the DTD graph in Figure 3.3, it is difficult to tell whether Year is an attribute of the relationship between Lecturer and Teaching or Teaching object class. Such information is important for result display for XML keyword search which we will discuss in Chapter 5.

While there are other kinds of semantic information in ORA-SS that DTD and other schema languages cannot capture, with focus on keyword search, we will discuss the importance of the above two kinds of semantic information in keyword search result display in Chapter 5.

3.3 Dewey labeling scheme

In most contexts, a labeling scheme is adopted to assign a numerical label to uniquely identify each node in an XML tree structure. In this thesis, we adopt Dewey number labeling scheme since it is can easily identify the Lowest Common Ancestor (LCA) between two given Dewey labels which is important for XML keyword search.

With Dewey labeling, each node is assigned a list of components to represent the path from the document's root to the node. Each component along the path represents the absolute order of an ancestor node within its siblings; and each path uniquely identifies the absolute position of the node within the document.

For example, the Dewey numbers are shown with their corresponding XML nodes (except text values) in Figure 1.1.

In the following, we present the properties of Dewey numbers in determining the relationship between two given XML nodes n_1 and n_2 with different Dewey number d_1 and d_2 respectively.

- **Ancestor-Descendant (A-D) relationship:** n_1 is an ancestor of n_2 if and only if d_1 is a proper prefix of d_2; meanwhile if n_1 is an ancestor of n_2, then n_2 is a descendant of n_1.

- **Parent-Child (P-C) relationship:** n_1 is a parent of n_2 if and only if d_1

is a prefix of d_2 and the length of d_1 is that of d_2 minus 1; meanwhile if n_1 is a parent of n_2, then n_2 is a child of n_1.

- **Siblings relationship:** n_1 and n_2 are siblings if and only if d_1 and d_2 only differ in the last component.
- **Document order**[1]**:** n_1 proceeds n_2 if and only if d_1 proceeds d_2 in lexicographical order.
- **LCA:** the LCA of n_1 and n_2 is the node with Dewey number which is the longest common prefix of d_1 and d_2.

Example 4 *In the XML data of Figure 3.1 and its tree (with ID references) representation in Figure 1.1, based on the above properties of Dewey numbers, we can conclude Courses:0.1 is an ancestor of Title:0.1.1.1; Course:0.1.1 is the parent of Title:0.1.1.1; the LCA of id:0.1.1.0 and Title:0.1.1.1 is Course:0.1.1. Finally, id:0.1.1.0 and Title:0.1.1.1 are siblings while id:0.1.1.0 proceeds Title:0.1.1.1 in document order.*

Note Dewey labels effectively capture the root to descendant paths in XML data. However, Dewey labels do not reflect ID reference information. We will discuss in Chapter 4 how such information is captured with the *connection table*.

3.4 Importance of ID references in XML

Foreign key reference has well-recognized importance in Relational databases. Its equivalence in XML databases, ID reference, is also defined in DTD and many other schema languages. In many XML databases, ID references are present and

[1]Document order represents the order of appearance of elements in XML document.

play an important role in eliminating redundancies and representing relationships between XML elements, especially when an XML database contains several types of real world entities and wants to capture their relationships. For example, in Figure 1.1, references indicate important *teaching* relationships between Lecturer and Course elements. Without ID references, the relationships have to be expressed in further nested structures (e.g. each lecturer is nested and duplicated in each course she/he teaches or vice versa), potentially introducing harmful redundancies. Thus, we believe ID references are usually present and important in XML databases which capture relationships among real world objects.

3.5 Tree + IDREF data model

Due to the hierarchical structure and the existence of ID references in XML databases, we model XML as special digraphs, *Tree + IDREF*, G=(N, E, E_{ref}), where N is a set of nodes, E is a set of tree edges, and E_{ref} is a set of ID reference edges between two nodes. Each node $n \in N$ corresponds to an XML element, attribute or text value. Each tree edge denotes a parent-child (nested) relationship. We denote a reference edge from u to v as $(u,v) \in E_{ref}$. In this way, we distinguish the tree edges from reference edges in XML. The subgraph $T = (N, E)$ of G without ID reference edges, E_{ref}, is a tree. When we talk about parent-child (P-C) and ancestor-descendant (A-D) relationships between two nodes in N, we only consider tree edges in E of T.

For example, we have seen the Tree + IDREF model representation in Figure 1.1 for the XML data of Figure 3.1.

With Tree + IDREF data model, we are able to capture important ID references in XML databases, which are missed in Tree data model for XML key-

word search. Meanwhile, our model distinguishes tree edges from IDREF edges in XML. In this way, we are able to leverage the efficiency benefit of the tree model (especially in finding node connections based on LCAs with Dewey labeling scheme) and significantly reduce the amount of expensive computations in finding node connections in graphs.

Chapter 4

XML Keyword Search with ID References

In this part, we first formally introduce novel *Lowest Referred Ancestor (LRA) pair, Extended LRA (ELRA) pair* and *Extended LRA (ELRA) group* semantics for XML keyword proximity search in *Tree+IDREF* data model to overcome the limitations of SLCA in the tree model. Then, we address the generality and applicability of the newly proposed semantics, followed by the algorithms to compute results based on our approach.

Since most of the examples in this chapters are based on Figure 1.1, we make a copy of this figure in this chapter as Figure 4.1 for easy reference.

4.1 Existing SLCA semantics

Smallest Lowest Common Ancestor (SLCA) semantics has been widely studied and accepted ([35, 42, 46]) as an efficient approach for XML keyword search in the tree data model. Now, we first review the concept of SLCA.

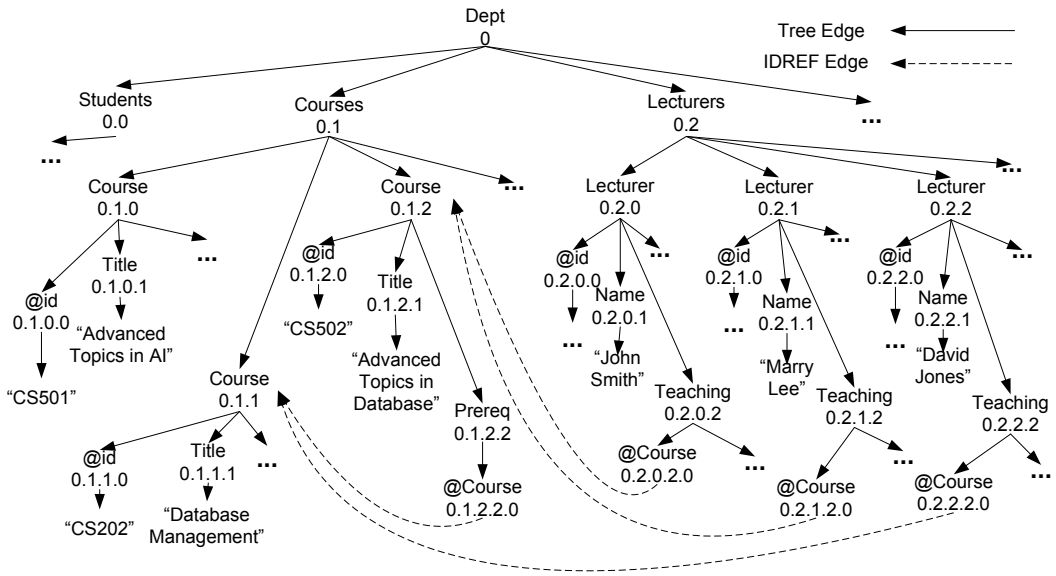


Figure 4.1: Example XML document of computer science department with Dewey labels (Copy of Figure 1.1)

Definition 1 *(**SLCA**) In an XML document, SLCA semantics of a set of keywords K returns a set of nodes such that each node u in the set covers all keywords in K, but no single proper descendant of u covers all keywords in K.*

Example 5 *In Figure 4.1, node Course:0.1.2 is the SLCA result for query "CS502 Advanced Database".*

However, given the importance of ID references in many XML databases, SLCA in the tree data model is not sufficient to meet all search requirements.

Example 6 *Consider keyword query "Advanced Database Smith" that probably looks for whether Smith teaches the specified course. In this case, the SLCA result is the meaningless overwhelming root Dept:0 (whole document) in Figure 4.1.*

An immediate solution to the above problem is to identify a set of overwhelming nodes at system setup phase and exclude these nodes from SLCA results. Overwhelming nodes can be identified by setting a threshold for the fanout

and/or size (i.e. number of descendants and/or bytes). Schema information can also be helpful to define overwhelming nodes (i.e. tags in DTD that have no *-annotated ancestor tags are likely to be overwhelming). For example, nodes Students, Courses, Lecturers and root Dept in Figure 4.1 can be identified as overwhelming nodes. However, exclusion of overwhelming nodes from SLCA results will generate no results in the above example (for query "Advanced Database Smith").

Note, we believe it is better to return *no result* in the case where the SLCA result is overwhelming, especially for huge databases. This is because overwhelming results can waste users' significant amount of effort in going through a huge ocean of information, most of which is likely irrelevant; while "no result" at least saves such efforts. Therefore, in the rest of this thesis, we assume overwhelming nodes are excluded from SLCA results.

One may further suggest using OR logic instead of AND to connect query keywords. Unfortunately, it still includes many irrelevant answers such as course "Advanced Topics in AI" and Lecturers named "Smith" who have no relationship with Advanced Database courses.

4.2 Proposed search semantics with ID references

4.2.1 LRA semantics

In this part, we introduce *Lowest Referred Ancestor pair (LRA pair)* semantics to exploit ID references for keyword proximity search in XML. Before that, we first define *reference-connection* that is important for *LRA pair* semantics.

Definition 2 *(**reference-connection**) Two nodes u, v with no A-D relationship in an XML database have a reference-connection (or are reference-connected) if there is an ID reference between u or u's descendant and v or v's descendant.*

Example 7 *There is a reference-connection between nodes Lecturer:0.2.0 and Course:0.1.2 in Figure 4.1 since there is an ID reference edge between their descendants (i.e. @Course:0.2.0.2.0 and @id:0.1.2.0)*[1].

Note the definition of reference-connection does not include the directions of ID reference edges since our focus is on whether or not two nodes are connected. However, directions can be enforced and displayed in the result output.

Keen readers may have noticed that some reference-connection are not very meaningful according to the definition of reference-connection. For example, Courses:0.1 and Lecturers:0.2 are also considered to have a reference-connection according to Definition 2, which, however, is not concise enough (i.e. indicating some lecturers teach some courses) as a meaningful connection. There are several ways to identify and exclude reference-connections that are not concise enough from meaningful connections. First, when we can identify overwhelming nodes, we can exclude reference-connections that involve overwhelming nodes from meaningful reference-connection. Or second, when the semantic information of ORA-SS model exists, we can restrict the set of XML nodes that may have meaningful reference-connections to the set of nodes that are considered as object classes or attributes of some object classes. For example, since Courses:0.1 and Lecturers:0.2 of the XML data in Figure 4.1 are neither considered as object classes nor attributes in ORA-SS model of Figure 3.4, we can exclude reference-connections that involve Courses:0.1 or Lecturers:0.2 from meaningful reference-

[1]In figure 4.1, since attribute id is the identifier of Course element, we show the reference from @Course nodes to Course nodes for simplicity.

connections. In the rest of the thesis, when we say reference-connection, we refer to meaningful reference-connections.

Now, we are ready to define LRA pair semantics for a list of keywords K.

Definition 3 *(**LRA pair**) In an XML database, LRA pair semantics of a list of keywords K returns a set of unordered node pairs $\{(u_1, v_1)$,(u_2, v_2),..., $(u_m, v_m)\}$ such that for any (u_i, v_i) in the set,*

(1) u_i and v_i each covers some and together cover all keywords in K; and

(2) there is a reference-connection between u_i and v_i; and

(3) there is no proper descendant u' of u_i (or v' of v_i) such that u' forms a pair with v_i (or v' forms a pair with u_i resp.) to satisfy conditions (1) and (2).

Intuitively, a pair of nodes (and their subtrees) form an LRA pair if they are connected by reference-connection and they are the lowest to together cover all keywords.

Example 8 *Consider keyword query "Smith Advanced Database" in Figure 4.1. Reference-connected Lecturer:0.2.0 and Course:0.1.2 form an LRA pair for this query, indicating Smith teaches the course; while the SLCA is the overwhelming root.*

We can see from above example, compared to SLCA, LRA pair semantics has a better chance to find smaller sub-structures, which is generally assumed better in most XML keyword proximity search approaches (e.g. [23, 30, 35, 46], etc).

4.2.2 ELRA pair semantics

In this part, we extend the reference-connection in LRA pair semantics to a chain of connections as *n-hop-connection* in *Extended LRA pair (ELRA pair)* semantics.

Definition 4 *(**n-hop-connection**) Two nodes u, v with no A-D relationship in an XML database have an n-hop-connection (or are n-hop-connected) if there are $n-1$ distinct intermediate nodes $w_1, ...w_{n-1}$ with no A-D pairs in $w_1, ...w_{n-1}$ such that $u, w_1, ..., w_{n-1}, v$ form a chain of connected nodes by reference-connection.*

Example 9 *In Figure 4.1, Lecturer:0.2.0 and Lecturer:0.2.1 are connected by a 2-hop-connection via node Course:0.1.2, which means the two lecturers teach the same course.*

Similarly, Lecturer:0.2.0 and Course:0.1.1 are connected by a 2-hop-connection via node Course:0.1.2, indicating Course:0.1.1 is a prerequisite of the course (i.e. Course:0.1.2) that Lecturer:0.2.0 teaches.

Definition 5 *(**ELRA pair**) In an XML database, ELRA pair semantics of a list of keywords K returns a set of unordered node pairs $\{(u_1, v_1)$,(u_2, v_2),..., $(u_m, v_m)\}$ such that for any (u_i, v_i) in the set,*

(1) u_i and v_i each covers some and together cover all keywords in K; and

(2) there is an n-hop-connection between u_i and v_i; and

(3) there is no proper descendant u' of u_i (or v' of v_i) such that u' forms a pair with v_i (or v' forms a pair with u_i resp.) to satisfy conditions (1) and (2).

Intuitively, ELRA pair semantics returns a set of pairs such that each pair are two lowest n-hop-connected nodes to together cover all keywords. When the length of the connection chain grows, we can potentially find more ELRA pairs at the cost of longer response time due to larger search space. However, the relevance between the nodes in each pair potentially becomes weaker in general as the chain grows longer. Thus, the system can first compute ELRA pairs whose connection chains are not longer than a default limit of L intermediate hops (i.e. n-hop-connection with $n \leq L$). Then if users are interested in more results, the

system can progressively increase the limit to find more results for users upon request. The value of n-hop-connection length limit can be set by users for each query or (by default) determined at the system tuning phase in the way that the execution time will not exceed users' time budget for a set of testing queries.

Therefore, we present the following L-limited ELRA pair semantics when the limit of n-hop-connection length is set to L.

Definition 6 *(**L-limited ELRA pair**) In an XML database, L-limited ELRA pair semantics of a list of keywords K returns a set of unordered node pairs $\{(u_1, v_1),(u_2, v_2),..., (u_m, v_m)\}$ such that for any (u_i, v_i) in the set,*

(1) u_i and v_i form an ELRA pair for K; and

(2) there is an n-hop-connection between u_i and v_i for an upper limit L of the connection chain length.

In the following, when we say ELRA pair, we mean L-limited ELRA pair with a tuned upper limit L for n-hop-connection length.

Example 10 *In Figure 4.1, for keyword query "Smith Lee", Lecturer:0.2.0 and Lecturer:0.2.1 (connected by a 2-hop-connection via node Course:0.1.2) form an ELRA pair if the limit of n-hop-connection chain length is set greater than or equal to 2. This ELRA pair result can be understood as Smith and Lee teach the same course. On the other hand, the SLCA result is the overwhelming node Lecturers:0.2 including all lecturers; while LRA pair semantic cannot find results for this query.*

Similarly, for keyword query "Smith Database Management", Lecturer:0.2.0 and Course:0.1.1 (connected by a 2-hop-connection via node Course:0.1.2) form an ELRA pair result, indicating Database Management is a prerequisite of the course

that Smith teaches. On the other hand, the SLCA result is the overwhelming root node Dept:0; while LRA pair semantic cannot find results for this query.

We can see from this example that ELRA pair semantics has a better chance to find more and/or smaller results than SLCA and LRA pair semantics since ELRA pair semantics is a more general case of LRA pair semantics. Note LRA pairs are the lowest pairs with direct reference-connection (or 1-hop-connection) while ELRA pairs are the lowest pairs with connections up to a tuned limit (L) intermediate hops including reference-connection. Therefore, ELRA pair semantics can effectively replace LRA pair semantics with a tuned limit L to ensure the query evaluation is within time budget.

It is interesting that there may be multiple n-hop-connections between two nodes. However, it is sufficient to find one existence of n-hop-connection instead of the "best" n-hop-connection during query processing since our focus is on the connected nodes that cover all keywords. In case users are also interested in the connections between a particular result pair, the system can compute a set of their connections to show different relationships between the pair upon request.

4.2.3 ELRA group semantics

Finally, we extend ELRA pair semantics to ELRA group semantics to define relationships among two or more connected nodes that together cover all keywords.

Definition 7 *(**ELRA group**) In an XML database, ELRA group semantics of a list of keywords K returns a set of node-group patterns $\{(h_1\text{-}G_1),(h_2\text{-}G_2),\ldots,(h_m\text{-}G_m)\}$ s.t. for each node-group pattern h_i-G_i $(1 \leq i \leq m)$,*

(1) each node in G_i covers some keywords and nodes in h_i and G_i together cover all keywords in K; and

(2) h_i connects all nodes in G_i by n-hop-connection; we call h_i the hub for G_i; and

(3) there are no proper descendants of any node u in G_i to replace u to cover the same set of keywords as u and are n-hop-connected ($n \leq L'$) to the hub; and

(4) there is no proper descendant d of G_i's hub h_i such that d is the hub of another ELRA group G_d and $(G_d \cup \{h_i\}) \supseteq G_i$.

Intuitively, ELRA group semantics returns a group of nodes which are connected to a common hub node such that all nodes in the group are the lowest to contain a subset of query keywords and the hub is also the lowest to connect this group of nodes.

Similar to ELRA pair semantics, we can choose a default value as the upper limit L' for n-hop-connection chain length in ELRA group semantics at the system tuning phase (L' is usually smaller than L which is the upper limit of chain length in ELRA pair semantics); and L' can be set or increased upon a user's request.

Therefore, we present the following L'-limited ELRA group semantics when the limit of n-hop-connection length is set to L'.

Definition 8 *(L'-**limited ELRA group**) In an XML database, L'-limited ELRA group semantics of a list of keywords K returns a set of node-group patterns $\{(h_1$-$G_1),(h_2$-$G_2),\ldots,(h_m$-$G_m)\}$ s.t. for each node-group pattern h_i-G_i ($1 \leq i \leq m$),*

(1) h_i connects all nodes in G_i by n-hop-connection with up to L' as the number of intermediate hops; and

(2) G_i form an ELRA group for K with h_i as the hub.

With the tuned limit L' for ELRA group semantics, the distances between any two nodes in one ELRA group are effectively restricted to not more than $(L' * 2)$ hops away. Similar to ELRA pair semantics, when we say ELRA group

semantics in the following, we refer to ELRA group with tuned upper limit L' of n-hop-connections.

Compared to SLCA and ELRA pair, ELRA group semantics can potentially find more and smaller connected nodes that cover some query keywords in the result.

Example 11 *Consider keyword query "Lee Smith Database Management" in Figure 4.1. With L' set as one, the node group Course:0.1.1, Lecturer:0.2.0 and Lecturer:0.2.1 form an ELRA group result with node Course:0.1.2 being the hub, indicating Lee and Smith teach the same database course and "Database Management" course is a prerequisite of their course. On the other hand, SLCA returns the root and ELRA pairs have no result for this query.*

4.2.4 Generality and applicability of the proposed semantics

Up to now, we have illustrated the benefit of exploiting ID references with proposed semantics compared to SLCA based on the particular example of Figure 4.1. Given the fact that ID references are important in XML to indicate the relationships between real world entities, we believe these semantics are applicable to many XML keyword search applications whose underlying XML database contains ID references since ID reference connected nodes are usually related to each other. In the following, we address the generality and applicability of the proposed semantics based on two of the most-cited XML benchmark datasets: DBLP [32] and XMark [41].

Figure 4.2 shows a part of the DTD graph for DBLP bibliography XML database. The main structure of DBLP is a list of papers; and each ID reference

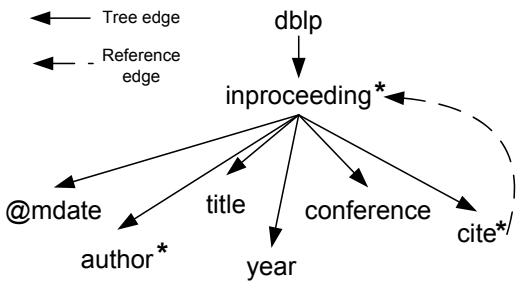


Figure 4.2: DBLP DTD graph (partial)

indicates one citation relationship between two papers.

In this case, given a keyword query in DBLP, LRA pair (with reference-connection) semantics can be used to find a paper that does not cover all query keywords, but citing or cited by another paper such that they together cover all keywords; ELRA pair (with 2-hop-connection) can be used to find two papers that together cover all keywords and citing and/or cited by some common paper. These papers (or paper pairs) can be good complementary results if users want more query related papers besides those SLCA results containing all query keywords. Note due to the citation relationships, it is reasonable to speculate these connected results are usually more relevant than results based on SLCA with keyword disjunction without considering ID references.

Consider, for instance, the query "XML querying processing". LRA pair semantics is able to find "query processing" papers that do not cover "XML" in the title, but citing or cited by XML papers. These "query processing" papers are usually more relevant to the query than other "query processing" papers not citing or cited by XML papers.

Similarly, in XMark auction XML data whose DTD graph sketch is shown in Figure 4.3, the information for persons, items and auctions is maintained separately and ID reference from an auction to a person (or an item) indicates the person attended (or the item is bidden for in) the auction. In this case, for

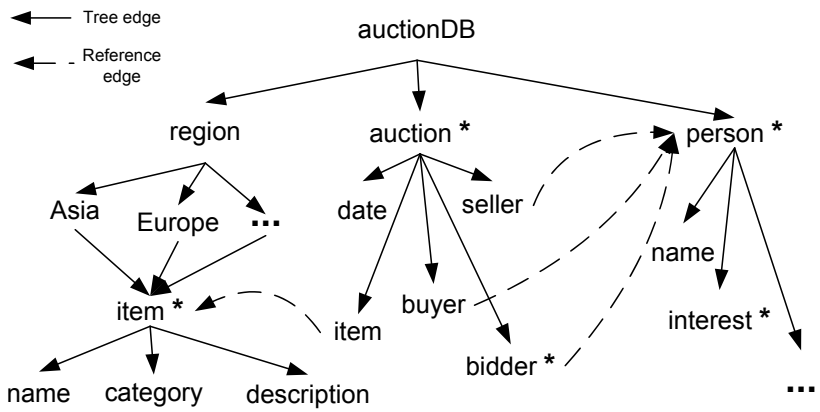


Figure 4.3: XMark DTD graph (partial)

a keyword query of a person name and an item name, ELRA pair semantics is able to find person and item pairs to see if the person attended some auction for the item. Also, for a keyword query of a person name, an item name and an auction date, ELRA group semantics is able to find out if the person attended the auction on the particular date for the item.

4.3 Algorithms for proposed search semantics

This section presents the data structures and two algorithms: sequential-lookup and rarest-lookup algorithms to find keyword search answers for the proposed semantics.

4.3.1 Data structures

The two data structures that we adopt in this paper are *keyword inverted lists* and *connection table*.

Keyword inverted lists are standard structures for keyword search applications. Each keyword inverted list stores the Dewey labels of all the parent nodes that directly contain the keyword in our approach. Moreover, an index (e.g.

B+-Tree) is built on top of each inverted list. Since inverted lists are standard structures for keyword search, we mainly discuss the connection table in the following.

The connection table maintains one connection-list, List(u), for each node u in the XML document such that List(u) contains all the lowest nodes (v) that have reference-connection (i.e. 1-hop-connection) to u in document order. From the Dewey label of v, we can easily get all v's ancestors that are not ancestors of u so that they are also reference-connected to u. Indexes can be built on top of the connection table to facilitate efficient retrieval of the connection list for a given node.

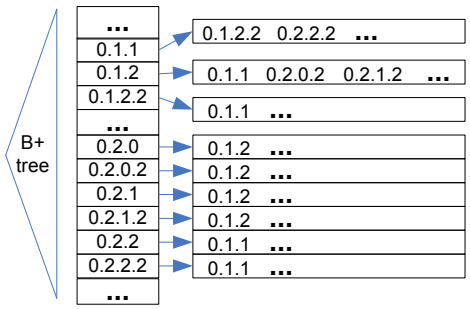


Figure 4.4: The Connection Table of the XML tree in Figure 4.1

For example, Figure 4.4 shows the B+-tree indexed connection table for the XML data in Figure 4.1. In Figure 4.4, we can see node 0.1.1 has reference connection to 0.1.2.2[2], thus we can tell that 0.1.1 is also reference-connected to 0.1.2. Note we do not keep the direction of ID references in the connection table. However, such information can be easily captured with one more bit for each node to indicate whether the direction of ID reference is incoming or outgoing.

The size of the connection table in the worst case is $O(|D| * |ID|)$, where $|D|$ and $|ID|$ are the number of nodes and IDs in an XML tree. However, the size is

[2]Note we ignore the reference connection between 0.1.1 and 0.1.2.2.0 in the connection table for simplicity since @Course:0.1.2.2.0 is an IDREF typed attribute of element Teaching:0.1.2.2.

usually much smaller than the worst case upper bound in most applications.

Note the connection table is similar to adjacency list representations of graphs. The only exception is that if u is reference-connected to v, then we should also keep in the connection table that u's ancestors a's are also reference-connected to v for those a's that are not v's ancestors. Therefore, we can follow standard graph traversal algorithms based on IDREFs to extract the connection table from XML documents, with some care of the above mentioned exception. Similarly, to compute u's n-hop-connected nodes with L as the tuned upper limit of connection chain length, we can do a depth-limited search (limited to L) from u based on the connection table with special care that if u is n-hop-connected to v, then u is also n-hop-connected to v's ancestors that are not u's ancestors.

For static XML data (or dynamic data where most updates are insertions), we can even pre-compute and store all n-hop-connected nodes ($n \leq L$ for some tuned L) for each node for faster query response. However, in this thesis we compute n-hop-connections during query processing for generality.

4.3.2 Naive algorithms for ELRA pair and group

In this part, we present the naive algorithms, called *sequential-lookup* algorithms, to compute all search results for ELRA pair and ELRA group semantics. The sequential-lookup algorithm for ELRA pair semantics, $\mathsf{ComputeELRA\_P}_{seq}$, is presented in Algorithm 1; while the sequential-lookup algorithm to compute ELRA groups is named $\mathsf{ComputeELRA\_G}_{seq}$ in Algorithm 3. Note we can get all LRA pairs by setting the limit L of n-hop-connection length in ELRA pairs as one. Therefore, we omit the algorithm for LRA pair semantics.

Algorithm computeELRA_P$_{seq}$

Now, we present the naive sequential-lookup algorithm, computeELRA_P$_{seq}$, to compute ELRA pair results in Algorithm 1. Its main idea is to check each node n and n's ancestors in all query keywords' inverted lists in document order to see whether they and their connected nodes can contribute to ELRA pair results.

The input parameters of Algorithm computeELRA_P$_{seq}$ include inverted lists of each individual query keyword I_1,I_2,...,I_k, the connection table CT and tuned upper limit of n-hop-connection length L for ELRA pairs. It sort-merges[3] I_1,I_2,...,I_k into I_{seq} and scans each node in I_{seq} and its ancestors to check if they and their n-hop-connected ($n \leq L$) nodes can form ELRA pair results; and returns all ELRA pair results upon completion.

Algorithm 1: computeELRA_P$_{seq}$

```
   Input: Keyword lists I1,I2,...,Ik, connection table CT, the upper limits L
   Output: ELRA_P
 1 initial empty ELRA_P ;     //mapping each u to ∀v s.t. u&v ∈ ELRA pair
 2 let I_seq be the sort-merged list of I1,I2,...,Ik;     // sort-merge can also be done on the fly
 3 for (each self-or-ancestors u of each node in I_seq in top-down order) do
 4 |   get I_i, ..., I_m whose keywords u does not cover ;
 5 |   if (u ∉ ELRA_P and u does not cover all keywords) then
 6 |   |   Q=getConnectedList(u, CT, L) ;
 7 |   |   remove ∀q ∈ Q from Q s.t. u proceeds q in document order ;
 8 |   |   S_u=computeSLCA(Q,I_i,...,I_m);     // adopt existing algorithms for SLCA
 9 |   |   remove ∀v ∈ S_u from S_u s.t. v covers all keywords ;
10 |   |   ELRA_P.put(u, S_u);
11 |   |   for (∀a s.t. a is ancestor of u and a ∈ ELRA_P) do
12 |   |   |   S_a = ELRA_P.get(a);
13 |   |   |   S_a = S_a - S_u;    // set difference
14 |   |   |   ELRA_P.update(a, S_a);
15 |   |   end
16 |   end
17 end
18 return ELRA_P;
```

The details of Algorithm computeELRA_P$_{seq}$ are as follows. It sequentially scans Dewey labels and their ancestors in the sort-merged list (I_{seq}) in top-down

[3]Sort-merge can also be done on the fly.

Function `getConnectedList(`u, CT, L`)`

1 return the list of lowest nodes computed by depth-limited search from CT that have n-hop-connection ($n \leq L$) to u in document order ;

(i.e. ancestor to descendant) order (line 3). For each currently scanned Dewey label u, we check whether u covers some but not all keywords (by probing the indexed inverted list of each keyword). If so , we find i) the keywords and their inverted lists $I_i, ..., I_m$ that u does not cover in line 4 and ii) all u's lowest n-hop-connected nodes (with chain length $n \leq L$) Q by calling Function getConnectedList (line 6) which we will discuss shortly (we also defer the discussion of line 7 which is mainly for efficiency purposes). Then in line 8, we find the set (S_u) such that each node v in S_u is a self-or-ancestor of some node in Q and v is the smallest node to cover the remaining keywords with inverted lists $I_i, ..., I_m$. This step is achieved by performing an SLCA operation for the lists of Q and $I_i, ..., I_m$. Now, each node v in S_u may potentially form an ELRA pair with u if we cannot find a descendant of u to form a lower connected pair with v to cover all keywords later on. So, we temporarily put u and S_u in result ELRA_P (line 10). Finally, we use S_u to prune the false positives of u's ancestor a's lowest connected nodes in S_a (lines 11-15) that each together with a covers all keywords. The reason is if some node $v \in S_u$ forms a lower pair with u to cover all keywords, it cannot form an ELRA pair with u's ancestor a according to the definition of ELRA pair semantics.

Function getConnectedList takes a node Dewey ID u, the connection table CT and the tuned upper limit of chain length L as inputs and returns all n-hop-connected ($n \leq L$) nodes for u. As mentioned in Section 4.3.1, to compute u's n-hop-connected nodes with L as the tuned upper limit of connection chain length, we can do a depth-limited search (limited to L) from u based on the

connection table with a special care that if u is n-hop-connected to v, then u is also n-hop-connected to v's ancestors that are not u's ancestors. Therefore, we omit the detailed pseudo code in the function.

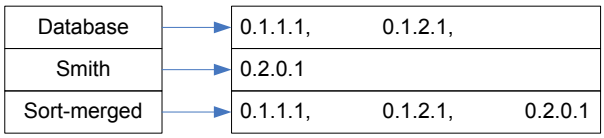
Now, we come to line 7 which is simply for efficiency purposes to make Q smaller as the input to Function computeSLCA. Now, assume nodes u and v form an ELRA pair and u proceeds v in document order. Then, we will first encounter u during the sequential scan and get u, v as an ELRA pair. After this, sequential scan will also encounter v and if we do not remove u from v's connected list in line 7, we will waste computation in getting the pair twice and removing duplicate results.

Note in line 8, given a number of existing algorithms for SLCA semantics, in this thesis, we currently adopt the Index Lookup Eager algorithm in [46] which is simple yet reasonably efficient since our focus is not on computing SLCAs. Other SLCA algorithms, such as Stack Algorithm [35, 46] and Multiway-SLCA [42] can be easily incorporated in our approach to replace Index Lookup Eager algorithm when necessary.

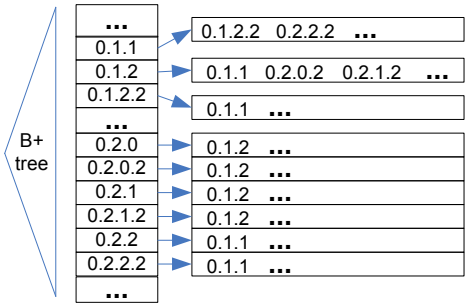
The following example shows a trace of Algorithm computeELRA_P$_{seq}$ for keyword query "Database Smith" in the XML database of Figure 4.1, with upper limit of n-hop-connection set to two (i.e. $n \leq L = 2$). Note the SLCA result of this query is the overwhelming root (or none if overwhelming nodes are removed from results).

Example 12 *Figure 4.5 (a) shows the inverted lists for keywords "Database", "Smith" and the sort-merged list; Figure 4.5 (b) shows part of the connection table for the XML database in Figure 4.1.*

The first node in the sort-merged list is 0.1.1.1. Following Algorithm computeELRA_P$_{seq}$, we scan all self-or-ancestors of 0.1.1.1 in top-down order. Since



(a)Inverted lists of keywords "Database", "Smith" and their sort-merged list



(b) The Connection Table of the XML tree in Figure 4.1 (copy of Figure 4.4)

Figure 4.5: Data structures used in processing query "Database Smith"

node 0 and 0.1 are overwhelming and excluded from results, we start with 0.1.1 which covers "Database". From Figure 4.5 (b), 0.1.1 is reference-connected to 0.1.2.2 and 0.2.2.2. Therefore, 0.1.1 is also reference-connected to 0.1.2 and 0.2.2 since they are ancestors of 0.1.2.2 and 0.2.2.2 respectively according to the definition of reference-connection. Note 0.1.2 is further reference-connected to 0.1.1, 0.2.0.2 and 0.2.1.2. As a result, 0.1.1 is 2-hop-connected to 0.2.0.2, 0.2.1.2 and their ancestors via node 0.1.2 (but 0.1.1 is not considered 2-hop-connected to 0.1.1 itself). Therefore, we conclude 0.1.1 is n-hop-connected ($n \leq L = 2$) to 0.1.2.2 (1-hop), 0.2.0.2 (2-hop via 0.1.2), 0.2.1.2 (2-hop via 0.1.2), 0.2.2.2 (1-hop) and their corresponding ancestors. After performing the SLCA operation between the n-hop-connection ($n \leq L = 2$) list of 0.1.1 and the inverted list of "Smith", we find the lowest connected node of 0.1.1 that covers the remaining keyword "Smith" is 0.2.0 via a 2-hop-connection. So, 0.1.1 and 0.2.0 together cover all keywords and are put into ELRA pair candidates. Next, we move on to 0.1.1.1 to check for ELRA pairs, which has no results since 0.1.1.1 is not reference-connected to any

node.

The second node in the sort-merged list is 0.1.2.1 which covers "Database". Its ancestor 0.1.2's n-hop-connected ($n \leq L = 2$) list includes 0.1.1 (1-hop) (removed by line 7 of Algorithm computeELRA_P$_{seq}$), 0.2.0.2 (1-hop), 0.2.1.2 (1-hop) and 0.2.2.2 (2-hop via 0.1.1). So, we can find another pair 0.1.2 and 0.2.0 with 1-hop-connection to cover all keywords. Since 0.1.2 is not a descendant of existing candidate pair 0.1.1 and 0.2.0, no false positive can be found after getting 0.1.2 and 0.2.0.

Finally, the third node in the sort-merged list is 0.2.0.1. Its ancestor 0.2.0 is n-hop-connected ($n \leq L = 2$) to 0.1.1.0, 0.1.2.0 and 0.2.1.2. The first two are removed by line 7 since they proceed 0.2.0.1 in document order. Therefore, no more ELRA pair candidates can be found.

At this stage, pair 0.1.1 and 0.2.0 via a 2-hop-connection and 0.1.2 and pair 0.2.0 with 1-hop-connection are returned as ELRA pair results.

Algorithm computeELRA_G$_{seq}$

Now, we present the sequential-lookup algorithm to compute ELRA group results, computeELRA_G$_{seq}$, in Algorithm 3. Its main idea is to check each node n in all query keywords' inverted lists in document order and n's ancestors to see whether they and their connected nodes can be a hub to connect a group of nodes to cover all query keywords in order to be an ELRA group result.

The input parameters of Algorithm computeELRA_G$_{seq}$ include inverted lists of each individual query keywords I_1,I_2,...,I_k, the connection table CT and the tuned upper limit of n-hop-connection length L' for ELRA groups. It sort-merges I_1,I_2,...,I_k into I_{seq} and scans each node in I_{seq} and its ancestors to check if they and their connected nodes can be a hub to connect a group of nodes to cover all

query keywords; and returns all ELRA group results upon completion.

Algorithm 3: computeELRA_G$_{seq}$

```
   Input: Keyword lists I1,I2,...,Ik, the connection table CT, the upper limits L'
   Output: ELRA_G
 1 initial empty ELRA_G ;     // mapping each u to a group G s.t. u is a hub to
                                 // connect ∀v ∈ G as an ELRA group
 2 let Iseq be the sort-merged list of I1,I2,...,Ik;     // sort-merge can also be done on the fly
 3 for (each self-or-ancestors u of each node in Iseq in top-down order) do
 4     G=getELRAGroup(u, I1,I2,...,Ik, CT, L')
 5     if G ≠ null then
 6         LRA_G.put(u, G) ;
 7     end
 8     for (∀a s.t. a is ancestor of u and a ∈ ELRA_G) do
 9         if (G ∪ {a} ⊇ ELRA_G.get(a)) then
10             ELRA_G.remove(a) ;
11         end
12     end
13     Q=getConnectedList(u, CT, L') ;
14     for (each self-or-ancestors q of each node in Q in top-down order) do
15         G=getELRAGroup(q, I1,I2,...,Ik, CT, L') ;
16         if (G ≠ null) then
17             ELRA_G.put(q, G) ;
18         end
19         for (∀a s.t. a is ancestor of u and a ∈ ELRA_G) do
20             if (G ∪ {a} ⊇ ELRA_G.get(a)) then
21                 ELRA_G.remove(a) ;
22             end
23         end
24     end
25 end
26 return ELRA_G ;
```

The details of Algorithm computeELRA_G$_{seq}$ are as follows. For each Dewey ID in I_{seq} and its ancestors (u) in top-down order (line 3), we check if u can be a hub to form an ELRA group G by calling Function getELRAGroup (line 4) which we will discuss shortly. After finding non-null group G, we check if u and G can prune away the groups hubbed by u's ancestors (lines 5–12). Then, we compute all u's n-hop-connected ($n \leq L'$) nodes in set Q (line 13) and check whether each node q in Q and q's ancestors can be hubs to form ELRA groups by calling Function getELRAGroup (line 15). Each time we find a new ELRA group with

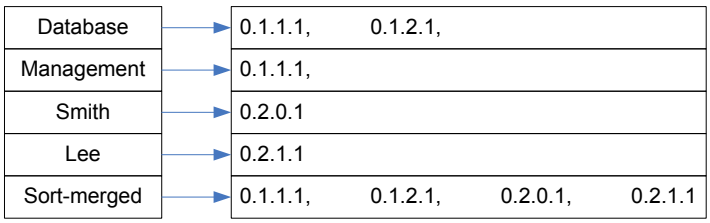
Function `getELRAGroup`(h, I_1,I_2,...,I_k, CT, L')

```
1  if (h cover all keywords) then
2  |  return null ;
3  end
4  Q = getConnectedList(h, CT, L') ;
5  initial empty set G ;
6  for (each I_i ∈ I_1, I_2, ..., I_k) do
7  |  Y = getSLCA(I_i, Q) ;
8  |  remove ∀y ∈ Y from Y s.t. y covers all keywords ;
9  |  if (Y is empty and h does not cover I_i's keyword) then
10 |  |  return null ;
11 |  end
12 |  G = G ∪ Y;
13 end
14 return G ;
```

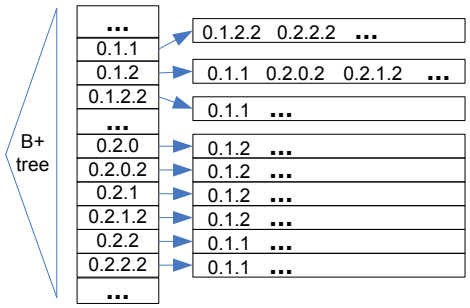
hub q, we check each existing group whose hub is q's ancestor to prune possible false positives (lines 19–23) according to the definition of ELRA group semantics.

Function getELRAGroup takes a node h, inverted lists of each individual query keywords I_1,I_2,...,I_k, the connection table CT and tuned upper limit of n-hop-connection length L' for ELRA group semantics as inputs; and returns a group of nodes G such that nodes in G form a candidate ELRA group with h as the hub (G is null if h cannot be a hub to form an ELRA group with n-hop-connections $n \leq L'$).

Function getELRAGroup first ensures input h is not a self-or-ancestor of SLCA (lines 1–3). Then, it gets all h's n-hop-connected ($n \leq L'$) nodes Q by calling Function getConnectedList (line 4). From line 6 to line 13, we get all h's n-hop-connected ($n \leq L'$) nodes that contain some query keywords. We achieve this by computing the SLCAs (Y) of list Q and inverted list of each query keyword in line 7. In line 8, we make sure each node in Y does not contain all query keywords. If the SLCA result (Y) is empty for a given keyword with inverted list I_i and h itself does not cover the corresponding keyword, then null is returned since h cannot be a hub to form an ELRA group (as all nodes that are n-hop-connected



(a)Inverted lists of keywords "Database", "Management", "Smith", "Lee" and their sort-merged list



(b) The Connection Table of the XML tree in Figure 4.1 (copy of Figure 4.4)

Figure 4.6: Data structures used in processing query "Database Management Smith Lee"

($n \leq L'$) to h including h do not cover the query keyword of I_i). Otherwise, if null is not returned in line 10 for all iterations, then all query keywords can be covered by some node with n-hop-connection ($n \leq L'$) to h. Therefore, a candidate ELRA group is found with h as the hub.

The following example shows a trace of Algorithm computeELRA_G$_{seq}$ for keyword query "Database Management Smith Lee" in the XML database of Figure 4.1, with upper limit of n-hop-connection set to one (i.e. $n \leq L' = 1$).

Example 13 *Figure 4.6 (a) shows the inverted lists for keywords "Database", "Management", "Smith", "Lee" and the sort-merged list; Figure 4.6 (b) shows part of the connection table for the XML database in Figure 4.1.*

The first node in the sort-merged list is 0.1.1.1. Following the function, we scan all self-or-ancestors of 0.1.1.1 in top-down order. Since node 0 and 0.1 are

overwhelming and excluded from results, we start testing whether 0.1.1 can be a hub to form an ELRA group, which is n-hop-connected ($n \leq L' = 1$) to 0.1.2.2, 0.2.2.2 and their corresponding ancestors. After performing SLCA operations based on the connected list and each keyword inverted list, we will not get meaningfully connected nodes to cover all query keywords. Thus, 0.1.1 cannot be a hub to form an ELRA group with n-hop-connection ($n \leq L' = 1$).

Next, Algorithm computeELRA_G_{seq} checks whether nodes in 0.1.1's connected list can be hubs to form candidate ELRA groups. The first connected node is 0.1.2.2. We first test its ancestor 0.1.2 for ELRA group hub. 0.1.2 is n-hop-connected ($n \leq L' = 1$) to 0.1.1, 0.2.0.2 and 0.2.1.2. After performing SLCA operations based on the connected list and each keyword inverted list, we will find nodes 0.1.1, 0.2.0 and 0.2.1 form an ELRA group with 0.1.2 as the hub. Thus, a candidate ELRA group is found. After checking that 0.1.2.2 cannot form an ELRA group, the previous candidate ELRA group becomes a real result. The second connected node of 0.1.1 is 0.2.2.2, for which we cannot find an ELRA group.

Now, Algorithm computeELRA_G_{seq} moves on to scan subsequent nodes in the sort-merged list and their connected lists to check for more candidate ELRA groups and prune false positives according to the definition of ELRA group semantics.

Finally, node 0.1.2 is returned as a hub to form an ELRA group since this group is not identified as false positives.

4.3.3 Rarest-lookup algorithms for ELRA pair and group semantics

The naive algorithm is expensive when the number of query keywords grows, since it sequentially scans all nodes in all keywords' inverted lists to check for ELRA pair and group results. In fact, it is sufficient to only check the shortest (rarest) inverted list for all results to significantly reduce the amount of computations, based on the following lemma.

Lemma 1 *Every ELRA pair (or ELRA group) must include at least one node (or its ancestor) from the shortest (rarest) inverted list of query keywords.*

Therefore, we propose rarest-lookup algorithms to compute ELRA pairs and groups, which are presented in Algorithm 5 computeELRA_P$_{rare}$ and Algorithm 6 computeELRA_G$_{rare}$ respectively.

Algorithm 5: computeELRA_P$_{rare}$

```
   Input: Keyword lists I1,I2,...,Ik, the connection table CT, the upper limits L
   Output: ELRA_P
 1 initial empty ELRA_P ;      //mapping each u to ∀v s.t. u&v ∈ ELRA pair
 2 let I_rarest be the rarest (shortest) list of I1,I2,...,Ik;
 3 for (each self-or-ancestors u of each node in I_rarest in top-down order) do
 4 |   get I_i, ..., I_m whose keywords u does not cover ;
 5 |   if (u ∉ ELRA_P and u does not cover all keywords) then
 6 |   |   Q=getConnectedList(u, CT, L) ;
 7 |   |   S_u=computeSLCA(Q,I_i,...,I_m);
 8 |   |   remove ∀v ∈ S_u from S_u s.t. v covers all keywords ;
 9 |   |   ELRA_P.put(u, S_u);
10 |   |   for (∀a s.t. a is ancestor of u and a ∈ ELRA_P) do
11 |   |   |   S_a = ELRA_P.get(a);
12 |   |   |   S_a = S_a - S_u;    // set difference
13 |   |   |   ELRA_P.update(a, S_a);
14 |   |   end
15 |   end
16 end
17 return ELRA_P;
```

Algorithm 6: computeELRA_G$_{rare}$

```
   Input: Keyword lists I1,I2,...,Ik, the connection table CT, the upper limits L'
   Output: ELRA_G
 1 initial empty ELRA_G ;      // mapping each u to a group G s.t. u is a hub to
                                  // connect ∀v ∈ G as an ELRA group
 2 let I_rarest be the rarest (shortest) list of I1,I2,...,Ik;
 3 for (each self-or-ancestors u of each node in I_rarest in top-down order) do
 4 |   G=getELRAGroup(u, I1,I2,...,Ik, CT, L') ;
 5 |   if G ≠ null then
 6 |   |   LRA_G.put(u, G) ;
 7 |   end
 8 |   for (∀a s.t. a is ancestor of u and a ∈ ELRA_G) do
 9 |   |   if (G ∪ {a} ⊇ ELRA_G.get(a)) then
10 |   |   |   ELRA_G.remove(a) ;
11 |   |   end
12 |   end
13 |   Q=getConnectedList(u, CT, L') ;
14 |   for (each self-or-ancestors q of each node in Q in top-down order) do
15 |   |   G=getELRAGroup(q, I1,I2,...,Ik, CT, L') ;
16 |   |   if (G ≠ null) then
17 |   |   |   ELRA_G.put(q, G) ;
18 |   |   end
19 |   |   for (∀a s.t. a is ancestor of u and a ∈ ELRA_G) do
20 |   |   |   if (G ∪ {a} ⊇ ELRA_G.get(a)) then
21 |   |   |   |   ELRA_G.remove(a) ;
22 |   |   |   end
23 |   |   end
24 |   end
25 end
26 return ELRA_G ;
```

Since Algorithm 5 computeELRA_P$_{rare}$ and Algorithm 6 computeELRA_G$_{rare}$ share great similarity with Algorithm 1 computeELRA_P$_{seq}$ and Algorithm 3 computeELRA_G$_{seq}$ respectively, we only highlight their differences in this part, omitting detailed explanations and examples for brevity.

The only changes from computeELRA_P$_{seq}$ in Algorithm 1 to computeELRA_P$_{rare}$ are in line 2 and line 7 of computeELRA_P$_{seq}$. In line 2, instead of getting the sort-merged list of all query keyword inverted lists, we choose the shortest (rarest) inverted list. For line 7, we need to remove it from computeELRA_P$_{seq}$ for computeELRA_P$_{rare}$. The reason is simply because we only scan the rarest inverted list

now, instead of scanning nodes in all inverted lists. Therefore, we need to make sure all connected nodes of each node in the shortest inverted list are checked for potential ELRA pair results.

Similarly, the only change from computeELRA_G$_{seq}$ in Algorithm 3 to computeELRA_G$_{rare}$ is in line 2. Instead of getting the sort-merged list of all query keyword inverted lists, we choose the shortest (rarest) inverted list.

4.3.4 Time complexity analysis

In the following, we present the analysis of time complexities of our algorithms.

Lemma 2 *The time complexities of naive sequential-lookup algorithms to compute ELRA pairs and ELRA groups are the following:*

- *Algorithm 1* computeELRA_P$_{seq}$ *for ELRA pair semantics:*

  $O(d\sum_{i=1}^{k}|N_i|(|E_L|+kd|Q_L|\log|N_{max}|))$*, and,*

- *Algorithm 3* computeELRA_G$_{seq}$ *for ELRA group semantics:*

  $O(|Q_{L'}|d^2\sum_{i=1}^{k}|N_i|(|E_{L'}|+kd|Q_{L'}|\log|N_{max}|))$

where k is the number of keywords; d is the maximum depth of the XML documents; $|N_{min}|$, $|N_{max}|$ and $|N_i|$ are the sizes of shortest, longest and i^{th} inverted lists in the query respectively; E_L and Q_L are the maximum number of edges and nodes reached by depth-limited search with chain length limit L for ELRA pair semantics; and finally $E_{L'}$ and $Q_{L'}$ are the maximum number of edges and nodes reached by depth-limited search with chain length limit L' for ELRA groups.

PROOF SKETCH:

We first derive the complexity of Algorithm 1 computeELRA_P$_{seq}$ *for ELRA pair semantics. The factor,* $d\sum_{i=1}^{k}|N_i|$*, in the complexity is simply due to the* for *statement in line 2 of Algorithm 1* computeELRA_P$_{seq}$*. The other factor,* $(|E_L|+kd|Q_L|\log|N_{max}|)$*, represents the complexity of each iteration (i.e. lines 3–15) inside the for statement. The two most significant operations in terms of big-O notation are line 5 and line 7. The complexity of line 5 is* $|E_L|+|Q_L|$*, which is for depth-limited search. The complexity for line 7 (SLCA) is* $kd|Q_L|\log|N_{max}|$[4]*. Thus, the sum of the two lines in terms of big-O is* $(|E_L|+kd|Q_L|\log|N_{max}|)$*. Therefore, the complexity for ELRA pair semantics is derived. Note the complexity of nested* for *loop (lines 10–14) is* $d|Q_L|$ *since the set different operation can be done in linear time given* S_a *and* S_u *are sorted in document order. This complexity for nested* for *loop is less significant than that of line 7.*

Now, we derive the complexity of Algorithm 3 computeELRA_G$_{seq}$*. First, the complexity of Function* getELRAGroup *is* $O(|E_{L'}|+kd|Q_{L'}|\log|N_{max}|)$*. The reason is the most significant operations in Function* getELRAGroup *are line 4 (i.e.* $|E_{L'}|+|Q_{L'}|$*) and iterated line 7, which is the product of* k *(due to loop) and the cost of SLCA for* I_i *and* Q *(i.e.* $2*d|Q_{L'}|\log|I_i|$*).*

Next, in line 14 of Algorithm 3 computeELRA_G$_{seq}$*,* getELRAGroup *is called* $|Q_{L'}|d^2\sum_{i=1}^{k}|N_i|$ *times in total due to the nested loops in line 2 and line 13. Therefore, the complexity of Algorithm 3* computeELRA_G$_{seq}$ *is the product of* $|Q_{L'}|d^2\sum_{i=1}^{k}|N_i|$ *and* $(|E_{L'}|+kd|Q_{L'}|\log|N_{max}|)$ *in big-O notation.*

Lemma 3 *The time complexities for rarest-lookup algorithms to compute ELRA pairs and groups are the following:*

[4]Since we adopt Index Lookup Eager algorithm in [46], the complexity for SLCA ($kd|Q_L|\log|N_{max}|$) directly comes from [46]. Interested readers may refer to [46] for the analysis, while we focus on using the analysis result of [46] to derive the complexity in computing our ELRA pair and ELRA group.

- *Algorithm 5* computeELRA_P_{rare} *for ELRA pair semantics:*

  $O(d|N_{min}|(|E_L| + kd|Q_L| \log |N_{max}|))$*, and,*

- *Algorithm 6* computeELRA_G_{rare} *for ELRA group semantics:*

  $O(|Q_{L'}|d^2|N_{min}|(|E_{L'}| + kd|Q_{L'}| \log |N_{max}|))$

where the variables are the same as those in sequential-lookup's complexity.

PROOF SKETCH: *Algorithm 5 and Algorithm 6 are similar to Algorithm 1 and Algorithm 3, except we use the rarest inverted list instead of the sort-merged list in Algorithm 5 and Algorithm 6. Therefore, we can simply substitute* $\sum_{i=1}^{k} |N_i|$ *in sequential-lookup algorithms by* $|N_{min}|$ *for the complexities of rarest-lookup algorithms.*

Chapter 5

Result Display with ORA-SS and DBLP Demo

Semantic information of the underlying XML database is important for result display of XML keyword search. In this chapter, we discuss some guidelines for result display based on object classes and relationship types in ORA-SS. Then, we present our keyword search demo system, ICRA, that provides keyword search services in DBLP bibliography.

Note the discussion for result display in XSeek [36] also uses the concept of object classes. However, it exploits neither ID references in XML nor relationships between (among) objects.

5.1 Result display with ORA-SS

In the following, we will discuss how ORA-SS can be used to interpret the meanings of keyword queries and present search results based on object classes and relationship types.

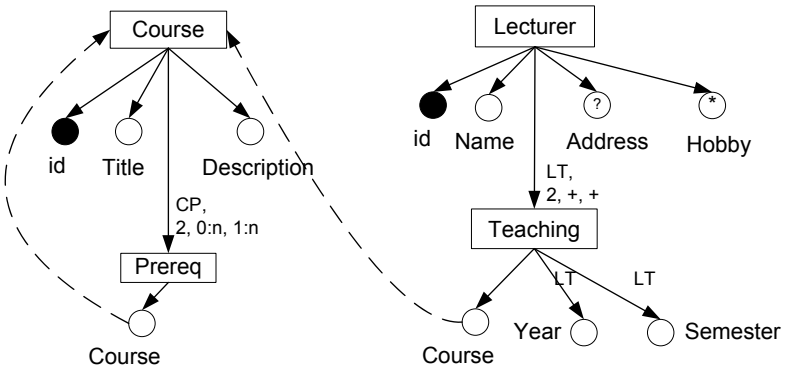


Figure 5.1: Example ORS-SS schema diagram fraction for the XML data in Figure 3.1 (Copy of Figure 3.4)

Note that all the examples in this section are based on the ORA-SS schema in Figure 3.4. For ease of reference, we show a copy of Figure 3.4 in Figure 5.1.

5.1.1 Interpreting keyword query based on object classes

Now, we discuss some guidelines of the interpretations of keyword queries based on object classes in ORA-SS model. This part focuses on the result display for SLCA.

First, the most common keyword queries are just a list of keywords which are values of object properties[1]. For these queries with only property values, we first compute the SLCA results. Then if the SLCA results are properties of objects, we should display all and only the information of the whole objects instead of just the keyword matching properties. For example, a user may search for a course via the course title with query "Database Management". In this case, the system should display the information of the course, including id (i.e. course-code), Title, Description, etc. However, the SLCA itself (i.e. Title) is not very informative as it is the same as the keyword query.

[1]In this chapter, we use *property* to refer to *attribute* of an object to avoid confusion with XML attributes

Second, sometimes users may want to define the output node as a keyword in the query. For example, a user can search for only id of a course via the Title with query "Database Management id". In this case, "id" is interpreted as output node since "id" matches a property name of Course object class. Therefore, we should output the course code (id) of "Database Management" instead of the whole object. Similarly, users can also use property name as a predicate for existence test, which we call *existential predicate*. For example, we can search for lecturers who have provided their address with query "Smith address". (i.e. search for lecturers with last name "Smith" and who have provided their addresses) or "Smith address Law Link" (i.e. search for lecturers with last name "Smith" and having a address in "Law Link"). In this case, the system will find the objects that contain the SLCAs of value "Smith" and address node (with value "Law Link" for the second query) to answer the keyword query.

Note it may be difficult to distinguish queries with existential predicates and queries with output nodes. For example, keyword "address" that matches a property name in query "Smith address" have the ambiguity of whether it should be interpreted as output node or existential predicate. We adopt the following rules to resolve the ambiguities.

- First, when keywords in a query match a property name and its value, then the system should interpret it as an existential predicate.

  For example, query "Name John Smith" should be interpreted as searching for lecturers with sname="John Smith" instead of searching for Name property of the matching lecturer.

- Second, if keywords of a query only match a property name that is mandatory in the object class or relationship type (i.e. the property appears at

least once in every object or relationship respectively) without matching the values of the property, intuitively, this keyword is an output node, instead of an existential predicate.

For example, since id is mandatory in Lecturer objects, the meaning of "id John Smith" is clear to search for id property of the matching lecturer.

- Finally, when keywords in a query match a property name that is optional in the object class or relationship type without matching the values of the property, then the system can regard this keyword as existential predicate to find all objects containing the node.

  For example, we will interpret query "Smith address" as searching for lecturers with last name "Smith" and having provided their addresses. The reason is users still can see the address of matching lecturers in case they want keyword "address" as output node.

Besides the above rules, we can also adopt simple syntax p to resolve ambiguities. We use "<>" to indicate output nodes, "[ ]" to indicate existential predicates and "N:k" (or "N:{k k' ...}") to indicate the containment of keyword k (or keywords k k'...) in a node N. For example, "Smith [address]" (or "Smith [address:{Law Link}]") means finding lecturers with last name "Smith" and having address (in Law Link); while "Smith <address>" means finding the address of "Smith".

5.1.2 Interpreting keyword query based on relationship-type

Relationship types in ORA-SS are also important to interpret keyword queries. Now, we extend the result display for ELRA pair and ELRA group. We start

with cases where each ELRA pair/group result matches a single relationship type in ORA-SS. Note when different ELRA pair/group results match different relationship types, we can display results in different categories.

First, if a keyword query matches a property of a relationship type, then the output of the query should be the whole relationship together with all the participating objects. It may not be correct and meaningful to return the SLCAs. For example, for query "Smith year:2007", the system should display the Lecturer-Teaching relationships including both Lecturer and Teaching objects with relationship property Year="2007", instead of just the subtree of corresponding SLCA Lecturer node. Note Teaching is a reference object, which means we should also include the information of the referenced object, i.e. Course, when we return Teaching. Similarly, for query "Database 2002", besides database courses that are taught in 2002, the system should also display the lecturers who taught Database in 2002. However, displaying only the LRA pair without the information of Lecturer may not be meaningful for this query.

Second, if a keyword query matches properties of two objects of two different object classes and there is a relationship type with respect to the two object classes, then the system should output the matching relationships. For example, for query "Database John Smith", the system should return the relationships involving the matching Course and Lecturer objects together with the relationship properties Year and Semester.

Third, if a keyword query matches the name of one object class and property values of an object for another object class and there is a relationship type with respect to the two object classes, then the system should output objects of the first object class together with the properties of relationships that involve the two object classes. For example, for query "John Smith <course>", the system should

return all courses and the corresponding years in which John Smith teaches the course.

The above guidelines are applicable when each ELRA pair/group result matches a single relationship type in ORA-SS. When an ELRA pair/group spans multiple relationship types, we can first extract result nodes in the pairs and groups. Then, we display these result nodes grouped by object classes with links to show their ELRA pairs and groups. In this way, each displayed result object contains some query keywords, but users have the choice to view the ELRA pairs and groups of each result.

Finally, grouping ELRA pairs/groups by participant nodes are also useful when the same node are duplicated in several ELRA pair and ELRA group results. For example, for query "Smith database", if Smith teaches multiple database courses, then the same Smith lecturer object will be duplicated in several ELRA pair results. Similarly, if the same database course is taught by several lecturers who share common name "Smith", then the same database course is also duplicated in different ELRA pair results[2]. In this case, we can group results by one type of node (object class) for clarity. For example, we can group results by lecturers for query "Smith database". Therefore, we will see all the database courses taught by each lecturer who has name Smith. It is up to the user to select which object class they want to group by while the system can choose a default one for the first display.

[2]This case may seem rare. However, for query "XML query processing" in DBLP, the same "XML" paper may cite or is cited by many "query processing" papers and meanwhile one "query processing" paper can cite or is cited by many "XML" papers

5.2 ICRA: online keyword search demo system

We apply our keyword search approach in the DBLP bibliography to provide keyword search services for the research community.

In this part, we demonstrate our ICRA online keyword search prototype. We will first present a simple briefing for the implementation of the demo system. Then we show the features of our system. The ICRA demo prototype is available at http://xmldb.ddns.comp.nus.edu.sg.

5.2.1 Briefing on implementation

Currently, we identify two object classes in DBLP, *publication* and *author*, for result display. Users can select one as their object class of interest for each query.

Search for publications

Given one keyword query when searching for publications, our system first computes the SLCA, ELRA pair and ELRA groups. SLCA results include all publications that each contains all query keywords; while ELRA pair (or group) results include all connected pairs (or groups) that each pair (or group) of publications contain all query keywords. We limit the length of the connection to 2 hops for ELRA pair and 1 hop for ELRA group so that papers citing or cited by some common papers are considered relevant.

Then, we group ELRA pair and ELRA group results by publications to avoid duplication. For clarity, we do not directly display the pairs and groups that each publication participates in, but provide links to show the pairs/groups of each paper for users to click.

Therefore, when publication is selected as the selected object class of interest,

the final result for each query is a list of publications.

We adopt the following simple rules for ranking purposes in our demo system. Anecdotal results of some sample queries proves the effectiveness of our ranking rules for DBLP. However, a general approach of result ranking in XML is left as future work.

- SLCA results are ranked before publications in ELRA pair and ELRA group results.
- In SLCA results, a publication with a property (i.e. author or title or conference/journal) that is fully specified in the keyword query is ranked higher than a publication without a fully specified property. For example, for query {Tian Yu}[3], a publication with one author whose name is Tian Yu is ranked before a publication with an author whose name is Tian-Li Yu.
- In ELRA pair and group results, a publication with more query keywords is ranked before a publication with less query keywords.
- In ELRA pair and group results, a publication that participates in more ELRA pairs is ranked higher than a publication participating in less ELRA pairs which is in turn ranked higher than a publication that participates in ELRA groups.

Search for authors

Given one keyword query when searching for authors, the system first computes the list of publications in the same way as searching for publications. Then, we

[3]In this chapter, we use curly bracket to enclose keyword queries.

extract all authors from these publications for result display. Since each author's name appears in each of his/her publications and there is no identifier for authors, we treat the same author names in different publications as the same person. How to distinguish different persons with the same name is beyond the scope of this thesis.

For ranking purposes, first, an author whose name is fully specified in the query is ranked higher than other authors. Also, we give higher rank to an author with more query relevant publications than another one with less.

5.2.2 Overview of demo features

Inspired by the simplicity of Google, our demo system provides a simple user interface for pure keyword queries; except users can specify if they are interested in publications (default) or authors in DBLP bibliography. We show the main user interface in Figure 5.2.

Search for publications

Our demo provides query flexibility in that users are free to issue keyword queries which can be any combination of words in full or partial author names, topics, conference/journal names and/or year.

In the following, we illustrate some ways to search for publications in our ICRA demo system. We do not demonstrate all the types of queries and reader are welcome to try other queries.

1. *An author name*: for example, users can input {Tian Yu} (or {Yu Tian} as Asian convention) to search for Tian Yu's publications. Our system automatically ranks Tian Yu's publications before other authors' whose names

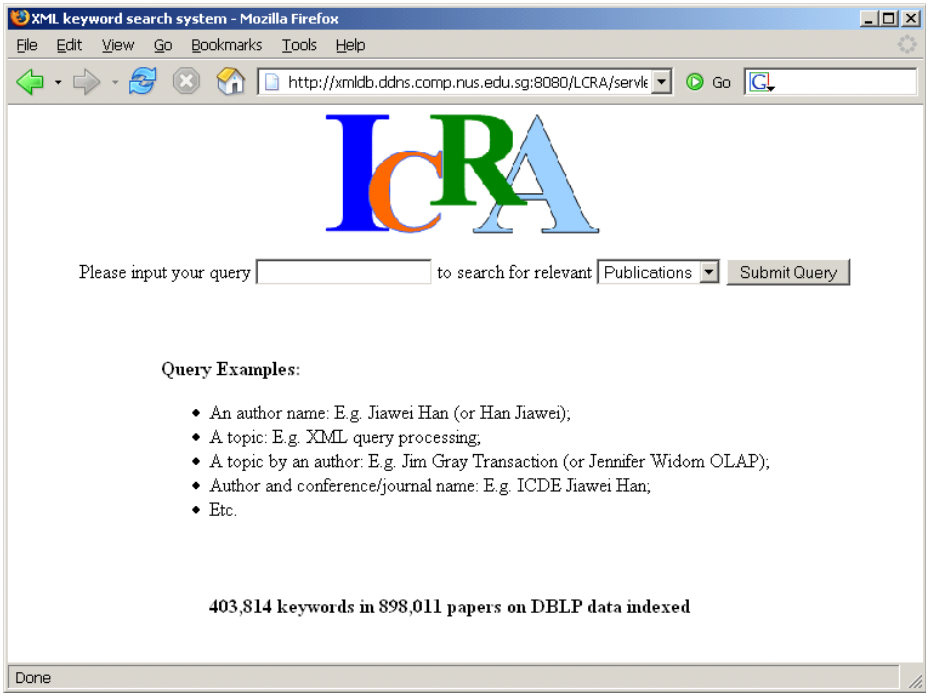


Figure 5.2: ICRA search engine user interface

include more than "Tian Yu" (e.g. Tian-Li Yu) as shown in Figure 5.3.

2. *Multiple author names*: we can search for co-authored papers with names of multiple authors.

3. *Topic*: we can input a topic to search for related publications. For example, we can query {XML query processing}. Besides papers containing all keywords, other related papers (e.g. "Query optimization for XML" written by Jason McHugh and Jennifer Widom) are also ranked and displayed according to our ELRA pair and ELRA group semantics).

4. *Topic by an author*: for example, we can query {Jim Gray transaction} for his publications related to transaction. Besides Jim Gray's papers containing "transaction", our system ranks his papers, which do not contain "transaction", but are "transaction" related, before his other papers due

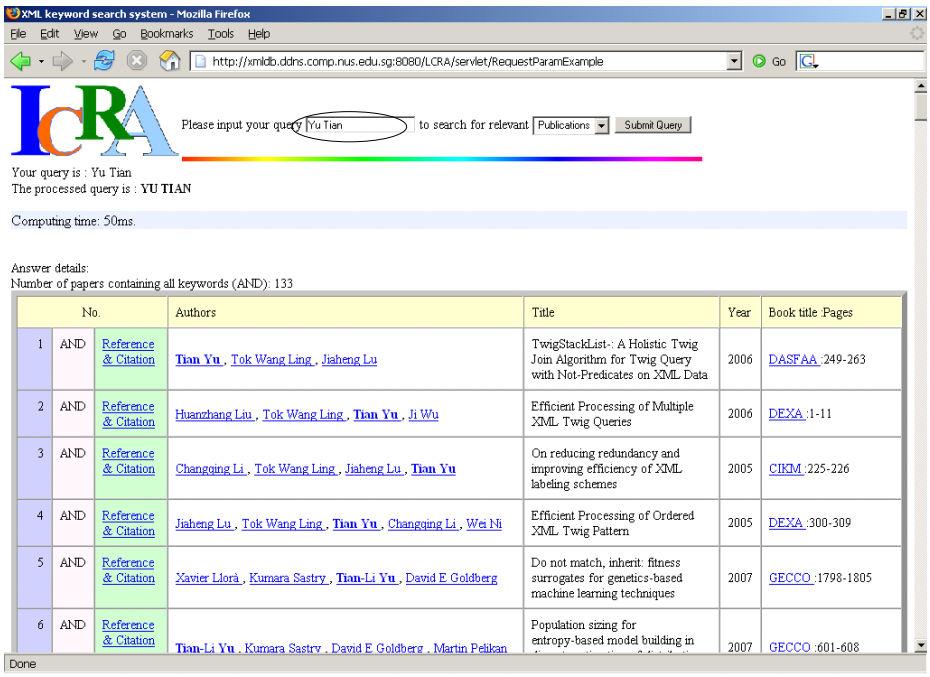


Figure 5.3: ICRA publication result screen for query {Yu Tian}

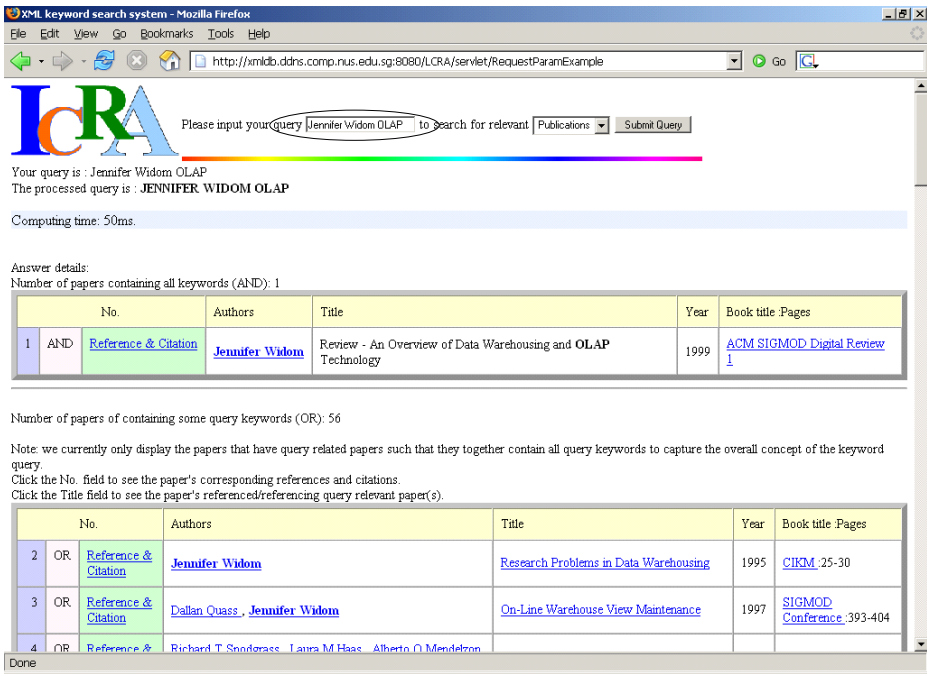


Figure 5.4: ICRA publication result screen for query {Jennifer Widom OLAP}

to the reference/citation relationships with transaction papers captured in ELRA pair and ELRA group semantics. Similarly, for query {Jennifer Widom OLAP}, our system is able to find her related papers whose titles do not contain "OLAP", but contain "data warehousing" since those papers cite or are cited by "OLAP" papers. A snapshot of the results for query {Jennifer Widom OLAP} is shown in Figure 5.4.

5. *Topic of a year*: for example, we can search for keyword search papers in 2007 with query {keyword search 2007}.

6. *Conference and author*: for example, we can search for Prof. Ooi Beng Chin's publications in ICDE with query {ICDE Beng Chin Ooi} as shown in Figure 5.5.

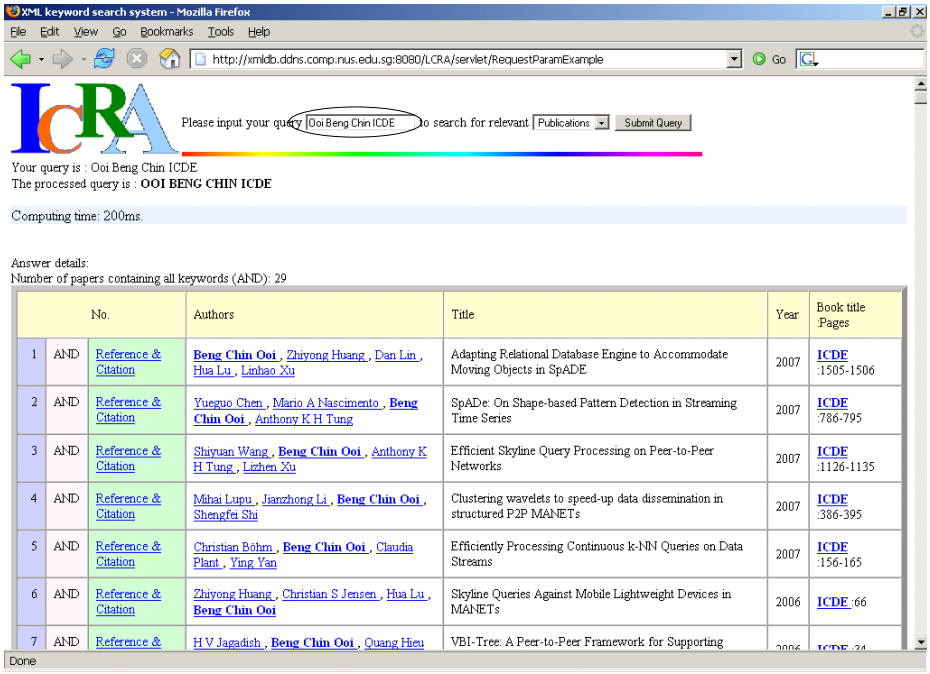


Figure 5.5: ICRA publication result screen for query {Ooi Beng Chin ICDE}

7. *Conference and year, author and year, etc.*

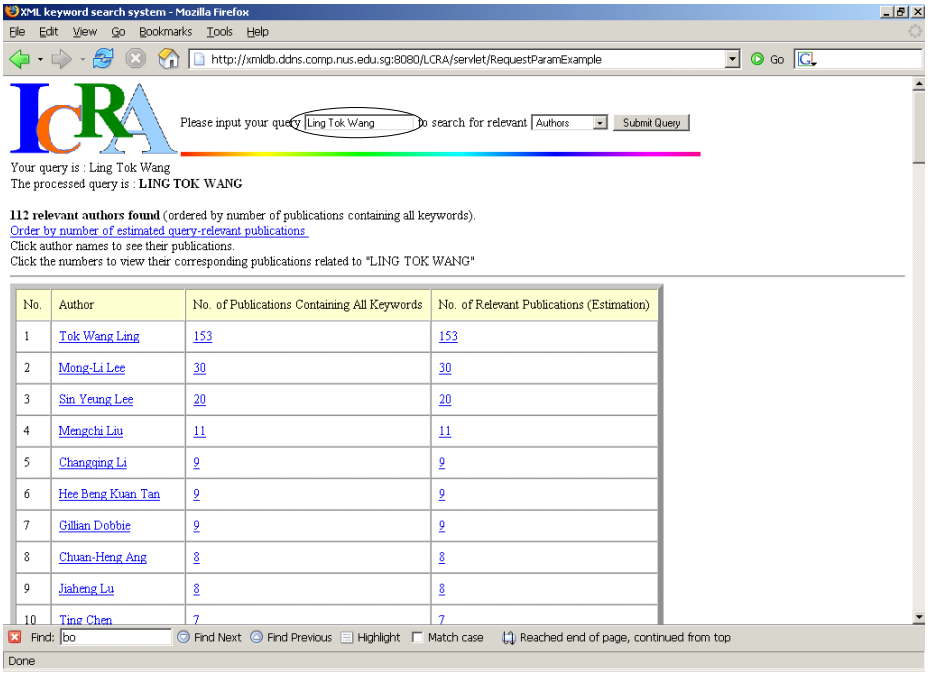


Figure 5.6: ICRA author result screen for query {Ling Tok Wang}

Search for authors

Users can also search for authors with a wide range of query types. In the following, we illustrate some ways to search for authors in our ICRA demo system. We do not demonstrate all the types of queries and reader are welcome to try other queries.

1. *Author name*: the most intuitive way to search for authors is to search by their names. In our system, besides the author with matching name, we also return his/her co-authors. For example, we can search for Prof. Ling Tok Wang's co-authors with query {Ling Tok Wang}; and some ICRA author results for this query are shown in Figure 5.6.

2. *Topic*: we can search for authors who have contributions to a topic. For example, we can input query {XML} for authors with most contributions

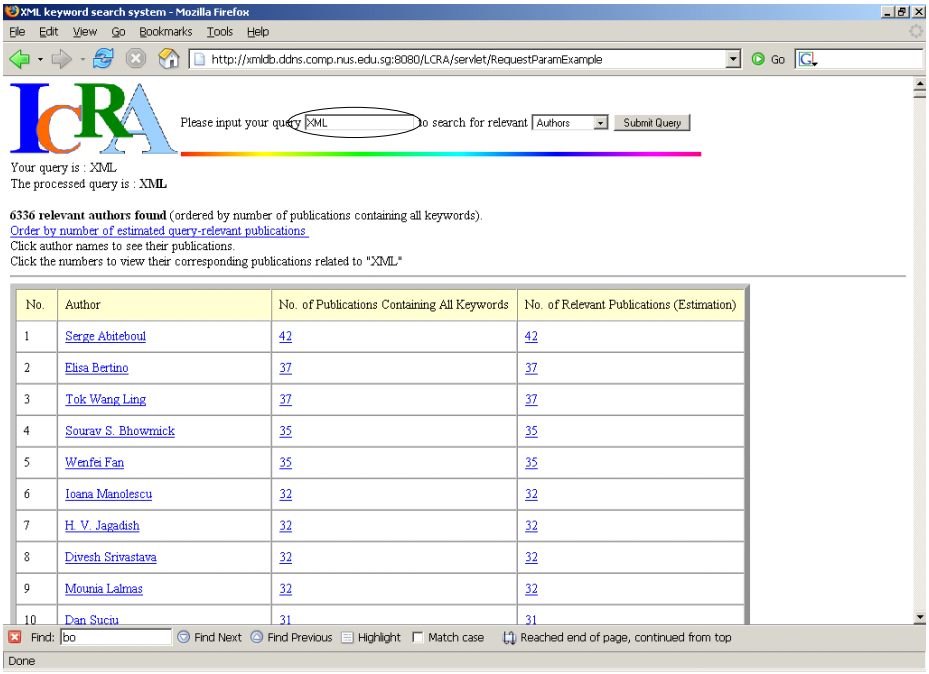


Figure 5.7: ICRA author result screen for query {XML}

to "XML" as shown in Figure 5.7.

3. *Conference/journal*: similar to topic, we can search for authors who are most active in a particular conference/journal. For example, Figure 5.8 shows the result of query {ICDE} to search for active authors in "ICDE".

4. *Author name and topic/conference/journal*: We can even search for one author's co-authors in a particular topic or conference/journal. For example, we can search for Surajit Chaudhuri's co-authors in ICDE with query {Surajit Chaudhuri ICDE} and the ICRA author results are shown in Figure 5.9.

Some readers may have noticed the two numbers displayed with each author result. They are for browsing purposes, which we will discuss shortly.

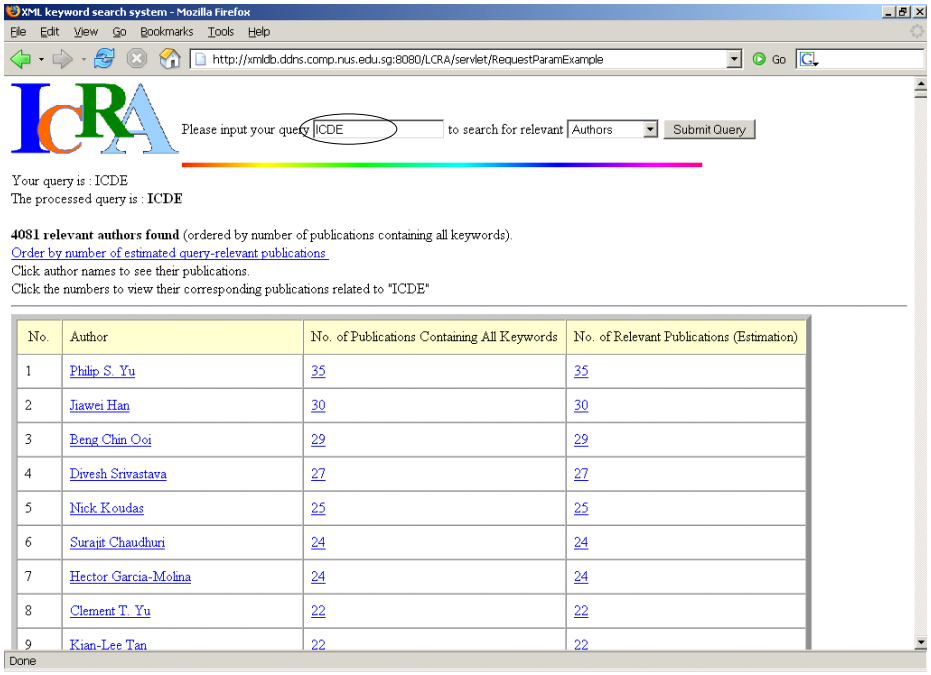


Figure 5.8: ICRA author result screen for query {ICDE}

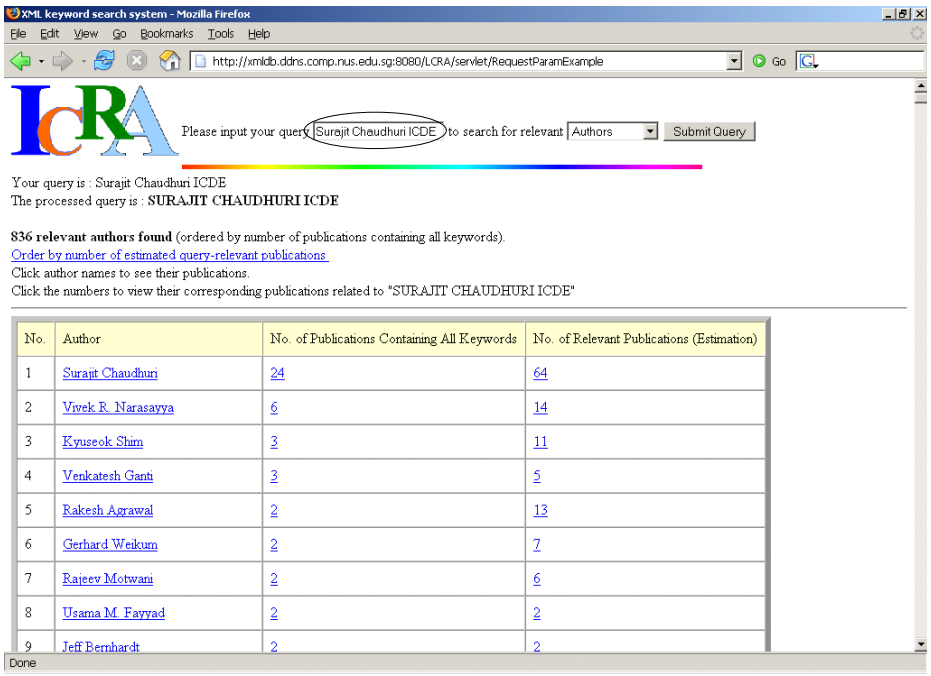


Figure 5.9: ICRA author result screen for query {Surajit Chaudhuri ICDE}

Note when we search for authors by name, since the system does not require users to specify the search intention of a keyword in the query (e.g. whether a keyword should be an author name or a part of a topic), the results also include other authors (e.g. most are co-authors since the co-authors' publications also contain the searched name) that do not match the name. However, we believe the inclusion of other authors usually does not affect the satisfaction of the user as long as the author matching the searched name is ranked in the top few (e.g. top-2 or 3) results. The reason is a query which searches an author by name is usually considered as *known item search*, meaning the searcher knows exactly what she needs. Thus, the user can simply stop reading other authors once she finds the searched author without being frustrated by a long list of results ranked after the searched author. And, importantly, our system is usually able to rank the author with matching name as the first result due to our ranking approach mentioned in Section 5.2.1.

Browsing

Besides searching, our system also supports browsing from search results to improve the practical usability. For example, users can click an author (or conference/journal) name in a result publication to see all publications of the author (or in the same proceeding/journal).

When searching for authors, we also output the number of publications containing all the query keywords and the number of publications (based on ELRA pair and group results) that may be relevant according to reference connections so that users can click the numbers to see the publications.

For example, when we search for authors with query {XML query processing}, author Daniela Florescu has 3 publications containing all the keywords and 7

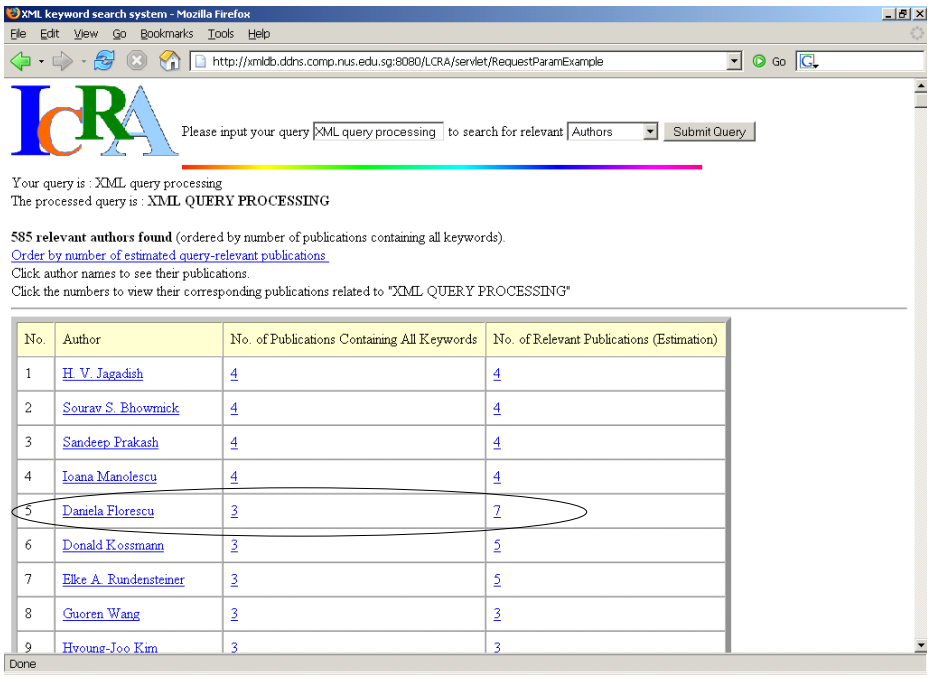


Figure 5.10: ICRA author result screen for query {XML query processing}

publications that may be related even though not all of them contain all keywords, which are shown in Figure 5.10. Note due to the incomplete information of citations in DBLP, our current estimation for relevant publications based on reference/citations may miss some relevant ones.

Chapter 6

Experimental Evaluation

6.1 Experimental settings

Hardware and implementation We use a PC with a Pentium 2.6GHz CPU and 1GB memory for our experiment. All codes are written in Java. In our experiments, we set the upper limit of connection chain length as two for ELRA pair and one for ELRA group. Results show these limits are reasonable for the tradeoff between execution time and result size.

Datasets and indexes creation We choose both real DBLP and synthetic XMark datasets in our experiment. They have been widely studied for measuring the efficiency of various XML keyword search applications(e.g. [6,30,46] etc.). The reality of DBLP also makes it possible to study the quality of search results for our demo system, which is available at http://xmldb.ddns.comp.nus.edu.sg.

Two datasets are pre-processed to create the inverted lists and the connection tables. They are stored in Disk with Berkeley DB [3] B+-trees and their entries are cached in memory only after the entries are used. The details of the file sizes and index creations of the two datasets are shown in Table 6.1. Note that the

Table 6.1: Data size, index size and index creation time

Data	File size	Keyword inverted lists		Connection table	
		creation time	size	creation time	size
DBLP	362.9MB	321 sec	145.7MB	81 sec	1.62MB
XMark	113.8MB	193 sec	140.3MB	234 sec	13.7MB

inverted lists of XMark has comparable size with DBLP's despite DBLP having a much larger file size. This is because each dewey ID in the inverted lists of DBLP is smaller due to its flat structure. Note that the connection table of DBLP is small due to the incomplete citation information of the data.

Queries and performance measures For each dataset, we generate random queries of 2 to 5 keywords long, with 50 queries for each query size. We use these random queries to compare the 1) efficiency of Sequential-lookup and Rarest-Look algorithms, 2) effectiveness of ELRA pair and group search semantics in Tree + IDREF model in terms of execution time and result size tradeoff as compared to SLCA alone, and 3) efficiency of computing ELRA pair/group as compared to Bi-Directional expansion heuristics in the general digraph model. We also use sample queries to measure the result quality in the real DBLP dataset. Our metric for result quality is the number of relevant answers among top-10, 20 and 30 results. Answer relevance is judged by discussions of a small group in our database lab, including volunteers.

6.2 Comparison of search efficiency based on random queries

6.2.1 Sequential-lookup v.s. Rarest-lookup

We present the efficiency comparisons between Sequential-lookup and Rarest-lookup in computing ELRA pair (Seq_P and $Rarest_P$) and ELRA group (Seq_G and $Rarest_G$) in Figure 6.1 for DBLP dataset and Figure 6.2 for XMark dataset respectively.

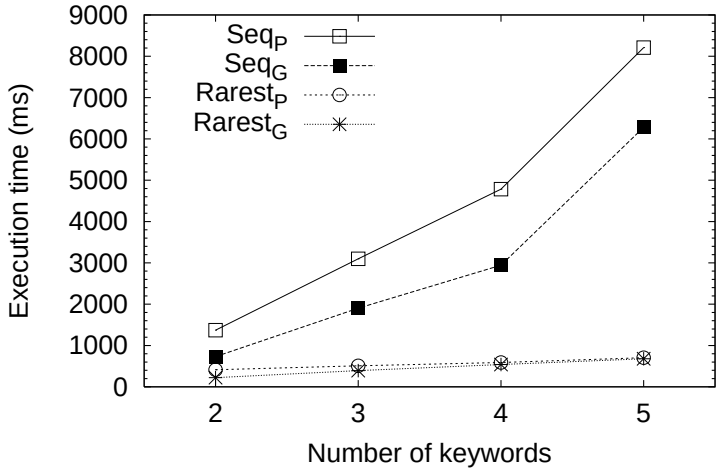


Figure 6.1: Time Comparisons between Rarest-lookup and Sequential-lookup in DBLP dataset

From Figure 6.1 and Figure 6.2, it is clear that Rarest-lookup achieves much better efficiency (up to 10 times faster for query size of five than Sequential-Lookup) in both datasets. Rarest-lookup is also more scalable to queries of more keywords since it only scans the shortest inverted lists, while Sequential-lookup goes significantly slower as the number of keywords increases. The reason is Sequential-lookup scans all keyword lists; thus it scans more and costs more when there are more keywords.

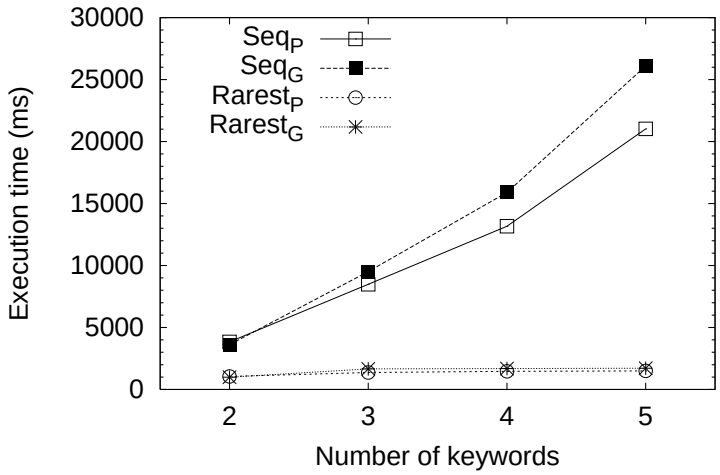


Figure 6.2: Time Comparisons between Rarest-lookup and Sequential-lookup in XMark dataset

We can also tell from the two figures that time spent in ELRA pair and group computation does not differ much for both algorithms in both datasets. The reason is ELRA pair and group semantics have about the same search space by setting chain length of ELRA pair and group semantics as two and one respectively (i.e. any two nodes in one result of either ELRA pair or ELRA group are not more than 2 hops away).

In Figure 6.2 for XMark dataset, time spent in ELRA pair semantics is slightly shorter than that in ELRA group, which conforms to our time complexity analysis. On the other hand, it is also interesting to see in Figure 6.1 that time spent in computing ELRA pair results is longer than that for ELRA group in DBLP although computation of ELRA groups has relatively larger theoretical time complexity. The reason is some papers in DBLP are connected to (cited by) many papers; thus depth-limited search from these papers for 2-hop-connections in ELRA pair semantics is costly. However, ELRA groups does not have this problem due to its 1-hop-connections in the experimental setting.

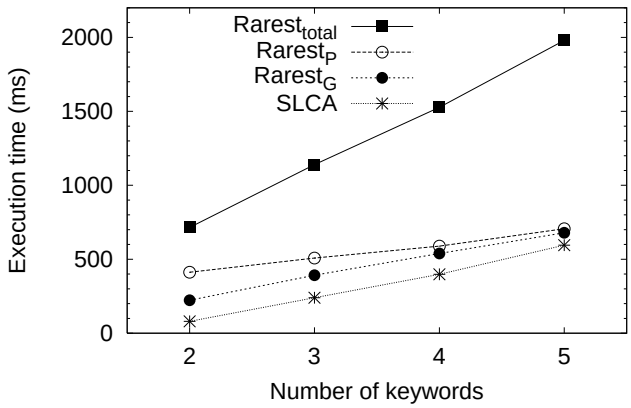


Figure 6.3: Time comparisons among SLCA, ELRA pair and group computation in DBLP dataset

6.2.2 Tree + IDREF v.s. tree data model

In this part, we compare our approach in Tree + IDREF with SLCA in the tree data model in terms of search efficiency and the total number of results returned. Since Rarest-Lookup outperforms Sequential-Lookup, we only show the efficiency of Rarest-Lookup algorithm for the comparison with SLCA in Figure 6.3 and Figure 6.4.

Note since we propose ELRA pair and group semantics as complements to SLCA results, we run SLCA followed by ELRA pair, which is in turn followed by ELRA group, to simulate real cases that system outputs SLCA first, followed by ELRA pair and group results. As a result, we also show the total time spent in all SLCA, ELRA pair and ELRA group semantics.

From Figure 6.3 and Figure 6.4, we can see the execution time for ELRA pair and ELRA group results are longer than the time for SLCA. This is expected since our approach needs to perform more computations to exploit ID references in XML.

For DBLP dataset in Figure 6.3, the computation for ELRA pair and ELRA

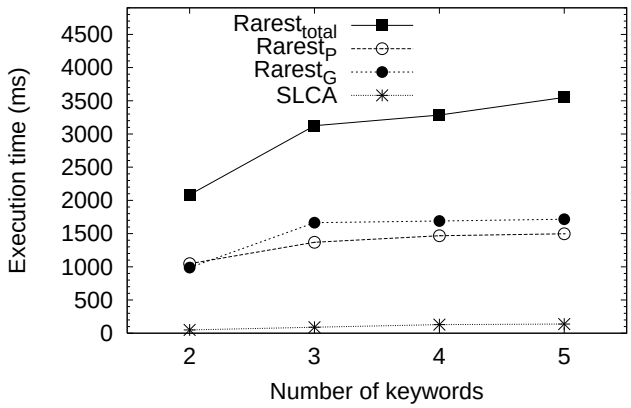


Figure 6.4: Time comparisons among SLCA, ELRA pair and group computation in XMark dataset

group results are 5 and 2.5 times slower than SLCA for queries of two keywords; while the differences become smaller to around 1.2 times slower for 5-keyword queries. The reason is the efforts in exploiting ID references for queries of 2 or 5 keywords are not significantly different in Rarest-Lookup to compute ELRA pair and group results; while the efforts of computing SLCA grows as the number of keywords increases since SLCA needs to probe more keyword inverted lists in Indexed Lookup Eager [46] that we use and potentially incur more disk accesses. Note that the computation of ELRA pair and group results also requires to probe keyword inverted lists. However, such probing can benefit from previous SLCA computation in the way that the inverted lists of query keywords are likely to be cached in memory. For XMark dataset in Figure 6.4, the experimental result is similar to DBLP's except that computation of ELRA pair and groups is further slower compared to SLCA computation. This is because there are more ID references in XMark than DBLP which slows down the computation of ID reference exploration.

Although ELRA pair and group semantics require more computation effort, the gain in more results does outweigh the cost as shown in Table 6.2 and Ta-

Table 6.2: Average result size for SLCA/ELRA pair/ELRA group of random queries in DBLP dataset

Keyword #	DBLP					
	SLCA	ELRA pair		ELRA group		Total
		pair#	node#	group#	node#	node#
2	86	597	174	142	174	260
3	6	83	56	94	288	294
4	3	14	11	44	227	230
5	3	4	4	24	267	270

ble 6.3. It is clear that ELRA pair and ELRA group results on top of SLCA can find significantly more results than SLCA alone (3-90 times more for DBLP and 40-1000 times more results for XMark in terms of total number of distinct result nodes). As discussed in previous chapters, these ID reference connected nodes are likely to be relevant to the keyword query. At least, our approach provides a good chance to find more relevant results in top ranked answers especially with good ranking methods according to application requirements. We will see shortly our demo system for DBLP dataset with application specific ranking indeed returns more relevant results in top ranked answers by exploiting ID references than SLCA alone.

Finally, we address that our approach computes SLCA, ELRA pair and ELRA group results in three independent steps. Therefore, real XML keyword search engine can first output SLCA results for those applications where the computation of ELRA pair and group results may be slow. Then, while users are consuming SLCA results, the system can continue searching ELRA pair and group result in the background such that the user will not perceive the relatively long execution time in exploiting ID references when users want more results based on ELRA pair and ELRA group semantics.

Table 6.3: Average result size for SLCA/ELRA pair/ELRA group of random queries in XMark dataset

Keyword #	XMark					
	SLCA	ELRA pair		ELRA group		Total
		pair#	node#	group#	node#	node#
2	31	3492	1148	791	1148	1177
3	6	1103	597	557	1755	1761
4	2	206	154	301	1916	1918
5	1	51	40	176	1898	1899

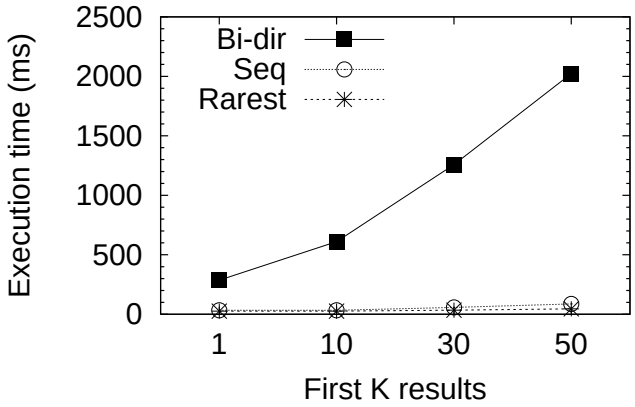
6.2.3 Tree + IDREF v.s. general digraph model

Bi-directional expansion (Bi-dir for short) [30] is one good heuristics for keyword search in the general digraph model. It tries to search as a small portion of a graph as is possible and outputs result reduced subgraphs in the approximate order of the result generation during expansion.

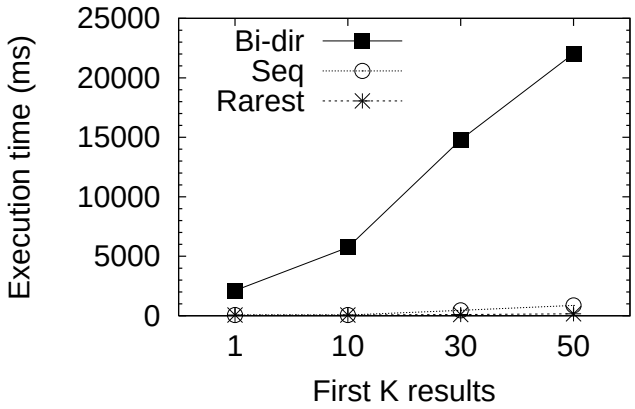
Therefore, instead of comparing the time spent in computing all search results, we compare the time spent in getting first-k responses between Bi-dir expansion and our algorithms in Tree + IDREF model. Note that we slightly modify Bi-dir expansion to not expand to a node that are more than two ID reference edges away from a keyword node. In this way, the results of Bi-dir is similar to our algorithms in that any two nodes in a result are not more than 2 hops away. Sample runs show this modification improves the efficiency of Bi-dir in getting first-k responses by limiting its search space. Also note that, even with this search space limitation, the time spent in waiting for Bi-dir to complete searching all results is unbearable for sample runs.

The experimental comparisons of various keyword query sizes are shown in Figure 6.6 for DBLP dataset and Figure 6.5 for XMark.

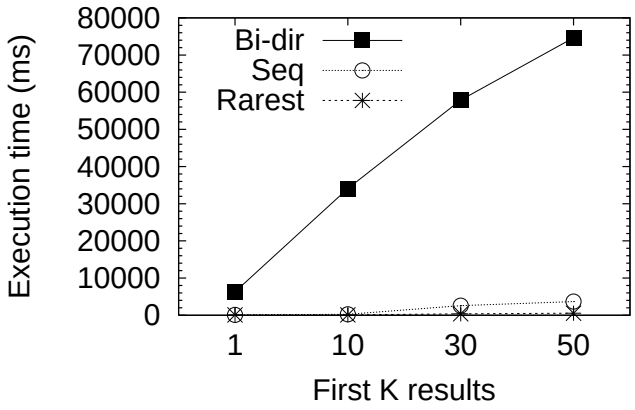
All results in both datasets clearly demonstrate Bi-directional (Bi-dir) in the digraph model is significantly slower than our Sequential-lookup (Seq) and



(a) queries of 2 keywords

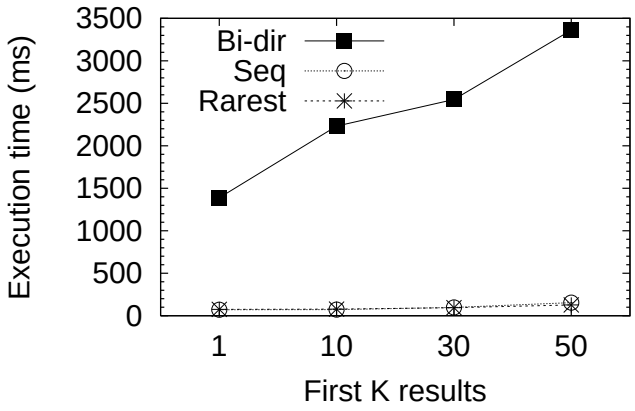


(b) queries of 3 keywords

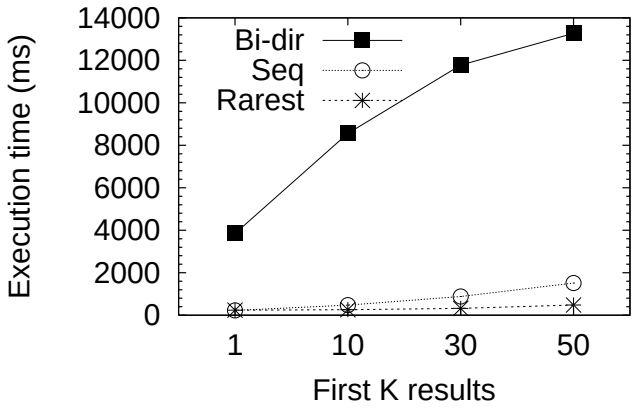


(c) queries of 4 keywords

Figure 6.5: Time comparisons between Bi-Directional Expansion and proposed algorithms for getting first-k responses in XMark



(a) queries of 2 keywords



(b) queries of 3 keywords

Figure 6.6: Time comparisons between Bi-Directional Expansion and proposed algorithms for getting first-k responses in DBLP

Rarest-lookup (Rarest) in Tree + IDREF data model. In many cases, first-k of Bi-dir is even slower than Rarest-lookup to compute all results (with result size in the order of hundreds). For example, if we consider Figure 6.6(a) and (b) with Figure 6.3, we can see the execution time of Bi-dir in getting the first response is much slower than Rarest-lookup to finish computing all results. Similarly, if we consider Figure 6.5(b) and (c) with Figure 6.4, we can see the time of Bi-dir in getting the first-10 response for queries of 3 and 4 keywords is again slower than Rarest-lookup in computing all results

The reasons for the inefficiency of Bi-dir are: Firstly, at each expansion, Bi-dir needs to find the best node to expand among all expandable nodes in order to find the next result quickly; while our algorithms simply check nodes in document order and saves the efforts in heuristic best node finding. Secondly, Bi-dir involves floating point numbers in computing and comparing the goodness of expandable nodes. Thirdly and more importantly, when Bi-dir computes or updates the goodness of a node, it has to recursively propagate the goodness to all neighbors to improve their goodness until no nodes' goodness can be improved.

Finally, some readers may notice we only have results of keyword query size up to three for DBLP dataset. This is due to that we follow [30] for Bi-dir to keep the entire searched digraph portion in memory. We encounter Java Heap out of memory exception in case of 4-keyword query size for DBLP and 5-keyword query size for XMark datasets, even we set java virtual memory to 800M. However, from existing results, we can see that the efficiency of Bi-dir drops as the number of query keywords increases; while Rarest-lookup is quite scalable to keyword query size.

6.3 Comparison of result quality based on sample queries

In this part, we study the result quality of our ICRA demo system based on Tree + IDREF model for XML keyword search in DBLP bibliography with application specific ranking.

We use sample queries of length 2-5 with wide range of meanings (as shown in Table 6.4) to measure the effectiveness of our ICRA demo system compared with other five existing systems, including three academic systems and two commercial

Table 6.4: Tested queries

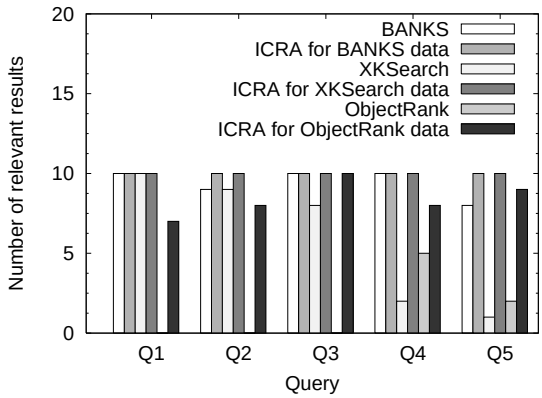
ID	Query	Meaning
Q1	Giora Fernández	Co-author
Q2	Jim Gray transaction	Topic by author
Q3	Dan Suciu semistructured	Topic by author
Q4	Conceptual design relational database	Topic
Q5	Join optimization parallel distributed environment	Topic

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 is judged by discussions of a small group in our database lab, including volunteers.

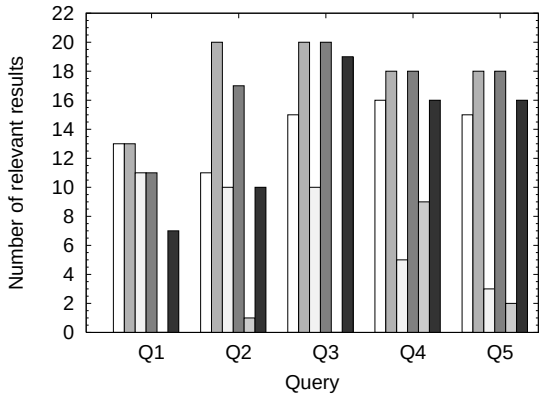
6.3.1 ICRA v.s. other academic demos

Now, we report the comparison among other academic demo systems for keyword search in DBLP (BANKS [10,30], XKSearch [46], ObjectRank [6]) and our ICRA demo system base on Tree + IDREF model with DBLP specific ranking. BANKS [10, 30] returns results based on their bi-directional expansion in digraph model. XKSearch [46] returns results based on SLCA semantics in tree model, except XKSearch returns a publication when an SLCA is smaller than the subtree rooted at the publication. ObjectRank [6] identifies publications as objects in DBLP and adopts authority-based ranking to rank publications for each query. Its main idea is that a publication p is more related to a keyword k if p is cited by more papers contains keyword k.

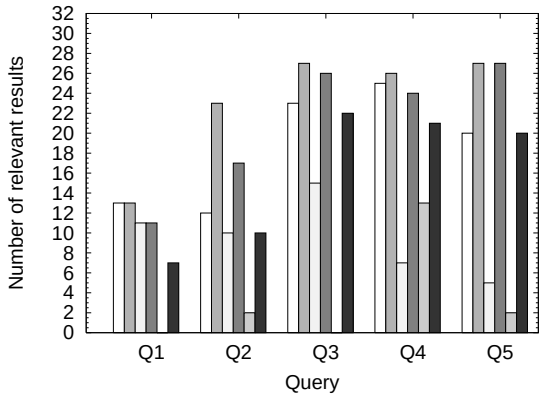
The comparisons for result quality are shown in Figure 6.7. Since four systems use different datasets, for fair comparison, we show the results of ICRA based on other systems' data. For example, "ICRA for BANKS data" in Figure 6.7



(a) top-10 answer



(b) top-20 answer



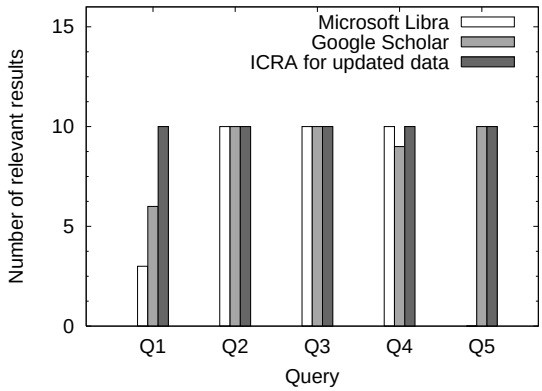
(c) top-30 answer

Figure 6.7: Comparisons of answer quality with other academic systems

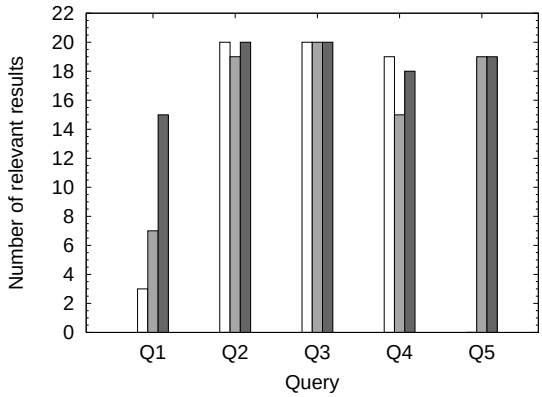
means that we run our system on data used by BANKS. Note that BANKS outputs results in the format of reduced trees (containing publication IDs) instead of lists of publications; we assume there is a middle-ware to transfer BANKS results to publication lists. From Figure 6.7, we can see the result quality 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 (no relevant result for Q1 and Q3) possibly because it does not well maintain information for author names. As expected, the SLCA semantics in XKSearch is too restrictive when it limits the results to publications containing all query keywords. Despite the slow response of BANKS based on Bidirectional expansion, our results are considerably better. For Q4, BANKS results are comparable to ours since they also captures ID references in XML.

6.3.2 ICRA v.s. commercial systems

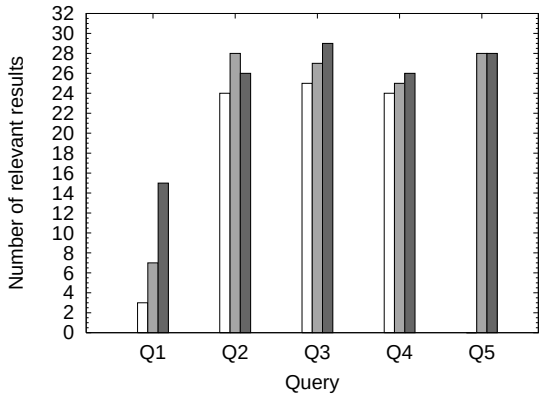
Finally, we show the comparisons of our system with existing commercial systems, Microsoft Libra [1] and Google Scholar [2]. We consider them as commercial systems since they are products of commercial companies. However, readers may regard them as non-commercial system at their choice. The possible 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 6.8 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



(a) top-10 answer



(b) top-20 answer



(c) top-30 answer

Figure 6.8: Comparisons of answer quality with commercial systems

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 co-authors. 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 keyword disjunction with IR-style ranking (i.e. TF*IDF [39] and PageRank [13]) 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 is able to achieve the positive facts of large amount of web data with only 384M DBLP data; and meanwhile our system is not affected by the noises.

Chapter 7

Conclusion

XML emerges as the standard for representing and exchanging electronic data on the Internet. With increasing volumes of XML data transferred over the internet, retrieving relevant XML fragments in XML documents and databases is particularly important. Among several XML query languages, keyword search is a proven user-friendly approach since it allows users to issue their search needs without the knowledge of complex query languages and/or the structures of the underlying XML databases.

7.1 Research summary

This thesis studies the problem of keyword search in XML documents. We propose Tree+IDREF data model for efficient and effective keyword search in XML by exploiting XML ID references. We also address the importance of schema semantics information in answering XML keyword queries when schema semantics is available.

Most prior XML Keyword search techniques are based on either tree or graph

(digraph) data models. In the tree data model, SLCA (Smallest Lowest Common Ancestor) semantics and its variations are generally simple and efficient for XML keyword search. However, they cannot capture the important information residing in ID references which is usually present in XML databases. In contrast, keyword search approaches based on the general graph or directed graph (digraph) model of XML capture ID references, but they are computationally expensive (NP-hard).

In this thesis, we address the importance of ID references in XML databases and propose Tree+IDREF data model to capture ID references while also leveraging the efficiency gain of the tree data model for efficient and effective keyword search in XML. In this model, we propose novel Lowest Referred Ancestor (LRA) pair, Extended LRA (ELRA) pair and ELRA group semantics as complements of SLCA. Studies based on common benchmark data for XML keyword search, such as DBLP and XMark, show the generality and applicability of our novel search semantics with ID references. We also present and analyze algorithms to efficiently compute the search results based on our semantics.

Then, we exploit underlying schema information in identifying meaningful units of result display. We propose rules and guidelines based on object classes and relationship types captured in ORA-SS to formulate result display for SLCA, ELRA pair and ELRA group results.

We also developed a keyword search demo system based on our approach for DBLP real-world XML database for the research community to search for publications and authors. A simple ranking approach is incorporated in the demo system; while a more general ranking approach is left as future work. The demo prototype is available at: http://xmldb.ddns.comp.nus.edu.sg

Experimental evaluation shows keyword search based on our approach in

Tree+IDREF data model achieves much better result quality than that based on SLCA semantics in the tree model; and much faster execution time with comparable or better result quality than that based on the digraph model. Comparisons with existing commercial keyword search system for academic field, such as Google Scholar and Microsoft Libra, also demonstrate comparable or even superior effectiveness of our approach in terms of result quality.

7.2 Future directions

Relevance oriented ranking is a crucial issue for effective keyword search systems. In this thesis, we only present a simple and specific ranking approach that is tailored for DBLP datasets. It would be an interesting future work to study and extend existing ranking approaches in Information Retrieval, such as TF*IDF [39], PageRank [13], HITS [31], etc for effective ranked keyword search in XML in general.

However, effective relevance oriented ranking in XML poses new challenges. One particular challenge is the ambiguity that the same word may appear in different tags and carries different meanings in XML. For example, "Lecturer Smith" is a reasonably intuitive query to search for Lecturer whose name is "Smith". However, our current approach also includes lecturers who do not have "Smith" in their names but have address in "Smith Street". Simple syntax can help to resolve such ambiguities. For example, users can use "Lecturer Name:Smith" or "Lecturer Address:Smith" to explicitly specify if they are interested in name or address. While syntax is powerful, users will prefer using pure keyword queries in most cases. Therefore, it would be interesting to resolve ambiguity based on human intuitions without syntax.

One possible way is to use statistics which is an effective approach to model intuitions. For example, when people see "Smith", it is more intuitive to be related to human names and less intuitive to be related to address, which can be explained from statistical point of view that "Smith" is more frequently used as a person's name. Similarly, when we see "address" in the database with schema context in Figure 3.3, we usually regard it as a tag name. Despite it matching the tag name Address, we can also explain such intuitive "regard" from statistics point of view that "address" is frequent in "Address" nodes as tag names. Therefore, it is interesting to make search engines understand human intuitions based on statistics to resolve ambiguities so that search engines will consider queries "Lecturer Smith" and "Lecturer Name Smith" as searching via Name nodes and query "Lecturer Address Smith" as searching via Address nodes.

Since most ranking approaches in IR are indeed based on statistics, combining ambiguity resolving into relevance oriented ranking based on statistics for keyword search in XML would be a promising direction.

Moreover, our current approach exploits schema information for result output, but does not fully exploit schema during the computation of SLCA, ELRA pair and group results. This approach has the advantage that result computation is largely system independent so that system administrators of a particular application can concentrate on issues of result display without bothering about result computation. However, it would be interesting to study whether schema can be helpful to improve efficiency during result computation. Removal of the requirement of schema in result display is also worth investigation for cases where ORA-SS is not available.

Finally, it is also interesting to study alternative index structures to improve the computation efficiency of our proposed search semantics.

Bibliography

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

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

[3] Berkeley DB. http://www.sleepycat.com/.

[4] Online computer library center. introduction to the dewey decimal classification.

[5] S. Agrawal, S. Chaudhuri, and G. Das. DBXPlorer: A system for keyword-based search over relation databases. In *Proc. of ICDE Conference*, pages 5–16, 2002.

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

[7] Z. Bao, B. Chen, and H. W. T. W. Ling. Using semantics in XML query processing. In *ICUIMC*, 2008.

[8] Z. Bao, T. W. Ling, and B. Chen. SemanticTwig: A semantic approach to optimize XML query processing. In *DASFAA*, pages 282–298, 2008.

[9] A. Berglund, S. Boag, and D. Chamberlin. XML path language (XPath) 2.0. W3C Working Draft 23 July 2004.

[10] 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.

[11] S. Boag, D. Chamberlin, and M. F. Fernandez. XQuery 1.0: An XML query language. W3C Working Draft 22 August 2003.

[12] J.-M. Bremer and M. Gertz. An efficient XML node identification and indexing scheme. Technical Report CSE-2003-04, University of California at Davis, 2003.

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

[14] N. Bruno, D. Srivastava, and N. Koudas. Holistic twig joins: optimal XML pattern matching. In *SIGMOD Conference*, pages 310–321, 2002.

[15] 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.

[16] B. Chen, T. W. Ling, M. T. Özsu, and Z. Zhu. On label stream partition for efficient holistic twig join. In *DASFAA*, pages 807–818, 2007.

[17] B. Chen, J. Lu, and T. W. Ling. Keyword search in bibliographic XML data. In *ICUIMC*, 2007.

[18] B. Chen, J. Lu, and T. W. Ling. Exploiting ID references for effective keyword search in XML documents. In *DASFAA*, pages 529–537, 2008.

[19] T. Chen, J. Lu, and T. W. Ling. On boosting holism in XML twig pattern matching using structural indexing techniques. In *SIGMOD*, pages 455–466, 2005.

[20] 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.

[21] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A semantic search engine for XML. In *VLDB*, pages 45–56, 2003.

[22] N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. In *SODA*, pages 253–259, 1998.

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

[24] S. L. Hakimi. Steiner's problem in graphs and its implications. Networks, 1:113-131, 1971.

[25] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD Conference*, pages 305–316, 2007.

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

[27] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *Proc. of VLDB Conference*, pages 670–681, 2002.

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

[29] H. Jiang et al. Holistic twig joins on indexed XML documents. In *Proc. of VLDB*, pages 273–284, 2003.

[30] 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.

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

[32] M. Ley. DBLP computer science bibliography record. http://www.informatik.uni-trier.de/ ley/db/.

[33] G. Li, J. Feng, J. Wang, and L. Zhou. Effective keyword search for valuable LCAs over XML documents. In *CIKM*, pages 31–40, 2007.

[34] 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.

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

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

[37] J. Lu, T. Chen, and T. W. Ling. Efficient processing of XML twig patterns with parent child edges: a look-ahead approach. In *CIKM*, pages 533–542, 2004.

[38] J. Lu, T. W. Ling, C. Chan, and T. Chen. From region encoding to extended dewey: On efficient processing of XML twig pattern matching. In *VLDB*, pages 193–204, 2005.

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

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

[41] A. R. Schmidt et al. XMark an XML benchmark project. http://monetdb.cwi.nl/xml/index.html.

[42] C. Sun, C. Y. Chan, and A. K. Goenka. Multiway SLCA-based keyword search in XML data. In *WWW*, pages 1043–1052, 2007.

[43] H. Wu, T. W. Ling, and B. Chen. VERT: A semantic approach for content search and content extraction in XML query processing. In *ER*, pages 534–549, 2007.

[44] X. Wu, T. W. Ling, M.-L. Lee, and G. Dobbie. Designing semistructured databases using ORA-SS model. In *WISE (1)*, pages 171–182, 2001.

[45] 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.

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

[47] N. Yuruk, X. Xu, C. Li, and J. X. Yu. Supporting keyword queries on structured databases with limited search interface. In *DASFAA*, pages 432–439, 2008.