TOWARDS AN EFFECTIVE PROCESSING OF XML KEYWORD QUERY

BAO ZHIFENG

Bachelor of Computing (Honors)

National University of Singapore

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2010

ACKNOWLEDGEMENT

My first and foremost thank goes to my supervisor Prof. Ling Tok Wang who first introduced me to database research. I still remember the first day I met Prof. Ling in year 2005, when I came into his office to express my willing to work on his project as my Honor Year project. Without his careful supervision, my work cannot be one of the best Honor Year student projects. His heuristic guidance in our discussion makes me think and work very independently and I really appreciate this "learn by doing" way. As a supervisor, his insights in database research and rigorous attitude are invaluable for my research. As a mentor, his kindness and wisdom help me to be a happy PhD student. I will benefit from these not only for a Ph.D. degree but also for the whole life.

Prof. Ooi Beng Chin, who has influenced me in many ways, deserves my special appreciations. He sets the high standard for our database research group, insists on the importance of hard working, and advocates the value of building real systems. Without his full credits to me, I would not be able to work in AT&T shannon lab and University of Queensland for summer internships. He does set a great figure in both my career and life to be a strong man anywhere anytime.

I would like to thank Prof. Stephane Bressan and Prof. Lee Mong Li for serving on my thesis committee and providing many useful comments on the thesis.

I would like to thank Dr. Divesh Srivastava who generously hosted me in AT&T Shannon lab, where I spent 5 months in USA. Whenever I have a question, his door is always open to discussion. Dr. Divesh taught me how to work hard and play harder, and it is invaluable for me to learn from him how to present one's idea in a precise and concise way. I also want to thank all my cooperators in AT&T Shannon lab, Dr. Graham Cormode, Dr. Theodore Johnson and Dr. Vladislav Shkapenyuk, who helped me start a new research area. Dong Xin and her family deserve my special thanks, they offer me their house for accommodation and taught me how to lead a delightful life. I would also like to thank Prof. Zhou Xiaofang, who hosted me for 3-month internship in University of Queensland, and colleagues in UQ, Henning, Xie Qing, Yang Yang, Zhu Xiaofeng, Zheng Kai and Cheng Ran.

I appreciate all the people coauthoring with me, especially Lu Jiaheng and Chen Bo. Their participation further strengthened the technical quality and literary presentation of our papers. I am also appreciated to the help from Prof. Anthony Tung, Prof. Tan Kian Lee and Prof Chan Chee Yong in our database group.

The last eight years in National University of Singapore have been an exciting and wonderful journey in my life. I met a lot of friends who brought a lot of fun to my life. They are Daisuke Mashima, Dong Xin, Eric, Ge Zihui, Jin Yu, Mao Yun, Pei Dan, Qian Feng, Yu Fang and Zhao Qi in AT&T lab, Cao Yu, Chen Su, Dai Bingtian, Liu Shanshan, Ju Lei, Sheng Chang, Sun Jie, Wang Xiaoli, Wu Huayu, Wu Ji, Wu Jun, Wu Sai, Wu Wei, Yang Fei, Xiang Shili, Xu Liang, Xue Mingqiang, Ying Shanshan, Zhang Dongxiang, Zhang Jingbo, Zhang Meihui and Zhang Zhenjie in NUS.

Lastly but not least, my deepest love is reserved for my parents, Bao Peiliang and Zhao Xiuming, and my grandparents. Their unconditional love and nutrition have brought me into the world and developed me into a person with endless passion and power.

Publications

Materials in this thesis are revised from the following list of our previous publications.

1. **Zhifeng Bao**, Bo Chen, Tok Wang Ling, Jiaheng Lu. "Effective XML Keyword Search with Relevance Oriented Ranking", *The 25th IEEE International Conference on Data Engineering (ICDE)*, PP. 517-528, Shanghai, China, 2009. [16]

2. **Zhifeng Bao**, Bo Chen, Tok Wang Ling, Jiaheng Lu. "Demonstrating Effective Ranked XML Keyword Search with Meaningful Result Display", *The 14th Conference on Database Systems for Advanced Applications (DASFAA)*, PP. 750-754, Brisbane, Australia, 2009. [15]

3. Jiaheng Lu, **Zhifeng Bao**, Tok Wang Ling, Xiaofeng Meng. "XML Keyword Query Refinement", *The 1st International Workshop on Keyword Search on Structured Data (KEYS)*, PP. 41-42, Providence, USA, 2009. [84]

4. **Zhifeng Bao**, Jiaheng Lu, Tok Wang Ling, Bo Chen. "Towards an Effective XML Keyword Search", *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2010. **Special Issue on Best Papers of ICDE 2009**. [19]

5. **Zhifeng Bao**, Jiaheng Lu, Tok Wang Ling, Liang Xu, Huayu Wu. "An Effective Object-level XML Keyword Search", *The 15th Conference on Database Systems for Advanced Applications (DASFAA)*, Tsukuba, Japan, 2010. [20]

6. **Zhifeng Bao**, Jiaheng Lu, Tok Wang Ling. "XReal: An Interactive XML Keyword Searching", *The 19th ACM International Conference on Information and Knowledge Management (CIKM)*, Toronto, Canada, 2010. [18]

7. Jiaheng Lu, **Zhifeng Bao**, Tok Wang Ling, Xiaofeng Meng. "Content-aware Query Refinement in XML Keyword Search", *Submitted to the IEEE Transactions on Knowledge and Data Engineering*. [83]

During the PhD study, I have participated in some XML query processing related works, and the resulted publications are listed in chronological order as follows:

8. Liang Xu, **Zhifeng Bao**, Tok Wang Ling. "A Dynamic Labeling Scheme Using Vectors", *The 18th International Conference on Database and Expert Systems Applications (DEXA)*, PP. 130-140. Regensburg, Germany, 2007. [115]

9. **Zhifeng Bao**, Huayu Wu, Bo Chen, Tok Wang Ling. "Using semantics in XML query processing", *The 2nd International Conference on Ubiquitous Information Management and Communication (ICUIMC)*, PP. 157-162, Suwon, Korea, 2008. [21]

10. **Zhifeng Bao**, Tok Wang Ling, Jiaheng Lu, Bo Chen. "SemanticTwig: A Semantic Approach to Optimize XML Query Processing", *The 13th Conference on Database Systems for Advanced Applications (DASFAA)*, PP. 282-298, New Delhi, India, 2008. [17]

11. Junfeng Zhou, **Zhifeng Bao**, Tok Wang Ling, Xiaofeng Meng. "MCN: A New Semantics Towards Effective XML Keyword Search", *The 14th Conference on Database Systems for Advanced Applications (DASFAA)*, PP. 511-526, Brisbane, Australia, 2009. [123]

12. Huayu Wu, Tok Wang Ling, Liang Xu, **Zhifeng Bao**. "Performing grouping and aggregate functions in XML queries", *The 18th International World Wide Web Conference (WWW)*, PP. 1001-1010, Madrid, Spain, 2009. [110]

13. Liang Xu, Tok Wang Ling, Huayu Wu, **Zhifeng Bao**. “DDE: from dewey to a fully dynamic XML labeling scheme”, *The 35th SIGMOD international conference on Management of data (SIGMOD)*, PP. 719-730, Providence, USA, 2009. [117]

14. Jiaheng Lu, Tok Wang Ling, **Zhifeng Bao**, Chen Wang. “Extended XML Tree Pattern Matching: Theories and Algorithms”, *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2010. [85]

15. Liang Xu, Tok Wang Ling, **Zhifeng Bao**, Huayu Wu. “Efficient Label Encoding for Range-Based Dynamic XML Labeling Schemes”, *The 15th Conference on Database Systems for Advanced Applications (DASFAA)*, PP. 262-276, Tsukuba, Japan, 2010. [116]

16. Huayu Wu, Tok Wang Ling, Gillian Dobbie, **Zhifeng Bao** and Liang Xu. “Reducing Graph Matching to Tree Matching for XML Queries with ID References”, *The 21st International Conference on Database and Expert Systems Applications (DEXA)*, Bilbao, Spain, 2010. [109]

CONTENTS

SUMMARY

Inspired by the great success of information retrieval (IR) style keyword search on the web, keyword search over XML data has emerged recently. As compared to keyword search on the web, XML keyword search brings several new challenges. (1) The target that a user query intends to search for is usually unknown or implicit. (2) The keyword ambiguity problem: a keyword can appear as both a tag name and a text value of some node; a keyword can appear as the text values of different XML node types and carry different meanings; a keyword can appear as the tag name of different XML node types with different meanings. It further obstructs identifying the constraints that a user query intends to search via. (3) The hierarchical structure of XML data has to be taken into account in devising the matching semantics and result ranking scheme. This dissertation discusses three aspects in the construction of an effective XML keyword search engine while conquering the above challenges.

First, we study the keyword search over XML data tree without ID references captured. In particular, we propose a statistics-based approach to identify the target(s) that a user query intends to search for, quantify the likeliness of different search intentions in result ranking, and end with designing an XML Term Frequency * Inverse Document

Frequency (XML TF*IDF) result ranking scheme. Second, we realize that by taking the ID references among elements in XML data into consideration, more relevant results can be found. Through identifying the objects of interest from the given semantic information of XML data, we model XML data as a set of object trees that are interconnected by either containment or reference edges, and propose a series of matching semantics at object tree level. As a result, user's search concern on real-world objects can be precisely captured; by distinguishing the containment and reference edge in XML data, the efficiency of matching result generation is improved as compared to previous works on keyword search over general directed graph. *Third*, we observe that user queries may contain irrelevant or mismatched terms, typos etc, which may easily lead to nonsensical or empty result. An effective query refinement is a demanding functionality of an XML keyword search engine. Specifically, we propose a novel query ranking model to quantify the confidence of a refined query (RQ) candidate, which can capture the morphological/semantical similarity between Q and RQ and the dependency of keywords of RQ over the XML data. Besides, we integrate the job of looking for RQ candidates and generating their matching results as a single problem, thus guaranteeing the existence of meaningful matching results of the suggested RQs.

As a result, by incorporating the above proposed techniques, a keyword search engine prototype have been built. Through a comprehensive experimental study on both the real-life and synthetic data set, the proposed solutions are shown to be efficient, effective and scalable.

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background on XML and XML Keyword Search

As the World Wide Web is becoming a major carrier to share and disseminate information, HTML (HyperText Markup Language) [99] and XML (EXtensible Markup Language) [26] were initially designed to tailor for large-scaled web-compliant information publishing on Web. On one hand, in contrast to HTML which has predefined elements and attributes, for output formatting purpose XML allows users to define their own elements specific to their application or business needs, where data stored in XML contains more meaningful structural and semantic information, manifesting more powerful expressiveness than HTML. On the other hand, in contrast to SGML (Standard Generalized Markup Language) [6] whose specification is too complex to use and implement, XML's specification keeps the essence of SGML's power and extensibility with a much simpler specification. All of these promote XML to be a standard in data exchange and representation over Internet, which increases the volume of data encoded in

XML.

Figure 1.1 shows a sample XML document containing the papers of an academic conference, where data is bounded by a pair of starting and ending tags. For example, line 1 describes the root element of the document, namely *conference*, and the remaining lines describe its four child elements, i.e. *year* (line 2), *title* (line 3), *venue* (line 4) and *inproceedings* (line 5-26); finally, the last line defines the end of the root element.

```
1  <conference>
2  <year> 2008 </year>
3  <title> Very Large Databases </title>
4  <venue> Auckland New Zealand </venue>
5  <inproceedings>
6      <paper id = "1">
7          <title> reasoning and identifying relevant matches for XML keyword search
8          </title>
9          <author> ziyang liu </author>
10         <author> yi chen </author>
11         <section name = "introduction">
12             <subsection name = "motivation">
13                 When processing a keyword query over XML data, we aim to find the
14                 most relevant and meaningful fragments …
15             </subsection> ……
16         </section>
17         ……
18         <section name = "Experimental study">
19             As we can see, the efficiency of our query processing method is …
20         </section>
21     </paper>
22     <paper id = "2">
23         <title> Generating XML structure using examples and constraints </title>
24         ……
25     </paper>
26 </inproceedings>
27 ……
28 </conference>
```

Figure 1.1: A sample XML document

The elements in an XML document usually form a document tree, starting at the root and branches to the lowest level of the tree. Each node in the tree corresponds to an element, an attribute or character data in XML document, and each edge in the tree represents the element-subelement or element-attribute relationship. For example, Figure 1.2 shows a tree model [1] of the XML document in Figure 1.1.

[1]For the convenience of typesetting, for the values of leaf nodes we only show part of them related to

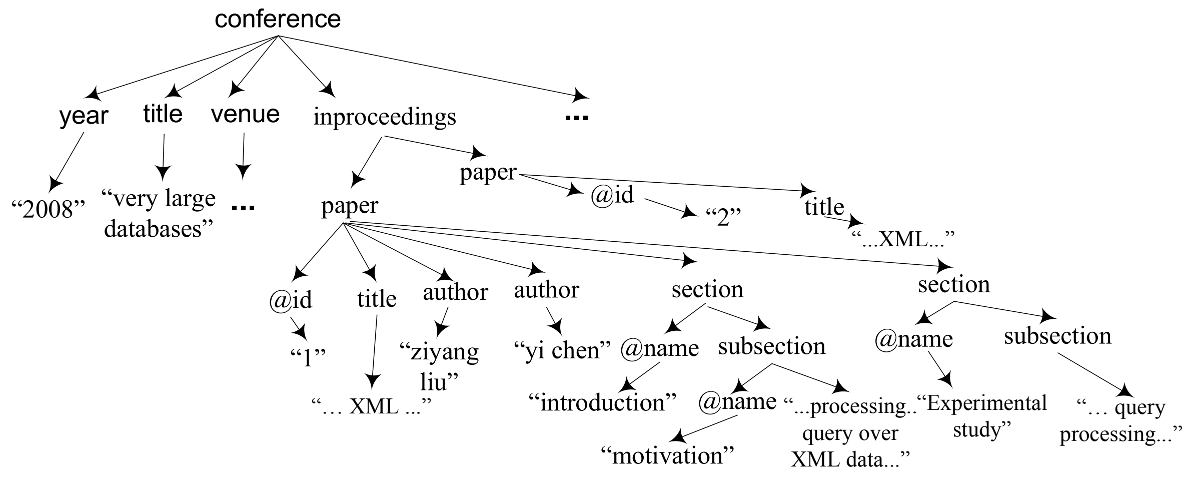


Figure 1.2: Tree model of XML document in Figure 1.1

As the volume of XML data is increasing, it is demanding to provide efficient and effective management over XML data, such as structured query processing and keyword query processing. Regarding structured query processing, database systems have been notorious for being hard to use (even for expert users) all the time, because users have to learn structured query languages specifically designed for such data (e.g. XQuery, XPath for accessing XML document), and have to be very familiar with the (possibly complex) underlying schema of such data. Even worse, unlike relational database where the schema is relatively small and fixed, XML data model allows varied structures and values, making it more difficult for web user to issue a structured query. On the contrary, keyword search allows users to pose their information need in a free form, and its great success on the World Wide Web, e.g. google keyword search engine, has inspired an increasing interest in studying keyword search over XML database.

Unlike the ranked retrieval style keyword search such as google over collections of unstructured documents, XML presents more structural and semantic information, thus a result matching semantics is needed to find the most relevant and meaningful fragments of XML data. Among all matching semantics proposed, the most basic one is called

the keyword query examples presented later in this section.

Lowest Common Ancestor (LCA) [52]. Intuitively, LCA returns a set of elements, each of which contains[2] at least one occurrence of all query keywords in its subtree, after excluding the occurrences of keywords in the sub-elements that already contain all query keywords. As a result, the above definition ensures that all independent occurrences of the query keywords are represented in the query result, as illustrated in Example 1.1.

Example 1.1. *Consider a keyword query Q = {XML, query, processing} issued on the XML data in Figure 1.1.*

By LCA semantics, two results R_1 and R_2 are returned: R_1 is the subsection *element (line 12-15), as it directly contains all query keywords[3] in its value part; R_2 is the* paper *element (line 6-21). We can find, although R_2 is an ancestor of R_1 which already contains all keywords, it also contains independent occurrences these keywords, where "XML" is contained in its* title *sub-element (line 7-8), "query" and "processing" are contained in its* section *sub-element (line 18-20).* □

Later on, the concept of Smallest Lowest Common Ancestor (SLCA) is proposed [118], in order to find the smallest LCAs that do not contain other LCAs in their subtrees. The rational behind is that, users often favor subtrees of smaller size as it contains more compact and specific information they intend to explore. For illustration, Let us refer back to Example 1.1, the LCA result R_2 (the *paper* element in line 6-21) is not a qualified SLCA, because it contains a *subsection* sub-element (line 12-15) which is already a LCA of all query keywords. Therefore, only R_1 is returned as an SLCA result.

1.2 Research Problem: Effective XML Keyword Search

As a keyword search engine, the most important issue to be resolved is how to improve the user search experience, especially for novice users. Regarding search expe-

[2]In this thesis, whenever we mention "contain", it means the keyword is contained within either the value part or the tag name of XML element.

[3]The keywords contained is highlighted in bold text.

rience, effectiveness and efficiency are the two critical aspects in evaluating the performance of a keyword search engine. In this thesis, we put the effectiveness issue as our major focus. In a nutshell, *effectiveness* in XML keyword search amounts to finding both *meaningful* and *relevant* fragments of XML data.

Inspired by the great success of information retrieval (IR) style keyword search on the web, keyword search on XML has emerged recently. However, the difference between unstructured web data and semi-structured XML data results in three new challenges:

1. Identify the user search intention, i.e. identify the XML node types that user wants to search for (i.e. search targets) and search via (i.e. search constraints).

2. Resolve keyword ambiguity problems: a keyword can appear as both a tag name and a text value of some node; a keyword can appear as the text values of different XML node types and carry different meanings; a keyword can appear as the tag name of different XML node with different meanings.

3. As the search results are sub-trees of the XML document, new scoring function is needed to estimate its relevance to a given query. Besides, an appropriate granularity for the sub-trees is critical.

As we can see, in order to resolve the above challenges thoroughly, we should be able to combine the techniques in database (DB) and information retrieval (IR) community, as it needs not only the DB-style specification on defining the structure-aware matching results, but also needs similar IR-style measurement to judge the similarity of the contents of matching results.

Unfortunately, existing methods cannot thoroughly resolve these challenges. One major problem is, existing works that focus on the matching semantics design [52, 79, 118, 119] only account for the internal structure and occurrences of keywords, without figuring out the most promising search targets and constraints of a user query.

Example 1.2. *Consider the query in Example 1.1 again, by LCA there are two matching results R_1 and R_2, which indeed represent two completely different search intentions respectively (even the search target is different): R_1 corresponds to a* subsection *whose content contains all query keywords, while R_2 corresponds to a* paper *which contains "XML" in its* title *and "query", "processing" in its* subsection*'s content. Unfortunately, LCA is neither able to distinguish these two search targets or intentions, nor able to account for the structural positions of the matched keywords in a matching LCA result; instead, it only trivially enforces the occurrences of all keywords in a result.*

From the above example, we can see that existing works that enforce the occurrences of query keywords in matching result definition cannot resolve the problem of search target identification, instead it mixes the results corresponding to each of the above search targets. Thus, it leads to a yet unsolved problem, which is to design IR-liked scoring methods quantify the confidences of those candidates as the desired search target. Further, an appropriate scoring model is needed to quantify the results associated with different search predicates (e.g. R_1 and R_2 have different matching criteria). Another problem of existing works is the integration of DB and IR techniques. Most previous works [52, 38, 73] adopt the following flow in answering a keyword query: it first finds all the matching results according to a particular matching semantics, followed by extending the existing IR scoring methods (such as TF*IDF) to account for the structural similarity of results. In other words, it separates the IR-style ranked retrieval approach and the DB-style precise matching in the exploration of query results, which may incur the problem of missing some relevant results.

1.3 Contributions of This Thesis

In this thesis, we mainly investigate how to integrate both DB and IR techniques in a seamless way to enforce effective keyword query processing over XML data. Our work is also in line with the current trend of DB&IR integration to achieve ranked retrieval on semi-structured XML data [12, 34]. Our major contributions include identifying the search target of an XML keyword query, illustrating what an appropriate matching result should be, proposing relevance-oriented result ranking scheme, finding appropriate content-aware refinements for an XML keyword query, and building an XML keyword search engine prototype incorporating our proposed techniques. The following three sections briefly describe the contribution of our three works respectively.

1.3.1 Effective Keyword Search Over XML Data Tree

When XML data is modeled as a labeled tree structure, the result is in form of a subtree containing all query keywords. We propose an IR-style approach for XML keyword query processing, which basically utilizes the statistics of underlying XML data to address the problem of search intention identification (which includes identifying the search targets and search constraints of a user query) and result ranking. We first propose three major guidelines that a search engine should meet in both search intention identification and relevance oriented ranking for search results. Then based on these guidelines, we design novel formulae to identify the desired search for nodes and search via nodes of a query, and design a novel XML TF*IDF ranking strategy to rank the individual matches of all possible search intentions. Lastly, our approach manifests its superiority especially for pure XML keyword queries.

1.3.2 Effective Keyword Search Over XML Directed Graph

Besides the containment edges (i.e. parent-child and ancestor-descendant edges) between XML elements, we find that without taking the ID references between elements in XML data into account, some relevant results may be missed. Therefore, in this work, we investigate how to find meaningful and relevant results of a keyword query over the XML data with IDRefs, which is modeled as a special directed graph.

In contrast to previous work on keyword search over general digraph [37, 65, 53, 57], we propose an alternative approach by utilizing the available semantic information to improve both the efficiency and effectiveness of the result matching and ranking part. In particular, we model XML document as a set of interconnected object-trees, where each object tree is in form of a subtree representing a real-world entity. An important feature of this model is, we distinguish containment edges and reference edges in XML data. Based on this model, we propose object-level matching semantics called *Interested Single Object* (ISO) and *Interested Related Object* (IRO), where ISO is to capture a single object as user's interested search target, while IRO is to capture multiple objects (connected/related by containment or reference edges) as user's interested target. Subsequently, we design an object-level relevance oriented result ranking scheme, and propose efficient algorithms to compute the query results and do the ranking during result exploration. Lastly, we build a prototype incorporating all the above techniques proposed, and an online demo of our system on DBLP data is available at **http://xmldb.ddns.comp.nus.edu.sg**.

1.3.3 Effective XML Keyword Query Refinement

The above two pieces of work focus on how to find relevant and meaningful data fragments for an XML keyword query, assuming each keyword is intended as part of it. It is also the major research directions in recent years. However, in XML keyword search, user queries quite often contain irrelevant or mismatched terms, typos etc, which may easily lead to empty or meaningless results. At first glance people may think it is nothing different with keyword suggestion facility in web search engines, and we can achieve query refinement through user interaction and feedback. However, interactive reformulation and browsing is generally time-consuming and may irritate customers [12]. It motivates us to introduce the problem of *content-aware XML keyword query refinement*, where the search engine should judiciously decide whether a user query Q needs to be refined during the processing of Q, and automatically find a list of promising refined query (RQ) candidates, and *content-aware* means each RQ candidate found guarantees to have meaningful matching results over the XML data, without any user interaction or a second try. To achieve this goal, we build a query refinement framework consisting of two core parts: (1) we build a query ranking model to evaluate the quality of a refined query RQ of a user query Q, which captures the morphological/semantical similarity between Q and RQ and the dependency of keywords of RQ over the XML data; (2) we integrate the exploration of RQ candidates and the generation of their matching results as a single problem, which is fulfilled within a one-time scan of the related keyword inverted lists optimally. Finally, an extensive empirical study verifies the efficiency and effectiveness of our framework.

1.4 Thesis Outline

The rest of this thesis is organized as follows.

- Chapter 2 reviews the related work. The surveyed topics include XML query languages, XML labeling schemes, XML structured query processing and XML keyword search methods for both labeled tree and directed graph models, and keyword query refinement work.
- Chapter 3 presents our method for identifying the user search target and relevance oriented result ranking scheme over XML data when it is modeled as a labeled tree.
- Chapter 4 presents our method for effective keyword search over XML data when ID references among XML elements are considered.
- Chapter 5 presents our method for effective keyword query refinement and result generation for keyword search over XML data tree.
- Chapter 6 concludes this thesis and lists several future research directions on the topic of effective XML keyword search.

CHAPTER 2

RELATED WORK

In this chapter, we would like to describe the related work. In particular, we first talk about the emergence of XML, followed by two major XML data models; then we discuss the labeling schemes designed for XML data to facilitate the processing of structured query or keyword query. Then we overview the recent literatures on keyword search over the above two data models respectively. Lastly, we investigate the topic of keyword query refinement, which is an important part of a real-life search engine.

XML stands for Extensible Markup Language, which is a markup language much like HTML. But in contrast to HTML which is used to display data, XML initially emerges as a format to transport and store data; moreover, the XML tags are not pre-defined and XML data is usually self-descriptive. From DB viewpoint, XML is an exchange format for structured data; while from IR viewpoint, XML is a format for representing the logical structure of documents. Recently, XML has been becoming a standard for the exchange of heterogeneous data over the web, which increases the volume of data encoded in XML. Therefore, it is attracting a lot of efforts to support structured query processing and keyword query processing on the potentially numerous XML data efficiently and effectively.

```
<StoreDB>
    <customers>
        <customer ID="C1">
            <name> Mary Smith </name>
            <address>
                <street> Art Street </street>
                <city> NJ </city>
            </address>
            <contact> … </contact>
            <interests>
                <interest> fashion </interest>
                <interest> tennis </interest>
            </interests>
        </customer>
        <customer ID="C2">
            <name> John Martin </name>
            …
            <interests>
                <interest> street art <interest>
            </interests>
        </customer>
        …
    </customers>
</StoreDB>
```

Figure 2.1: Sample StoreDB XML document

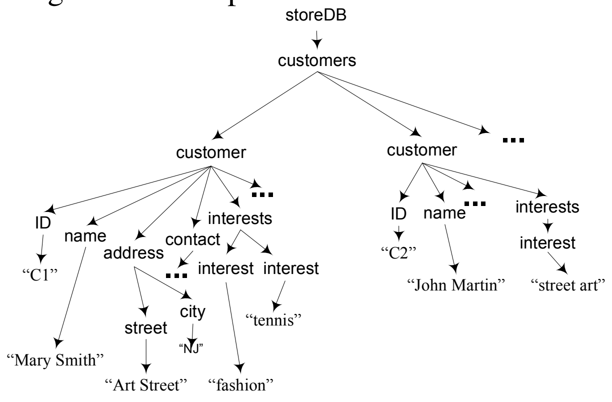


Figure 2.2: Tree model representation for the XML data in Figure 2.1

2.1 XML Data Model

2.1.1 Tree Model

Most of the time, XML documents are treated as trees of nodes, and the root of the tree is called the document node or root node. There are seven major kinds of nodes, i.e. element, attribute, text, namespace, processing-instruction, comment, and root node. Usually the mostly used nodes are element, attribute and text. Figure 2.1 shows a sample

XML document storing the customer information of a store, and Figure 2.2 shows its tree structure representation.

```
<bookstore>
    <books>
        <book ID = "B1">
            <title> XML Introduction </title>
            <authors>
                <author> John Williams </author>
                <author> Daniel Jones </author>
            </authors>
            <cite IDREF = "B2">
            ...
        </book>
        <book>
            <title> ... </title>
            <authors>
                <author> Edward Martin </author>
                <author> Sophia Jones </author>
            </authors>
            <publisher> Oxford </publisher>
        </book>
        ...
    </books>
</bookstore>
```

Figure 2.3: Sample bookstore XML document

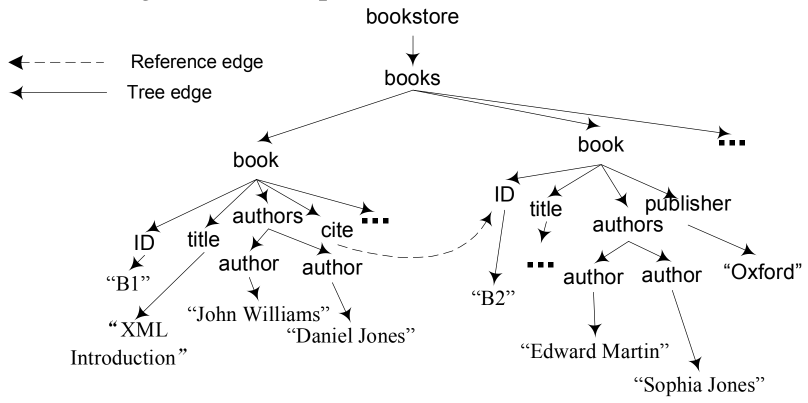


Figure 2.4: Digraph model representation for the XML data in Figure 2.3

2.1.2 Directed Graph Model

Since ID reference (IDRef) in XML data is used to represent the relationship between two XML elements that do not have a hierarchical structural relationship, when the IDRef in XML data is considered in data modeling, the XML data is not of a hierarchical tree structure anymore. Instead, it is more like a directed graph: the containment edge in the previous tree model can be viewed as a directed edge from the parent node to its child node, and the reference edge is a directed edge from one node to another node by IDRef notation in XML document. For instance, Figure 2.3 shows a sample bookstore XML document, which contains the citation relationship between books via IDRef. Such citation can be easily identified in its digraph model, as shown in Figure 2.4, the dotted IDRef edge from book “B1” to book “B2” denotes a citation relationship from “B1” to “B2”.

2.2 Labeling Schemes For XML Data

In the evaluation of (structured or keyword) queries over the XML data tree T, it may frequently involve the determination of whether a structural relationship exists between two nodes in T. In order to facilitate such determinations, nodes are typically labeled. Regarding the design of XML labeling scheme, it should not only support an efficient determination of Ancestor-Descendant (A-D) and parent-child (P-C) relationship at least, but also keep the total label size as compact as possible.

Containment Labeling Scheme

At an earlier time, the containment labeling scheme is proposed [76, 122, 7]. Basically, when preprocessing the XML data tree in document order, it assigns a pair of values in form of $< start : end >$ to each node n, where $start$ denotes the starting position of n being visited, and end denotes the ending position of n being visited. In this way, a node n_1 is an ancestor of node n_2 if the following two properties hold

- $start_{n_1} < start_{n_2}$
- $end_{n_1} > end_{n_2}$

Moreover, in order to decide the Parent-Child (P-C) relationship between two nodes, the only adaption of the above scheme is to add the level information of each node (in the XML data tree) as part of its label.

Dewey Labeling Scheme

Another widely adopted one is the *Dewey number labeling scheme* [105], which works as below: when traversing the XML document in a breadth-first order, each node is assigned a label which is a concatenation of its parent's label and its local order. For instance Figure 2.5 shows an XML data tree by Dewey labeling scheme (note that the values contained within the leaf nodes of the XML data tree is not labeled). A dewey label is a sequence of components separated by '.' where the last component of the sequence represents the local order of the node. The sequence of components before the last component is called the parent label of the node as it is inherited from its parent node. The local order of a node is i if it is the i^{th} child of its parent. Besides, the level information of a node is implicity stored in its dewey label, which is the number of components of a Dewey label.

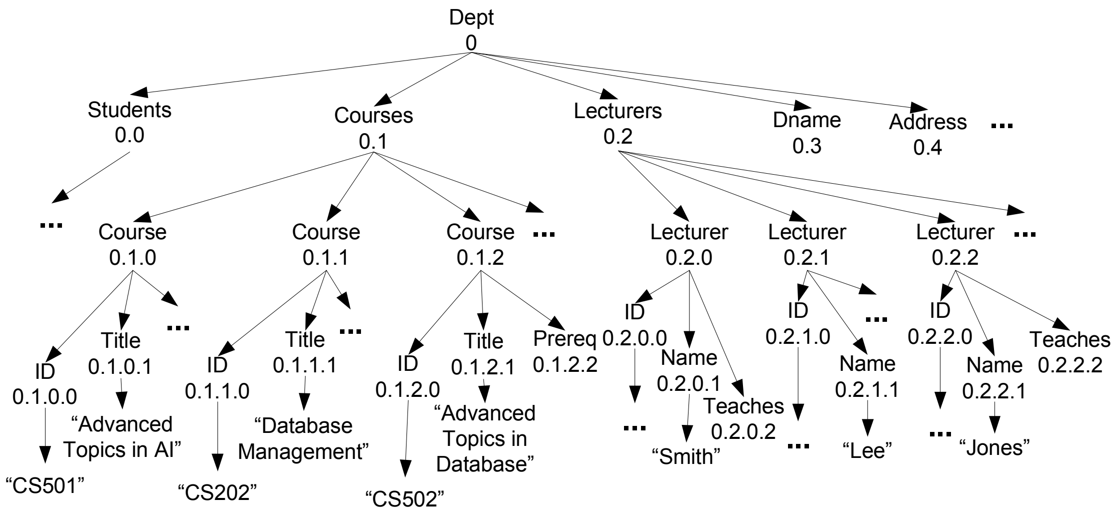


Figure 2.5: Sample XML document (with Dewey Labels)

Since the path information of a node is contained in its labels, Dewey labeling can compute the LCA (Lowest Common Ancestor) of a set of nodes directly, thus becomes the natural choice for XML keyword query processing [118, 52, 38, 79]. For example in Figure 2.5, from the label 0.1.2.1 of node Title, we can know it is at level 4, and is the first child of its parent; the LCA of node 0.1.2.1 and node 0.1.2.2 is Course:0.1.2. Moreover, from dewey label, it is easy to quickly identify the A-D, P-C and sibling relationship between two nodes.

Dynamic XML Labeling Schemes

However, the above two basic labeling schemes only work well for the static XML document, rather than the dynamic XML document. In order to resolve it, Li et al. first proposed to leave some space between adjacent labels for future node insertions [76]; however, it needs relabeling the whole XML document when the spare space is used up. Later, O'Neil et al. proposed a variant of dewey labeling, namely ORDPATH, to resolve the relabeling problem by assigning only positive odd integers in initial labeling, while keeping even and negative integers reserved for later node insertion. A potential problem of this approach is, skipping the even numbers may make the label size less compact. Wu et al. [111] proposed a prime labeling scheme, where the label of a node n is the product result of its self label and the label of its parent node. As all self labels are distinct prime numbers, the A-D and P-C relationship can be easily determined by judging whether the mod of their labels equals to 0. The problem of this approach is, it is expensive to do the computation of prime numbers, and it cannot be used to label a large XML document.

As an alternative approach to avoid relabeling (especially when the XML document is frequently updated), several encoding schemes were proposed, which transform the labels to another format [71, 72, 115, 117]. In particular, Li et al. proposed the Compact Dynamic Binary String (CDBS) encoding [72], which guarantees that a node can be inserted between any two consecutive CDBS labels with the orders maintained and no relabeling of any existing nodes at all. In QED (Quaternary Encoding for Dynamic XML data) [71], given a set of three numbers S={1,2,3}, a QED code is a sequence of the elements in S ending with 2 or 3. Given any two QED codes, it is guaranteed to find a QED code falling between them in the lexicographical order. However, it may not scale well for skewed node insertions due to the fast increase of QED code's length. Thus, Xu et al. proposed a vector based label [115], which is less compact than QED and scales better for skewed insertions. Most recently, a new labeling scheme called DDE (i.e. Dynamic DEwey) [117] was proposed to well control the label quality, which is the most resilient to the number and order of node insertions; besides, it can support LCA computation efficiently.

2.3 Structured Query Languages on XML

Several structured query languages have been proposed so far. They are Lorel [8], XML-QL[40], XML-GL[31], Quilt[32], XPath[23] and XQuery[25]. Here, we mainly discuss XPath and XQuery, both of which are the W3C (World Wide Web Consortium) recommendation.

XPath [23] is a language for addressing parts of an XML document or navigating within an XML document, designed to be used by both XSLT [113] and XPointer [88]. In XPath, an XML document is treated as a tree of nodes, and it mainly uses path expressions (which are similar to traditional file system paths) to locate node or node-sets in an XML document. XPath contains seven major axes, i.e. ancestor, descendant, parent, child, preceding, following, attribute. A location path consists of one or more steps, each separated by a slash(/) or double slash(//). For example, the path expression "//$StoreDB/customers/customer/name$" (issued on the XML document in Figure 2.1) is to find the *name* child of all *customer* elements in StoreDB, and the result returned is a set of nodes {<name>Mary Smith</name>, <name>John Martin</name>}. Here, a double slash (//) signals that all StoreDB elements in the XML document that match the search criteria are returned, regardless of the location or level within the document.

Recently, XQuery [25] is standardized as the major XML query language. The main building block of XQuery consists of path expressions, which addresses part of XML documents for retrieval by value search and structure search in their elements, and returns

a sequence of values. XQuery can be viewed as a big extension of XPath, which gives the possibility of declaring custom functions. So it is something like programming language, which works natively with XML. For example, the following path expression
$for\ \$a\ in//customer[.//interest = `fashion']$
$return\ \$a/name$
is to find the name of customer who is interested in '$fashion$' over the XML document in Figure 2.1. The XQuery evaluation engine returns 'Mary Smith' as a result.

As a core operation in structured XML query processing, XML twig pattern matching has been attracting a lot of research efforts [122, 28, 86, 61, 60, 36, 11, 62, 112, 17]. An XML twig query, represented as a small query tree, is essentially a complex selection on the structure of an XML document. Matching a twig query means finding all the instances of the query tree embedded in the XML data tree. In particular, the idea of *holistic* XML twig pattern processing is first proposed in [28], which has the unique advantage of efficiently controlling the size of intermediate results.

2.4 Keyword Search on Web

In the web, data is stored in form of unstructured documents, and the main issue for keyword search on web is to design the result ranking scheme. There have been a lot of research efforts conducted, and the most classical one is called the Term Frequency * Inverse Document Frequency (TF*IDF) scoring function [101], which emphasizes the *relevance* between a document and a user query. The detailed rational can be referred in section 3.2.1 of chapter 3 later. Another classical ranking model is the well-known PageRank [27] used by the google internet search engine, which emphasizes the *importance* of the document over the World Wide Web. PageRank is a numeric value that represents how important a page is on the web. Google figures that when one page links to another page, it is effectively casting a vote for the other page. The more votes that are cast for a page, the more important the page must be. Also, the importance of the page that is casting the vote determines how important the vote itself is. Google calculates a page's importance from the votes cast for it. How important each vote is is taken into account when a page's PageRank is calculated. PageRank is Google's way of deciding a page's importance. It matters because it is one of the factors that determines a page's ranking in the search results. Note that it isn't the only factor that Google uses to rank pages, but it is an important one.

2.5 Keyword Search on XML Tree Model

As keyword search methods over XML data involve the matching semantics design, efficient evaluation method and result ranking scheme, we will discuss them one by one for the XML labeled tree model.

2.5.1 Matching Semantics and Efficiency Issue

At the early stage of the research in XML keyword search, most research efforts focus on how to define an appropriate matching semantics to find the smallest sub-structures in XML data that each contains all query keywords in tree data model, and meanwhile design efficient algorithms to find all the matched results in XML databases [52, 38, 79, 118, 80, 104, 73, 67, 16, 119, 81].

In *tree* data model, LCA (lowest common ancestor) semantics is first proposed and studied in [102, 52] to find XML nodes, each of which contains all query keywords within its subtree. XRANK [52] proposes a stack-based algorithm to utilize the inverted lists of Dewey labels to compute the LCA results of a query. An inverted list of a keyword k is a list of Dewey labels, each of whose corresponding node directly contains k. The algorithm maintains a result heap and a Dewey stack. The result heap keeps track of the LCA results seen so far. The Dewey stack keeps the current dewey ID, and the longest common prefixes computed. The algorithm sort merges all keyword lists, then each time chooses the node n with the smallest Dewey label (in document order) from the merged list, and computes the longest common prefix of the node denoted by the top entry of the stack and n. Then it pops out all top entries (in the Dewey stack) containing Dewey components that are not part of the common prefix. If a popped entry e contains all keywords, then e is a result node. Otherwise, the information about which keywords that e contains is used to update its parent entry's keywords array. Also, a stack entry is created for each Dewey component of n which is not part of the common prefix, to push n into the stack. The action is repeated for every node from the sort merged input lists. Later, Xu et al. propose a more efficient algorithm called Indexed Stack to find the LCA results of a query [119].

XSEarch [38] introduces the concept of *interconnection* to find meaningfully related nodes as search results. The intuitive definition is as below: For a given keyword query Q="k_1,k_2,...,k_m", suppose there exists node n_i such that n_i directly contains keyword k_i either in its value or its label for i∈[1,m], then n_1 up to n_m are said to be interconnected if along the path from v to each n_i, there are no two distinct nodes with the same node name. The LCA of n_1 up to n_m is counted as a result. E.g. consider a query Q = "John, tennis" on the XML data tree in Figure 2.2. By LCA semantics, node customers is returned; however, it should not be a meaningful answer because the two nodes that contain the above two keywords are descendants of different customer. The rational behind is that, it tries to constrain the answer to be a single real-world entity containing all query keywords; however, it may miss some relevant results as user's search concern may involve more than one entity. Li et al. proposed a new indexing way to find the above matching results in a more efficient way [73].

Subsequently, SLCA (smallest LCA [79, 118]) is proposed to further constrain the LCA results of a query, i.e. to find the smallest LCAs that do not contain other LCAs in their subtrees. In particular, Li et al. [79] incorporate SLCA in XQuery and propose a so called 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 XQuery queries. However, when users have no idea of the schema, they can still use keyword queries with Schema-Free XQuery. [79] also proposes a stack-based sort merge algorithm to compute SLCA results, which is similar to the stack algorithm in XRANK [52].

XKSearch [118] 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 cannot contribute to the final results can be effectively skipped. In contrast, Scan Eager algorithm scans all inverted lists for cases when the inverted lists of all query keyword have similar sizes. Experimental evaluation shows the superiority of these two algorithms as compared to the stack-based algorithm in [79]. 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. [104] make a further effort to improve the efficiency of computing SLCAs.

It discovers the fact that we may not need to completely scan the shortest 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 [118] 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 [118] during SLCA computation. Although algorithms in Multiway SLCA [104] have the same theoretical time complexity as Indexed Lookup Eager algorithm in [118], experimental results show the superiority of Multiway-based algorithms. In addition, [104] 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. [54] proposed 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 [80].

XSeek [80] generates the return nodes which can be explicitly inferred by keyword match pattern and the concept of entities in XML data. However, it addresses neither the ranking problem nor the keyword ambiguity problem. Besides, it relies on the concept of entity (i.e. object class) and considers a node type t in DTD as an entity if t is "*"-annotated in DTD. As a result, customer, interest, book in Figure 2.4, are identified as object classes by XSeek. However, it causes the multi-valued attribute to be mistakenly identified as an entity, causing the inferred return node not as intuitive as possible. E.g. interest is not intuitive as entities. In fact, the identification of entity is highly dependent on the semantics of the underlying XML data rather than its DTD, so it usually requires the verification and decision from database administrator. Therefore, the adoption of entities for keyword search should be optional although this concept is very useful. Based on SLCA, Liu et al. further proposed an axiomatic way to decide whether a result is relevant to a keyword query [81], in term of two properties called monotonicity and consistency with respect to the XML data and query, as shown below:

- (*Data Monotonicity*) If a new node is inserted into the data, then the data content becomes richer, thus the number of query results should be (non-strictly) monotonically increasing.
- (*Query Monotonicity*) If a new keyword is added to the query, then the query becomes more restrictive, therefore the number of query results should be (non-strictly) monotonically decreasing.
- (*Data Consistency*) After a new node n is inserted into the data, then each additional subtree that becomes (part of) a query result should contain n.
- (*Query Consistency*) If a new keyword k is added to the query, then each additional subtree that becomes (part of) a query result should contain at least a match to k.

We can find that among all the matching semantics proposed so far, no one has explicitly addresses the problem of identifying the target that a user query intends to search for. That motivates our works in this thesis.

2.5.2 Result Ranking on XML Data Tree Model

Result ranking is another crucial issue in building an effective XML keyword search framework. XRANK [52] presents a ranking method to rank subtrees rooted at LCAs. XRANK extends the well-known Google's PageRank [27] 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. In contrast, our work [16] in this thesis is built at sub-tree level, which coincides with the fact that the answer to a keyword query should be a subtree rooted at an appropriate node rather than the LCA or SLCA node itself. In addition, no empirical study is done to show the effectiveness of its ranking function. XSEarch [38] adopts a variant of LCA, and combines a simple TF*IDF IR ranking with size of the tree and the node relationship to rank results; but it requires users to know the XML schema information, causing limited query flexibility. Most recently, EASE [74] proposes a unified graph index to handle keyword search on heterogenous data which includes unstructured, structured and semi-structured data. It combines IR ranking and structural compactness based DB ranking to fulfill keyword search on heterogenous data. However, they either don't take the hierarchical structure of XML data into consideration in their ranking function design, or the granularity of ranking function designed is at node level rather than subtree level. Another important problem during result ranking is to identify the search target of an XML keyword query, which is initialized by our work [16], which utilize the statistics of underlying database to issue a formula to compute the confidence of each node type in XML data as the potential search targets.

For the ranking methods in IR field, TF*IDF similarity [101], which is originally designed for flat document retrieval, is insufficient for XML keyword search due to XML's hierarchical structure and the presence of keyword ambiguities mentioned in [16]. The details of TF*IDF will be introduced in section 3.2. Several proposals for XML information retrieval suggest to extend the existing XML query languages [46, 13, 106] or use XML fragments [30] to explicitly specify the search intention for result retrieval and ranking.

XRANK [52] relies on a static processing of ranking score computation, while our work [20] (as described in chapter 4) employs a dynamic computation. Some previous methods such as ObjectRank [14] and HITS [66] also employ the dynamic ranking methods, but in contrast, our approach (as later shown in Chapter 4) takes advantage of the co-occurrence of query keywords in a single logical result while they cannot. As a result, the relevance rank computed by HITS and ObjectRank may be biased to keywords which are frequent among objects, especially when there are three or more keywords.

2.5.3 Improving User Search Experience

Besides the design of search semantics, efficient evaluation method and result ranking scheme, there are many other issues that need consideration in building a keyword search engine over semi-structured data. One important issue is how to help users analyze the results and offer them a friendly search experience. In recent literature, two works are worth mentioning.

The first one is about result snippet generation [58], which is used to complement the result ranking scheme to effectively handle user searches, which are inherently ambiguous and whose relevance semantics are difficult to assess. The authors first regulate four guidelines for a desired result snippet: (1) a result snippet should be *self-contained* so that users can understand it; (2) different result snippets should be *distinguishable* from each other, so that users can differentiate the results from their snippets with little effort; (3) a snippet should be *representative* to the query result, thus users can grasp the essence of the result from its snippet; (4) a result snippet should be small so that users can quickly browse several snippets. The first three goals are conflicting with the last goal, as the larger the snippet size is, the more information it can contain. Then the authors prove that, the decision problem of selecting as many features as possible to form a snippet given an upper bound of the snippet size is NP-complete (by reducing the classical set cover problem [39] to it). Besides, they quantify the above four goals by a careful design of scoring metrics, and design a greedy algorithm to achieve an efficient generation of semantic result snippets.

The second work is about result differentiation. Most recently, Liu et al. [82] raise the issue of automatically differentiating the search results of an XML keyword query, aiming to save user efforts in manually investigating and comparing potentially large results. They first define what a differentiation feature set (denoted as DFS) should be for a search result, and propose three desired features for a good DFS, which are *differentiability*, *validity* and *small size*. Then they prove the NP-hardness for the problem of constructing DFSs that are valid and maximally differentiate a set of results within a size bound. Instead, they adopt two relaxed constraints and propose a dynamic programming solution to achieve the local optimality for the DFS construction. Experimental study has validated the practical effectiveness and efficiency of their approach.

Lastly, there is also an emerging research effort in answering the top-k keyword search over XML database. [35] is the first to study the problem of finding the top-k results for XML keyword search based on the LCA matching semantics with a given result ranking scheme. The authors first mention that existing approaches designed for efficient result evaluation (such as [118, 119, 104]) cannot be adapted to support top-k query in XML keyword search, because they determine and generate the results in document order instead of the order of ranking scores of results. Then they discuss why existing top-k algorithms designed for relational databases such as the famous Threshold

Algorithm (TA) [43] cannot be easily applied to the context of XML keyword search, because in finding the results by SLCA semantics (which has been one of the most widely adopted matching semantics so far) which is a subset of LCA semantics, it not only needs to compute LCAs of individual nodes, but also needs to prune those that are already ancestors of other LCAs, which indeed is a complex computation; thus directly applying TA's intuition may easily lose the optimization of the semantic pruning and makes it very expensive. The authors propose a novel way to combine the semantic pruning and top-k processing to support top-k keyword search over XML data, which reduces the keyword query evaluation to a series of relational joins and adapt the idea of traditional top-k join in relational database to XML database in a seamless way.

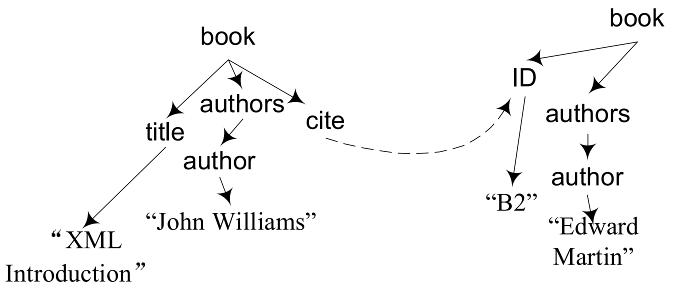


Figure 2.6: Reduced subgraph for Q="XML, John, Martin" on Figure 2.4's XML data

2.6 Keyword Search on Digraph Model

By capturing the IDRef edges in XML data, it is assured that more relevant results may be found. Similar to tree model, the most important semantics is to find the smallest substructure of the XML data containing all query keywords [37, 65, 53, 57]. The key concept in the existing semantics is called *reduced subgraph* ([37]). An informal definition is as below: given an XML graph G and a list KS of keywords, a connected subgraph G' of G is a reduced subgraph w.r.t KS if G' contains all keywords in KS, and no proper subgraph of G' contains all these keywords. For example, given a query Q="XML, John, Martin" issued on the XML document shown in Figure 2.4, a possible reduced subgraph result for Q is shown in Figure 2.6.

However, the cost of finding all such G' ranked by size is intrinsically expensive as the reduced tree problem on graph is as hard as NP-complete [77]. Therefore, in *digraph* data model, there has not been any work that can resolve the exploration of reduced subgraphs in an efficient way. Previous approaches are heuristics-based in order to reduce the search space as much as possible. In particular, Li et al. [77] propose a method of retrieving and organizing web pages by *In formation Unit* and show the reduction from minimal reduced tree problem to the NP-complete Group Steiner Tree problem on graphs. BANKS [65] uses bidirectional expansion heuristic algorithms to search as small portion of graph as possible and ranks resulted reduced-trees in approximate order of result generation during the expansion. Given a good estimation of “important” nodes and edges where the expansion starts with, BANKS seems to work well for a small number of results. However, BANKS still has the inherited limitation of slow query response when a large number of results are required, because it requires the entire visited graph in memory. BLINKS [53] improves it by proposing a bi-level index to prune and accelerate searching for top-k results in digraphs, with the tradeoffs in index size and maintenance cost. Its main idea is to maintain indices to keep the shortest distance from each keyword to all nodes in the entire database graph. XKeyword [57] uses schema information to reduce search space, but its query evaluation is based on the method of DISCOVER [56] built on relational database, so it still suffers from the efficiency problem. Besides, it needs to compute candidate networks and thus is constrained by schemas.

In a nut shell, it is inefficient to treat XML data (with IDref edges) as a directed graph and apply the above approaches in finding the reduced subgraphs, due to two reasons. *First*, the number of all reduced subgraphs may be exponential in the size of G. In contrast, the number of LCA subtrees (in tree data model) is bounded by the size of the given XML tree. Different reduced subgraphs present different connected relationships in real world, and most of them cannot be trivially judged as redundant results. *Second*, if we adopt enumerating results by increasing the sizes of reduced subgraphs (as smaller subtree usually indicates a closer connection according to general assumption of XML keyword proximity search), this problem can be NP-hard; the well-known *Group Steiner tree* problem [33] for graph can be reduced to it [77]). Although there are a multitude of polynomial time approximation approaches such as [33, 48] that can produce solutions with bounded errors for *minimal Steiner* problem, they require an examination of the entire graph. These approaches are not desirable since the overall graph of XML keyword search is often very large.

2.7 Keyword Search over Relational Database

Keyword search over relational database is also related to this thesis. Although it can be generalized to the keyword search over directed graph, there are many efforts targeted specifically to relational database. In particular, several prototypes such as DBXplorer [10], DISCOVER [56], DISCOVER-II [55], BANKS [24] and BANKS-II [65] have been proposed, and efficiency has been the main focus of these works. DBXplorer generates trees of tuples that are connected through primary key-foreign key relationship and contain all query keywords. BANKS identifies connected trees in a labeled graph by using an approximation of the Steiner tree problem. DISCOVER-II [55] considers the problem of keyword proximity search by *disjunctive* semantics, as opposed to DISCOVER [56] which adopts conjunctive semantics. In [65], a bidirectional approach is proposed to further improve the search efficiency, but it suffers from identifying Steiner trees from the whole graph due to the difficulty in identifying structural relationships through inverted indices. Ding et al. [41] designed a dynamic programming method to improve the efficiency in identifying steiner trees. Guo et al. proposed a data topology search approach to find meaningful structures from richer structural data such as complex biological databases [51]. He et al. proposed a partition-based method to improve search efficiency with a novel bi-level index [53]. Markowetz et al. studied the problem of keyword search over relational data streams [89]. Luo et al. proposed a new ranking method that extends state-of-the-art IR ranking function to the resulted joined database tuples [87]. Recently, Li et al. propose an efficient and adaptive keyword search method called EASE [74] to index and query large collections of *heterogeneous* data, including unstructured, structured and semi-structured data. They first model unstructured, structured and semi-structured data as graph G, where nodes represent documents, tuples and XML elements respectively, and edges represent hyper-links, primary-key-foreign-key relationship and parent-child relationship (or IDREF) respectively. They introduce the concept of r-radius steiner graph[1], and then reduce the problem of finding the matching results of a query to a so called *r-Radius Steiner Graph Problem*: given a graph G and an input query K, we try to find all the r-radius steiner graphs in G, which contain all or part of the input keywords in K ranked by its relevance w.r.t K. As opposed to the steiner tree based methods which need to maintain the whole graph in memory, the authors novelly propose to cluster the r-radius graphs and then partition the whole graph based on clusters to facilitate identifying r-radius graphs without maintaining the whole graph. Accordingly, they propose an effective cost metric to guarantee a high-quality and meaningful cluster to be found.

All the above works focus on answering a keyword query over a single relational database. As the distributed databases emerge in many real world applications, it is necessary to support keyword-based querying over distributed databases. A specific research problem is about how to select the top-k database sources in peer-to-peer context, in order to avoid the high cost of searching in large number of potentially irrelevant databases in such systems. [120] is the first work that addresses this issue by summarizing the relationships between keywords in the underlying databases. Its main idea is to build an entry for each pair of keywords for each database, recording the frequencies of

[1]Simply speaking, it is a sub-graph of G whose radius is not greater than r, where r is the minimum of the max distance from one node u to any other node, for each node u in G.

co-occurrences of the two terms at different *distances*, where *distance* is defined as the the number of join operations in a joining sequence of tuples. When a user query q is issued, the similarity between q and each database is computed by using the entries of all possible keyword pairs in q. However, it exploits only the binary relationships between keyword terms to eliminate the non-promising databases, it cannot be easily fit to the IR-style ranking measures, and may even produce many false positive results for queries where all pairs of keywords are related but no explicit join sequence connecting these binary relationships within a single result. In order to address the above limitations, [108] follows up this problem and proposes to summarize each database as a *keyword relationship graph*, where nodes represent terms and edges describe relationships between them. It also proposes an IR-style scoring method to measure the importance of nodes and edges, and an algorithm to estimate the potential of a join solution containing all query keywords for selecting the top-k databases as a result.

2.8 Keyword Query Refinement

In keyword search over any type of data, user queries may contain irrelevant or mismatched terms, typos etc, which may easily lead to empty or nonsensical results. Therefore, it is demanding to provide an automatic query refinement strategy to relieve as many user efforts as possible along the way to find their desired results.

2.8.1 Keyword Query Refinement in IR Field

Keyword query refinement in information retrieval field can be divided into two main spectrums: (1) A fully automatic refinement [114, 64, 50], which modifies and subsumes terms to queries according to a thesaurus or terms in documents, with no intervention on user part; the thesaurus itself may automatically or manually be generated. (2) An interactive refinement [100], such as relevance feedback requiring user to manually identify relevant documents whose terms are removed from (or added to) the query.

Common refinement tasks include query expansion, query word deletion and word substitution. Query word substitution can be further classified into spelling error correction, synonym substitution, acronym expansion, merging of words, split of words etc.

In particular, most existing works focus on designing specific solution (which applies a particular type of refinement operation) rather than an all-purpose solution. In *query expansion*, one adds new terms to the query to overcome the term mismatch problem [29, 97], assuming no error exists in the queries submitted by users. The so-called global analysis and local analysis are usually used in query expansion to find related queries to the user to enhance his/her search experience [22, 45, 68]. In *query substitution* [64], its key idea is to replace current query with a new query that can improve search relevance by learning from search log data. In *query word deletion*, queries with no matches can have words deleted till a match is obtained, and the prediction is done by tracing the users' search modifications [63]. Lastly, *spelling error correction* [75] is achieved by using a Maximum Entropy (ME) model as well as the source channel model, and then utilizing the distributional similarities between the query word and its correction candidate as features in the ME model. Two models, linear classification and linear regression, are trained by using labeled data and employed in the substitution and query suggestion is that the former is used to consider inter-query relations, while the latter considers intra-query relations.

Regarding the generation of the refinement rules, such information can be obtained from the existing dictionary such as WordNet [44], document mining for synonyms [42], query log analysis [63, 47] or manual annotation.

2.8.2 Keyword Query Cleaning in Relational Database

In the context of relational database, keyword query cleaning consists of rewriting the user query, segmenting the keywords, matching each segment to database items, and tagging the segments by their meta-data information. [96] is the first work studying this problem. It introduces a preprocessing stage to clean the raw text and extract a high quality keyword query, in order to reduce the search space of a keyword query. This method, while pioneering, has two major drawbacks: (1) The cleaned query is not guaranteed to have matching results in database. (2) Their methods to rank the cleaned queries do not consider the matching results of those queries in database, thus significantly hampering effectiveness. In our work as introduced in Chapter 5 later, our focus is to build a refinement solution that circumvents all the pitfalls encountered above by efficiently retrieving the real matching results of the refined queries.

Alternatively, Pu [95] proposed to solve the above problem by building a probabilistic model. In particular, the author proposed to model user queries by constructing a generative probabilistic Hidden Markov Model [98] (HMM), and reduce the problem of finding the optimal query cleaning to the problem of finding the most likely path of the constructed HMM. In contrast to [96], the HMM-based approach can provide more theoretical explanation as to why certain cleaned queries are better, offer extra flexibility of being adaptive to user feedback and existing query logs, and facilitate the extension of the cost model to incorporate new ways of cleaning a keyword query. At the same time, Yu et al. studied a particular problem in keyword query cleaning, i.e. query segmentation which groups the nearby keywords in a query to segments [121]. Instead of the above HMM, the authors present a principled approach based on another statistical model called conditional random fields (CRF) [69, 94]. Similar to [95], this approach also can be learned from past search history and adapted to user feedback.

2.8.3 Keyword Query Refinement in XML Retrieval

XML retrieval is an important new area for the application of IR methods. A representative framework for the evaluation of XML retrieval methods is INEX (INitiative for the Evaluation of XML Retrieval) [4], where the search quality is judged by large-scale user studies.

In the field of XML retrieval, a dominant refinement strategy is query expansion, which adds new terms highly correlated with the initial query to enhance the result quality. In particular, [90] adopts a pseudo-feedback to choose the expanded terms from the top ranked results of the initial query; [92] expands the initial query based on the feedback of result relevance from user; TopX [107] generates the potential expanded terms by using a thesaurus database WordNet [44]. Another related direction is to transform an XML keyword query into a set of structured twig queries based on the schema of XML data [93]. However, it is time consuming, as a keyword query may derive many structured queries. [93] achieves it by assuming the user has provided structural clues to indicate the type of XML elements he is interested in and no keyword ambiguity occurs in the query; however, user is required to learn the (possibly very complex) structure of XML data, which defeats the purpose of keyword search. Moreover, in our opinion, the best way for user to judge the quality of the structured query is to look at its matching result rather than the query itself.

In contrast, our work (as described in chapter 5) achieves an automatic query refinement at two levels: *first*, it automatically and quickly peruses the XML document and makes appropriate modifications on the query by exploiting the refinement rules; *second*, it can automatically generate the query results for both the original and refined query within a one-time scan of related keyword inverted lists. This is convenient for users to quickly decide which list of results meets their search needs without a second try. Moreover, we propose a general scoring model to qualify the refined query candidates, by accommodating for different kinds of refinement operations including term deletion, merging, split and substitution.

Table 2.1 summarizes a classification of all the above works related to keyword query processing and keyword query refinement in XML search, web search and relational database.

Table 2.1: Summary of Related Works

Data Model	**Matching Semantics**	**Result Ranking**	**Result Computation**	**Search Experience**
XML Tree	LCA[52], SLCA[118] XSeek[80], ELCA[119] GDMCT[54], Interconnected Semantics[38, 73]	XRANK[52] XSEarch[118]	XKSearch[118] Multiway SLCA[104]	snippet generation[58] result differentiation[82] top-k keyword search[35]
XML Graph	Reduced Subgraph [37, 65, 53, 57]	EASE[74]	BANKS[65] BLINKS[53] XKeyword[57] DISCOVER[56]	
RDMBS	Reduced Subgraph [37, 65, 53, 57]		DBXPlorer[10] BLINKS[53] DISCOVER[56] DISCOVER-II[55] BANKS[24] BANKS-II[65]	

Keyword Query Refinement	**Web Search**	**XML Search**	**RDBMS**	
	Automatic refinement [114, 64, 50] Interactive way [100]	Pseudo-feedback for query expansion [90, 92]	Keyword query cleaning[96]	

CHAPTER 3

EFFECTIVE KEYWORD SEARCH OVER XML DATA TREE

3.1 Introduction

The extreme success of web search engines makes keyword search the most popular search model for users. As XML is becoming a standard in data representation, it is desirable to support keyword search in XML database. It is a user friendly way to query XML database since it allows users to pose queries without the knowledge of complex query languages and the database schema.

As most XML data is modeled as a labeled tree [118, 38, 80, 73, 119], in this chapter our main solution is built for the XML data tree model. For instance, In Figure 3.1, the XML data stores the information of a bookstore about the books sold and their frequent customers. Unless otherwise specified, we do not consider the dotted reference edges in the XML data throughout this chapter; instead, we treat it as a labeled tree in designing solutions for result matching and ranking.

Effectiveness in term of result relevance is the most crucial part in keyword search, which can be summarized as the following three issues in XML field.

Issue 1: It should be possible to effectively identify the type of target node(s) that a keyword query intends to search for. We call such target node as *search for node*.

Issue 2: It should be possible to effectively infer the types of condition nodes that a keyword query intends to search via. We call such condition nodes as *search via nodes*.

Issue 3: It should be possible to rank each query result in consideration of the above two issues.

The first two issues address the search intention problem, while the third one addresses the relevance based ranking problem w.r.t. the search intention. Regarding Issue 1 and Issue 2, XML keyword queries usually have ambiguities in interpreting the search for node(s) and search via node(s), due to three reasons below.

- **Ambiguity 1**: A keyword can appear both as an XML tag name and as a text value of some other nodes.
- **Ambiguity 2**: A keyword can appear as the text values of different types of XML nodes and carry different meanings.
- **Ambiguity 3**: A keyword can appear as an XML tag name in different contexts and carry different meanings.

Ambiguity 1 is syntactic; while Ambiguity 2 and 3 are semantic. For example see the XML document in Figure 3.1, keywords *customer* and *interest* appear as both an XML tag name and a text value (e.g. value of the title for book B1); *art* appears as a text value of interest, address and name node; *name* appears as the tag name of the name node of both customer and publisher.

Regarding Issue 3, the search intention for a keyword query is not easy to determine and can be ambiguous, because the search via condition is not unique; so how to measure the confidence of each search intention candidate, and rank the individual matches of all these candidates are challenging.

Although many research efforts have been conducted in XML keyword search [38, 52, 118, 80, 54, 81], none of them has well resolved the above three issues yet. For instance, one widely adopted approach so far is to find the smallest lowest common ancestor (SLCA) of all keywords [118]. Each SLCA result of a keyword query contains all query keywords but has no subtree which also contains all the keywords.

In particular, regarding Issue 1 and 2, SLCA may introduce answers that are either irrelevant to user search intention, or answers that may not be meaningful or informative enough. For example, when a query "Jim Gray" that intends to find Jim Gray's publications on DBLP [70] is issued, SLCA returns only the *author* elements containing both keywords. Besides, SLCA also returns publications written by two authors where "Jim" is a term in the first author's name and "Gray" is a term in the second author, and publications with *title* containing both keywords. It is reasonable to return such results because search intention may not be unique; but they should be given a lower rank, as they are not matches of the major search intention. Regarding Issue 3, no existing approach has studied the problem of relevance oriented result ranking in depth yet. Moreover, they don't perform well on pure keyword query when the schema information of XML data is not available [80]. The actual reason is, none of them can solve the above keyword ambiguity problems, as demonstrated by the following example.

Example 3.1. *Consider a keyword query "*customer interest art*"*[1] *issued on the bookstore data in Figure 3.1, and most likely it intends to find the customers who are interested in art. If adopting SLCA, we will get 5 results, which include the title of book* $B1$ *and the customer nodes with IDs from* $C1$ *to* $C4$ *(as these four customer nodes contain "customer", "interest" and "art" in either the tag names or node values) in Figure 3.1.*

[1]Throughout this chapter, a keyword query is in form of "k_1 k_2, ...,k_n", where neighboring keywords are separated by a single space.

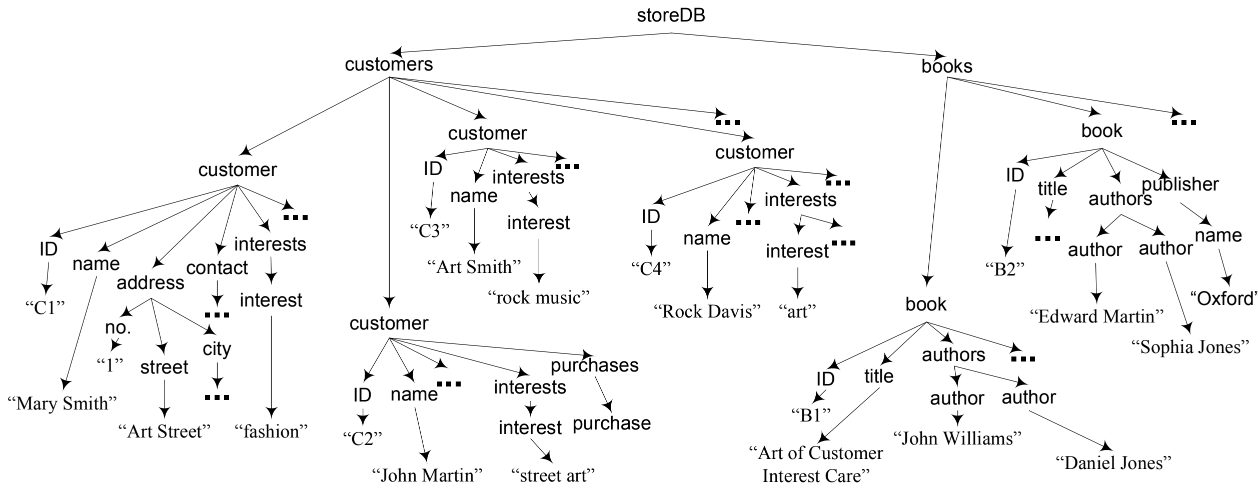


Figure 3.1: Portion of data tree for an online bookstore XML database

Since SLCA cannot well address the search intention, all these 5 results are returned without any ranking applied. However, only C4 *is desired which should be put as the top ranked one, and* C2 *is less relevant, as his interest is "street art" rather than "art", while* C1 *and* C3 *are irrelevant.* □

Inspired by the great success of IR approach on web search (especially its distinguished ranking functionality), we aim to achieve similar success on XML keyword search, to solve the above three issues without using any schema knowledge.

The main challenge we are going to solve is how to extend the keyword search techniques in text databases (IR) to XML databases, because the two types of databases are different. *First*, the basic data units in text databases are flat documents. For a given query, IR systems compute a numeric score for each document and rank the document by this score. In XML databases, however, information is stored in hierarchical tree structures. The logical unit of answers needed by users is not limited to individual leaf nodes containing keywords, but a subtree instead. If we are not aware of which kind of subtree should be an appropriate information unit for user's search intention at first, while view subtrees of different levels as a document and simply apply existing TF*IDF-like approaches to rank those subtrees, it will cause two problems: (1) It incurs an inefficient computation to return many more results which even have a containment relationship with each other, while it is more difficult for user to identify which (kind of) subtrees match his search target; (2) the index size for the keyword inverted list will be much larger, as a same keyword appearing in a subtree T also appears with the subtrees rooted at T's ancestors. *Second*, unlike text database, it is difficult to identify the (major) user search intention in XML data, especially when the keywords contain ambiguities mentioned before. The search intention in the context of XML search includes the search target part which never appears in flat document search. *Third*, effective ranking is a key factor for the success of keyword search. There may be dozens of candidate answers for an ordinary keyword query in a medium-sized database. For instance, in Example 3.1, five subtrees of the XML tree in Figure 3.1 can be the query answers, but they are not equally useful to user. Due to the difference in basic answer unit between document search and database search, in XML database we need to assign a single ranking score for each subtree of certain category with a fitting size, in order to rank the answers effectively.

Statistics is a mathematical science pertaining to the collection, analysis, interpretation or explanation of data; it can be used to objectively *model a pattern* or *draw inferences* about the underlying data being studied. Although keyword search is a subjective problem that different people may have different interpretations on the same keyword query, statistics provides an objective way to distinguish the major search intention(s).

It motivates us to model the search engine as a domain expert who automatically interprets user's all possible search intention(s) through analyzing the statistics knowledge of underlying data. As a result, we propose an IR-style approach which well captures XML's hierarchical structure, and works well on pure keyword query independent of any schema information of XML data. A search engine prototype called XReal is implemented to achieve effective identification of user search intention and relevance oriented ranking for the search results in the presence of keyword ambiguities.

Example 3.2. *We use the query in Example 3.1 again to explain how XReal infers user's desired result and puts it as a top-ranked answer. XReal interprets that user desires to search for* customer *nodes, as all three keywords have high frequency of occurrences in customer nodes. Similarly, since keywords "interest" and "art" have high frequency of occurrences in subtrees rooted at* interest *nodes, it is considered with high confidence that this query wants to search via* interest *nodes, and incorporates this confidence into our ranking formula. Besides, customers interested in "art" should be ranked before those interested in (say) "street art". As a result, C4 is ranked before C2, and further before customers with address in "art street"(e.g. C1) or named "art" (e.g. C3).* □

To our best knowledge, we are the first that exploit the statistics of underlying XML database to address search intention identification, result retrieval and relevance oriented ranking as a single problem for XML keyword search. Our contributions in this chapter are summarized as follows:

1. This is the first work that addresses the keyword ambiguity problem and search target identification.
2. We define our own XML TF (term frequency) and XML DF (document frequency), which are cornerstones of all formulae proposed later.
3. We propose three important guidelines in identifying the user desired *search for* node type, and quantify the confidence of a certain node type to be a desired *search for* node based on the guidelines.
4. We design formulae to compute the confidence of each candidate node type as the desired *search via* node to model natural human intuitions, in which we take into account the pattern of keywords' co-occurrence in query.
5. We propose a relevance oriented ranking scheme called *XML TF*IDF similarity* which can capture the hierarchical structure of XML and resolve Ambiguity 1-3 in

a heuristic way. Besides, the popularity of query results is designed to distinguish the results with comparable relevance scores.

6. We implement the proposed techniques in a keyword search engine prototype called XReal. Extensive experiments are conducted to show its effectiveness, efficiency and scalability.

The rest of the chapter is organized as follows. We present the preliminaries in Section 3.2. Section 3.3 infers user search intention. Section 3.4 discusses the ranking scheme. Section 3.5 presents the search algorithms. Experimental study is discussed in Section 3.6 and we summarize in Section 3.7.

3.2 Preliminaries

3.2.1 TF*IDF Cosine Similarity

TF*IDF (Term Frequency * Inverse Document Frequency) cosine similarity [101] is one of the most widely used approaches to measure the relevance of keywords and document in keyword search over flat documents. We first review its basic idea, then address its limitations for keyword search over XML data. The main idea of TF*IDF is summarized in the following three rules.

- **Rule 1**: A keyword (or term[2]) appearing in a few documents may be regarded as being more important than a keyword appearing in many.

- **Rule 2**: A document with more occurrences of a query keyword may be regarded as being more important for that keyword than a document that has less.

- **Rule 3**: A normalization factor is needed to balance between long and short documents, as Rule 2 discriminates against short documents which may have less chance to contain more occurrences of keywords.

[2]We use *keyword* and *term* interchangeably in this chapter.

To combine the intuitions in the above three rules, the TF*IDF similarity is designed:

$$\rho(Q, D) = \frac{\sum_{k \in Q \cap D} W_{Q,k} * W_{D,k}}{W_Q * W_D} \tag{3.1}$$

where Q represents a query, D represents a flat document and k is a keyword appearing in both Q and D. A larger value of $\rho(Q, D)$ indicates Q and D are more relevant to each other. $W_{Q,k}$ and $W_{D,k}$ represent the weights of k in query Q and document D respectively; while W_Q and W_D are the weights of query Q and document D. Among several ways to express $W_{Q,k}$, $W_{D,k}$, W_Q and W_D, the followings are the conventional formulae:

$$W_{Q,k} = \ln\left(N/(f_k + 1)\right) \tag{3.2}$$

$$W_{D,k} = 1 + \ln\left(f_{D,k}\right) \tag{3.3}$$

$$W_Q = \sqrt{\sum_{k \in Q} W_{Q,k}^2} \tag{3.4}$$

$$W_D = \sqrt{\sum_{k \in D} W_{D,k}^2} \tag{3.5}$$

where N is the total number of documents, and document frequency f_k in Formula 3.2 is the number of documents containing keyword k. Term frequency $f_{D,k}$ in Formula 3.3 is the number of occurrences of k in document D.

$W_{Q,k}$ is monotonically decreasing w.r.t. f_k (*Inverse Document Frequency*) to reflect Rule 1; while $W_{D,k}$ is monotonically increasing w.r.t. $f_{D,k}$ (*Term Frequency*) to reflect Rule 2. The logarithm in Formula 3.2 and 3.3 are designed to normalize the raw document frequency f_k and raw term frequency $f_{D,k}$. Finally, W_Q and W_D are increasing w.r.t. the size of Q and D, playing the role of normalization factors to reflect Rule 3.

However, the original TF*IDF is inadequate for XML, because it is not able to fulfill the job of search intention identification or resolve keyword ambiguities resulted from

XML's hierarchical structure, as Example 3.3 shows.

Example 3.3. *Suppose a keyword query* "art" *is issued to search for* customer*s interested in "art" in Figure 3.1's XML data. Ideally, the system should rank* customer*s who do have "art" in their nested* interest *nodes before those who do not have. Moreover, it should give customer* A *who is only interested in art a higher rank than another customer* B *who has many interests including art (e.g.* C4 *in Figure 3.1).*

*However, it causes two problems if directly adopting the original TF*IDF to XML data. (1) If the structure in* customer *nodes is not considered, customer* A *may have a lower rank than* B *if* A *happens to have more keywords in its subtree (analog to long document in IR) than* B*. (2) Even worse, suppose a customer* C *is not interested in "art" but has* address *in "art street". If C has less number of keywords than A and B in the underlying XML data, then C may have a higher rank than A and B.* □

3.2.2 Data Model

We model XML document as a rooted, labeled tree, such as the one in Figure 3.1. Our approach exploits the prefix path of a node rather than its tag name for result retrieval and ranking. Note that the existing works [80, 73] rely on DTD while our approach works without any XML schema information.

Definition 3.1. *(**Node Type**) The type of a node n in an XML document is the prefix path from root to n. Two nodes are of the same node type if they share the same prefix path.*

In Definition 3.1, the reason that two nodes need to share the same prefix path instead of their tag name is, there may be two or more nodes of the same tag name but of different semantics (i.e. in different contexts) in one document. E.g. in Figure 3.1, the *name* of publisher and the *name* of customer are of different node types, which are storeDB/books/book/publisher/name and storeDB/customers/customer/name respectively. Besides, when XML database contains multiple XML documents, the node type should also include the document name.

To facilitate our discussion later, we use the tag name instead of the prefix path of a node to denote the node type in all examples throughout this chapter. Besides, in order to separate the content part from leaf node, we distinguish an XML node into either a *data node* or a *structural node*.

Definition 3.2. *(**Data Node**) The text values that are contained in the leaf node of XML data and have no tag name is defined as a* data node.

Definition 3.3. *(**Structural Node**) An XML node labeled with a tag name is called a* structural node. *A structural node that contains other structural nodes as its children is called an* internal node*; otherwise, it is called a* leaf node.

In this chapter, we do not consider the case that an *internal node* n contains both *data nodes* and *structural nodes*, as we can easily avoid it by adding a dummy structural node with a tag name say "*value*" between n and the data nodes during node indexing without altering the XML data.

With the above two definitions, the value part and structure part of the XML data is separated. For instance, within the subtree of customer C1 in Figure 3.1, address is an *internal node*, street is a *leaf node*, and "Art Street" is a *data node*.

Definition 3.4. *(**Single-valued Type**) A structural node T is of* single-valued type *if each node of type T has at most one occurrence within its parent node.*

Definition 3.5. *(**Multi-valued Type**) A structural node T is of* multi-valued type *if some node of type T has more than one occurrence within its parent node.*

Definition 3.6. *(**Grouping Type**) An internal node T is defined as a* grouping type *if each node of type T contains child nodes of only one multi-valued type.*

XML nodes of *single-valued type* and *multi-valued type* can be easily identified when parsing the data. A node of single-valued (or multi-valued, or grouping) type is called a single-valued (or multi-valued, or grouping) node. For example in Figure 3.1, address is a *single-valued node*, while interest is a multi-valued node and interests is a grouping node for interest. In this chapter, for ease of presentation later, we assume every multi-valued node has a grouping node as its parent, as we can easily introduce a dummy grouping node in indexing without altering the data. Note a grouping node is also a single-valued node. Thus, the children of an internal node are either of same multi-valued type or of different single-valued types.

3.2.3 XML TF & DF

Inspired by the important role of data statistics in IR ranking, we try to utilize it to resolve ambiguities for XML keyword search, as it usually provides an intuitionistic and convincing way to model and capture human intuitions.

Example 3.4. *When we talk about* "art" *in the domain of database like Figure 3.1, we in the first place consider it as a value in* interest *of* customer *nodes or* category *(or* title*) of* book *nodes. However, we seldom first consider it as a value of other node types (e.g.* street *with value "Art Street").*

The reason for this intuition is, usually there are many nodes of interest *type and* category *type containing "art" in their text values, while "art" is infrequent in* street *nodes. Such intuition (based on domain knowledge) always can be captured by statistics of underlying data. Similarly, when we talk about* "interest"*, intuitionally we in the first place consider it as a node type instead of a value of the* title *of* book *nodes. Besides the reason that "interest" matches the XML tag* interest*, it can be explained from statistical point of view, i.e. all* interest *nodes contain keyword "interest" in their subtrees.* □

This example clearly shows the importance of statistics as formalized below.

Intuition 3.1. *The more XML nodes of a certain type T (and their subtrees) contain a query keyword k in either their text values or tag names, the more intuitive it is that that nodes of type T are more closely related to the query w.r.t. keyword k.*

Note that, the above intuition seems to contradict the Rule 1 in TF*IDF cosine similarity (in section 3.2.1). Indeed it is not, because Intuition 3.1 talks about the type of the node, which we can understand as which kind of "document" as the *search target* of a query, while the Rule 1 in TF*IDF talks about the possibility of a single document as a *result* of a query.

In this chapter, we maintain and exploit two important basic statistics terms, $f_{a,k}$ and f_k^T.

Definition 3.7. (XML TF) $f_{a,k}$*: The number of occurrences of a keyword k in a given* data node a *in XML data.*

Definition 3.8. (XML DF) f_k^T*: The number of T-typed nodes that contain keyword k in their subtrees in XML data.*

Here, $f_{a,k}$ and f_k^T are defined in an analogous way to term frequency $f_{d,k}$ (in Formula 3.3) and document frequency f_k (in Formula 3.2) used in the original TF*IDF similarity respectively; except that we use f_k^T to distinguish statistics for different node types, as the granularity on which to measure similarity in XML is a subtree rather than a document. Therefore, $f_{a,k}$ and f_k^T can be directly used to measure the similarity between a data node (with parent node of type T) and a query based on the intuitions of original TF*IDF. Besides, f_k^T is also useful in resolving ambiguities, as Intuition 3.1 shows. We will discuss how these two sets of statistics are used for relevance oriented ranking for XML keyword search in presence of ambiguities.

3.3 Inferring Keyword Search Intention

In this section, we discuss how to interpret the search intentions of a keyword query by linking the intuitively defined statistics for the query keywords in XML data and the pattern of keyword co-occurrence in the query.

3.3.1 Inferring the Node Type to Search For

The desired node type to search for is the first issue that a search engine needs to address in order to retrieve the relevant answers, as the search target in a keyword query may not be specified explicitly like in structured query language. Given a keyword query Q, a node type T is considered as the desired node to search for only if the following three guidelines hold:

Guideline 1: Query keywords coverage. T is intuitively related to every query keyword in Q, i.e. for each keyword k, there should be some (if not many) T-typed nodes containing k in their subtrees.

Guideline 2: Maximum relevant information. XML nodes of type T should be informative enough to contain enough relevant information.

Guideline 3: Minimum irrelevant information. XML nodes of type T should not be overwhelming to contain too much irrelevant information.

Guideline 2 prefers an internal node type T at a higher level to be the returned node, while Guideline 3 prefers that the level of T-typed node should not be very near to the root node. For instance let's refer to Figure 3.1: according to Guideline 2, leaf nodes of type interest, street etc. are usually not good candidates for desired returned nodes, as they are not informative. According to Guideline 3, nodes of type customers and books are not good candidates as well, as they are too overwhelming as a single keyword search result.

By incorporating the above guidelines, we define $C_{for}(T, Q)$, which is the confidence of a node type T to be the desired search for node type w.r.t. a given keyword query Q as follows:

$$C_{for}(T, Q) = \log_e(1 + \prod_{k \in Q} f_k^T) * r^{depth(T)} \quad (3.6)$$

where k represents a keyword in query Q; f_k^T is the number of T-typed nodes that contain k as either values or tag names in their subtrees (as explained in Section 3.2.3 to reflect Intuition 3.1); r is a reduction factor with range (0,1] and normally chosen to be 0.8, and $depth(T)$ represents the depth of T-typed nodes in document.

In Formula 3.6, the first multiplier (i.e. $\log_e(1 + \prod_{k \in Q} f_k^T)$) actually models Intuition 3.1 to address *Guideline 1*. Meanwhile, it effectively addresses *Guideline 3*, since the candidate overwhelming nodes (i.e. the nodes that are near the root) will be assigned a small value of $\prod_{k \in Q} f_k^T$, resulting in a small confidence value. The second multiplier $r^{depth(T)}$ simply reduces the confidence of the node types that are deeply nested in the XML database to address *Guideline 2*.

In addition, we use product rather than sum of f_k^T (i.e. $\prod_{k \in q} f_k^T$) in the first multiplier to combine statistics of all query keywords for each node type T. The reason is, the search intention of each query usually has a unique desired node type to search for, so using product ensures that a node type needs to be intuitively related to all query keywords in order to have a high confidence as the desired type. Therefore, if a node type T cannot contain all keywords of the query, its confidence value is set to 0. Furthermore, when the schema of XML data is available, the entity can be inferred (by adopting XSeek's methods [80]) and used to constrain the search for node candidates produced by Formula 3.6, as users are usually interested in real world entities. Similar to all the existing works [52, 118, 80, 54], in this chapter we assume each query keyword has at least one occurrence in the XML document being queried.

Example 3.5. *Given a query* "customer interest art"*, node type* customer *usually has* *high confidence as the desired node type to search for, because the values of three statistics* $f^{customer}_{\text{“customer”}}$, $f^{customer}_{\text{“interest”}}$ *and* $f^{customer}_{\text{“art”}}$ *(i.e. the number of subtrees rooted at* customer *nodes containing “customer”, “interest” and “art” in either nested text values or tags respectively) are usually greater than 1. In contrast, node type* customers *doesn’t have high confidence since* $f^{customers}_{\text{“customer”}} = f^{customers}_{\text{“interest”}} = f^{customers}_{\text{“art”}} = 1$. *Similarly, node type* interest *doesn’t have high confidence since* $f^{interest}_{\text{“customer”}}$ *usually has a small value. E.g. in Figure 3.1’s XML data,* $f^{interest}_{\text{“customer”}} = 0$. □

Finally, with the confidence of each node type being the desired type, the one with the highest confidence is chosen as the desired search for node, when the highest confidence is significantly greater than the second highest. However, when several node types have comparable confidence values, the system can either offer users a choice to decide the desired one, or do a search for each convincing candidate node by default (in case user rejects the offer). Regarding the threshold for comparableness judgement, we adopt the results from our empirical study: when the difference percentage of the scores of these node types is within 10%, they are viewed as “comparable”. Although not always fully automatic, our inference approach still provides a guidance for the system-user interaction for ambiguous keyword queries in absence of syntax. For example, the search engine can provide a guidance for users to browse and select their desired node type(s) in case that the keyword queries are ambiguous, before adopting the ranking strategy to rank the individual matches.

3.3.2 Inferring the Node Types to Search Via

Similar to inferring the desired search for node, *Intuition 3.1* is also useful to infer the node types to search via. However, unlike the search for case which requires a node type to be related to all keywords, it is enough for a node type to have high confidence as the desired search via node if it is closely related to some (not necessarily all) keywords, because a query may intend to search via more than one node type. For example, we can search for customer(s) named "Smith" and interested in "fashion" with query *"name smith interest fashion"*. In this case, the system should be able to infer with high confidence that name and interest are the node types to search via, even if keyword "interest" is probably not related to name nodes.

Therefore, we define $C_{via}(T, Q)$, which is the confidence of a node type T to be a desired type to search via as below:

$$C_{via}(T, Q) = \log_e(1 + \sum_{k \in Q} f_k^T) \tag{3.7}$$

where variables k, Q and T have the same meaning as those in Formula 3.6. Compared to Formula 3.6, we use sum of f_k^T instead of product, as it is sufficient for a node type to have high confidence as the search via node if it is related to some of the keywords. In addition, if all nodes of a certain type T do not contain any keyword k in their subtrees, f_k^T is equal to 0 for each k in Q, resulting in a zero confidence value, which is also consistent with the semantics of SLCA. Then, the confidence of each possible node type to search via will be incorporated into *XML TF*IDF similarity* (which will be discussed in Section 3.4.2) to provide answers of high quality.

3.3.3 Capturing Keyword Co-occurrence

In this section, we discuss the search via confidence for a *data node*. Although statistics provide a macro way to compute the confidence of a *structural node* type to search via, it alone is not adequate to infer the likelihood of an individual *data node* to search via for a given keyword in the query.

Example 3.6. *Consider a query "*customer name Rock interest Art*" searching for customers, each of whose name includes "Rock" and interest includes "Art". Based on statistics, we can infer that* name*-typed and* interest*-typed nodes have high confidence* *to* search via *by Formula 3.7, as the frequency of keywords "name" and "interest" are high in node types* name *and* interest *respectively. However, statistics is not adequate to help the system infer that the user wants "Rock" to be a value of* name *and "Art" to be a value of* interest*, which is intuitive with the help of keyword co-occurrence captured. Thus, if purely based on statistics, it is difficult for a search engine to differ* customer *C4 (with name "Art" and interest "Rock") from C3 (with name "Rock" and interest "Art") in Figure 3.1.* □

Motivated from the above example, the pattern of keyword co-occurrence in a query provides a micro way to measure the likelihood of an individual data node to search via, as a compliment of statistics. Therefore, for each query-matching data node v in XML data, in order to capture the co-occurrence of keyword k_t matching the node types of an ancestor node of v and keyword k matching a value in v (if they do exist in the query) in both query and XML data respectively, the following distances are defined.

The design of IQD is motivated by an observation: when users want to specify both the predicate k_t and its value k in a keyword query, they always put k_t and k close to each other, regardless of the search habits of different users, i.e. no matter whether k is specified *before/after* k_t for a particular user.

Definition 3.9. ***(In-Query Distance (IQD))*** *The* In-Query Distance $Dist_q(Q, k_t, k)$ *between keyword k and node type k_t in a query Q is defined as the absolute value of the position distance between k_t and k in Q; otherwise, $Dist_q(Q, k_t, k)$=∞.*

Note that, the above definition assumes there is no repeated k_t and k in a query Q, and the position distance of two keywords k_1 and k_2 in a query Q is the difference of k_1's position and k_2's position in the query.

Definition 3.10. ***(Structural Distance (SD))*** *The* Structural Distance $Dist_s(Q, v, k_t, k)$ *between k_t and k w.r.t. a data node v is defined as the depth distance between v and the* *nearest k_t-typed ancestor node of v in XML data ; $Dist_s(Q, v, k_t, k) = \infty$ if v does not have k_t-typed ancestor.*

IQD and SD are designed to capture the closeness of such node type k_t and keyword k in the input user query and underlying XML data resp. With intuition thinking, a data node v is favored when such k_t and k associated with it appear closely to each other in both the query and XML data, as stated in Intuition 3.2 and captured in Definition 3.11.

Intuition 3.2. *For a data node v, if the keyword k_t matching its associated node type and keyword k covered by v appear closely to each other in both the user query and XML data, it is more intuitive that v has a high confidence to be searched via. w.r.t keywords k_t and k.*

Definition 3.11. ***(Value-Type Distance (VTD))*** *$Dist(Q, v, k_t, k)$ between k_t and k w.r.t. a data node v is defined as* $\max(Dist_q(Q, k_t, k), Dist_s(Q, v, k_t, k))$.

In general, the smaller the value of $Dist(Q, v, k_t, k)$ is, it is more likely that Q intends to search via the node v with a value matching keyword k. Note that, any monotonic function can be applied in Definition 3.11 to fulfill such intuition, while *max* is one of them. Therefore, we define the confidence of a data node v as the node to search via w.r.t. a keyword k appearing in both query Q and v as follows.

$$C_{via}(Q, v, k) = 1 + \sum_{k_t \in Q \cap ancType(v)} \frac{1}{Dist(Q, v, k_t, k)} \tag{3.8}$$

Example 3.7. *Consider the query in Example 3.6 again, i.e. Q=“*customer name Rock interest Art*”. Let n_3 and i_3 represent the data nodes under* name *(i.e. Art Smith) and* interest *(i.e. rock music) of customer C3. Similarly, let n_4 and i_4 be the data nodes under* name *and* interest *of customer C4. Now, the in-query distance between name and Art is 3, i.e. $Dist_q(Q,\textsf{name}, Art) = 3$; $Dist_s(Q, n_3, \textsf{name}, Art) = 1$; as a result*

*$Dist(Q, n_3, \text{name}, Art) = 3$ and $C_{via}(Q, n_3, Art)$ = 4/3. Similarly, $C_{via}(Q, i_3, Rock)$ = 1; $C_{via}(Q, n_4, Rock)$ = 2; and $C_{via}(Q, i_4, Art)$ = 2. We find, the two predicates of customer C4 have a larger confidence to be searched via than those of customer C3. Intuitively, C4 should be more preferred than C3 as the result of Q. We will discuss how to incorporate these values into our XML TF*IDF similarity in section 3.4.2.* □

3.4 Relevance Oriented Ranking

In this section, we first summarize some unique features of keyword search in XML, and address the limitations of traditional TF*IDF similarity for XML. Then we propose a novel XML TF*IDF similarity, which incorporates the confidence formulae designed in Section 3.3, to resolve the keyword ambiguity problem in relevance oriented ranking.

3.4.1 Principles of Keyword Search in XML

Compared with flat documents, keyword search in XML has its own features. In order for an IR-style ranking approach to smoothly apply to it, we present three principles that the search engine should adopt.

Principle 1: When searching for XML nodes of desired type D via a *single-valued node type V*, ideally, only the values and structures nested in V-typed nodes can affect the relevance of D-typed nodes as answers, whereas the existence of other typed nodes nested in D-typed nodes should not. In other words, the size of the subtree rooted at a D-typed node d (except the subtree rooted at the search via node) shouldn't affect d's relevance to the query. □

Let us take a look at the following example to have an intuitionistic understanding of Principle 1.

Example 3.8. *When searching for customer nodes via street nodes using a keyword query "*Art Street*", a customer node (e.g. customer $C1$ in Figure 3.1) with the matching* *keyword* "street" *shouldn't be ranked lower than another customer node (e.g. customer* $C3$ *in Figure 3.1) without the matching keyword* "street"*, regardless of the sizes, values and structures of other nodes nested in* $C1$ *and* $C3$*. Note this is different from the original TF*IDF similarity that has strong intuition to normalize the relevance score of each document with respect to its size (i.e. to normalize against long documents).* □

Principle 2: When searching for the desired node type D via a *multi-valued node type* V', if there are many V'-typed nodes nested in one node d of type D, then the existence of one query-relevant node of type V' is usually enough to indicate, d is more relevant to the query than another node d' also of type D but with no nested V'-typed nodes containing the keyword(s). In other words, the relevance of a D-typed node which contains a query relevant V'-typed node should not be affected (or normalized) too much by other query-irrelevant V'-typed nodes. □

Example 3.9. *Consider when searching for customers interested in art using the query* "art"*, a customer with "art"-interest along with many other interests (e.g.* $C4$ *in Figure 3.1) should not be regarded as less relevant to the query than another customer who doesn't have "art"-interest but has "art street" in address (e.g.* $C1$ *in Figure 3.1).* □

Compared to the existing works which blindly exploit the compactness of the query results in result ranking [52, 38, 74], a significant difference of the above two principles is: the internal structure of a query result should be exploited as a critical factor to reflect the real relevance of the query results.

Principle 3: The *proximity* of keywords in a query is usually important to indicate the search intention. □

The first two principles look trivial if we know exactly the search via node. However, when the system doesn't have exact information of which node type to search via (as user issues pure keyword query in most cases), they are important in designing the formula of XML TF*IDF similarity; we will utilize them in designing Formula 3.14 for W_a^Q in section 3.4.2.

3.4.2 XML TF*IDF Similarity

$$\rho_s(Q,a) = \begin{cases} & \text{(a) } a \text{ is value node} \\ \frac{\sum_{k \in Q \cap a} W_{Q,k}^{T_a} * W_{a,k}}{W_Q^{T_a} * W_a} & \text{(base case)} \\ & \text{(b) } a \text{ is internalnode} \\ \frac{\sum_{c \in chd(a)} \rho_s(Q,c) * C_{via}(T_c,Q)}{W_a^Q} & \text{(recursive case)} \end{cases} \qquad (3.9)$$

We propose a recursive Formula 3.9, which captures XML's hierarchical structure, to compute XML TF*IDF similarity between an XML node of the desired type to search for and a keyword query. It first (*base case*) computes the similarities between the leaf nodes l of XML data and the query, then (*recursive case*) it recursively computes the similarities between internal nodes n and the query, based on the similarity value of each child c of n and the confidence of c as the node type to search via, until we get the similarities of search for nodes.

In Formula 3.9, Q represents a keyword query; a represents an XML node; $\rho_s(Q,a)$ represents the similarity value between Q and a. We first discuss the intuitions behind Formula 3.9 briefly.

(1) In the base case, we compute the similarity values between XML leaf nodes and a given query in a similar way to the original TF*IDF, since leaf nodes contain only keywords with no further structure.

(2) In the recursive case: on one hand, if an internal node a has more query relevant child nodes while another internal node a' has less, then it is likely that a is more relevant to the query than a'. This intuition is reflected as the numerator in Formula 3.9(b). On the other hand, we should take into account the fan-out (size) of the internal node as normalization factor, since the node with large fan-out has a higher chance to contain

more query relevant children. This is reflected as the denominator of Formula 3.9(b).

Next, we will illustrate how each factor in Formula 3.9 contributes to the XML structural similarity in Section 3.4.2 (for base case) and 3.4.2 (for recursive case).

Base case of XML TF*IDF

Since XML leaf nodes contain keywords with no further structure, we can adopt the intuitions of the original TF*IDF to compute the similarity between a leaf node and a keyword query by using statistics terms f_k^T and $f_{a,k}$ which have been explained in Section 3.2.3.

However, unlike Rule 1 in the original TF*IDF which assigns the same weight to a query keyword w.r.t. all documents (i.e. $W_{Q,k}$ in Formula 3.2), we model and distinguish the weights of a keyword w.r.t. different XML leaf node types (i.e. $W_{Q,k}^{T_a}$ in Formula 3.10), as shown in Example 3.10.

Example 3.10. *Keyword* street *may appear quite frequently in* address *nodes of Figure 3.1 while infrequently in other nodes. Thus it is necessary to distinguish the (low) weight of* street *in* address *from its (high) weight in other nodes. Similarly, we distinguish the weights of a query w.r.t. different XML node types (i.e. $W_Q^{T_a}$), rather than a fixed weight for a given query for all flat documents.* □

Now let us take a detailed look at Formula 3.9. In the base case for XML leaf nodes, each k represents a keyword appearing in both query Q and data node a; T_a is the type of a's parent node; $W_{Q,k}^{T_a}$ represents the weight of keyword k in Q w.r.t. node type T_a. $W_{a,k}$ represents the weight of k in data node a; $W_Q^{T_a}$ represents the weight of Q w.r.t. node type T_a; and W_a represents the weight of a. Following the conventions of the original TF*IDF, we propose the formulas for $W_{Q,k}^{T_a}$, $W_{a,k}$, $W_Q^{T_a}$ and W_a in Formula 3.10, 3.11, 3.12 and 3.13 respectively:

$$W_{Q,k}^{T_a} = C_{via}(Q,a,k) * \log_e(1 + N_{T_a}/(1 + f_k^{T_a})) \tag{3.10}$$

$$W_{a,k} = 1 + \log_e (f_{a,k}) \tag{3.11}$$

$$W_Q^{T_a} = \sqrt{\sum_{k \in Q} (W_{Q,k}^{T_a})^2} \tag{3.12}$$

$$W_a = \sqrt{\sum_{k \in a} W_{a,k}^2} \tag{3.13}$$

In Formula 3.10, N_{T_a} is the total number of nodes of type T_a while $f_k^{T_a}$ is the number of T_a-typed nodes containing keyword k; $C_{via}(Q, a, k)$ is the confidence of node a to be a search via node w.r.t. keyword k (explained in Section 3.3.3).

In Formula 3.11, $f_{a,k}$ is the number of occurrences of k in data node a. Similar to Rule 1 and Rule 2 in original TF*IDF, $W_{Q,k}^{T_a}$ is monotonically decreasing w.r.t. $f_k^{T_a}$, while $W_{a,k}$ is monotonically increasing w.r.t. $f_{a,k}$. W_a is normally increasing w.r.t. the size of a, so put it as part of denominator to play a role of normalization factor to balance between leaf nodes containing many keywords and those with a few keywords.

Recursive case of XML TF*IDF

The recursive case of Formula 3.9 recursively computes the similarity value between an internal node a and a keyword query Q in a bottom-up way based on two intuitions.

Intuition 3.3. *An internal node a is relevant to Q, if a has a child c such that the type of c has a high confidence to be a* search via *node w.r.t. Q (i.e. large $C_{via}(T_c, Q)$), and c is highly relevant to Q (i.e. large $\rho_s(Q, c)$).*

Intuition 3.4. *An internal node a is more relevant to Q if a has more query-relevant children when all others being equal.*

In the recursive case of Formula 3.9, c represents one child node of a; T_c is the node type of c; $C_{via}(T_c, Q)$ is the confidence of T_c to be a search via node type presented in Formula 3.7; $\rho_s(Q, c)$ represents the similarity between node c and query Q which is computed recursively; W_a^Q is the overall weight of a for the given query Q.

Next, we explain the similarity design of an internal node a in Formula 3.9: we first get a weighted sum of the similarity values of all its children, where the weight of each child c is the confidence of c to be a *search via* node w.r.t. query Q. This weighted sum is exactly the numerator of formula 3.9, which also follows *Intuition 3.3* and *3.4* mentioned above. Besides, since *Intuition 3.4* usually favors internal nodes with more children, we need to normalize the relevance of a to Q. That naturally leads to the use of W_a^Q (Formula 3.14) as the denominator.

Normalization factor design

Formula 3.14 presents the design of W_a^Q, which is used as a normalization factor in the recursive case of XML TF*IDF similarity formula. W_a^Q is designed based on *Principle 1* and *Principle 2* pointed out in section 3.4.1.

$$W_a^Q = \begin{cases} \sqrt{\sum\limits_{c \in chd(a)} (C_{via}(T_c, Q) * B + DW(c))^2} & \text{(a) if } a \text{ is} \\ & \quad \text{grouping node} \\ \sqrt{\sum\limits_{T \in chdType(T_a)} C_{via}(T, Q)^2} & \text{(b) otherwise} \end{cases} \tag{3.14}$$

Grouping Node Case

Formula 3.14(a) presents the case that internal node a is a *grouping node*; then for each child c of a (i.e. $c \in chd(a)$), B is considered as a Boolean flag: $B = 1$ if $\rho_s(Q, c) > 0$ and $B = 0$ otherwise; $DW(c)$ is a small value as the default weight of c which we choose $DW(c) = 1/\log_e(e - 1 + |chd(a)|)$ if $B = 0$ and $DW(c) = 0$ if $B = 1$, where $|chd(a)|$ is the number of children of a, so that W_a^Q for grouping node a grows with the number of query-irrelevant child nodes, but grows very slowly to reflect *Principle 2*. Note $DW(c)$ is usually insignificant as compared to $C_{via}(T_c, Q)$.

The intuition for the formula 3.14(a) of *grouping node* a comes from *Principle 2*, so we don't count $C_{via}(T_c, Q)$ in the normalization unless c contains some query-relevant keywords within its subtree. In this way, the similarity of a to Q will not be significantly normalized (or affected) even if a has many query-irrelevant child nodes of the same type. At the same time, with the default weight $DW(c)$, we still provide a way to distinguish and favor a grouping node with small number of children from another grouping node with many children, in case that the two contain the same set of query-relevant child nodes. In other words, the *result specificity* is taken into account in this case.

Non-Grouping Node Case

When internal node a is a *non-grouping node*, we compute W_a^Q based on the type of a rather than each individual node. In Formula 3.14(b), $chdType(T_a)$ represents the node types of the children of a, and it computes the same W_a^Q for all a-typed nodes even if each individual a-typed node may have different sets of child nodes (e.g. some customer nodes have nested address while some do not have).

This design has two advantages. *First*, it models *Principle 1* to achieve a normalization that the size of the subtree of individual node a does not affect the similarity of a to a query.

Example 3.11. *Given a query Q* "customer Art Street", *since* address *has high confidence to be searched via (i.e. $C_{via}(address, Q)$), $C1$ (with address in "Art Street") will be ranked before $C2$ (with interest in "street art") according to the normalization in Formula 3.14(b). However, if we compute the normalization factor based on the size of each individual node, then the high confidence for* address *node doesn't contribute to the normalization factor of $C2$ (who even doesn't have* address *and* street *nodes etc.). As a result, $C2$ has a good chance to be ranked before $C1$ due to its small size which results in small normalization factor.* □

Second, Formula 3.14(b)'s design has advantage in term of computation cost. With W_a^Q for non-grouping node computed based on node types instead of data nodes, we only need to compute W_a^Q for all a-typed nodes once for each query, instead of repeatedly computing W_a^Q for each a-typed node in the data.

Note that, the normalization factor in Formula 3.14(b) potentially favors nodes with more nested node types. However, the existence of one or a few nodes containing query keywords but with low-confidence to be searched via is usually insufficient to outweigh a query-relevant search via node with high confidence. In addition, we do not choose the same normalization factor for all nodes of the same type, because we have to prevent the similarity of internal nodes (up to the search for node) from increasing monotonically from the base case of the recursive XML TF*IDF formula (i.e. Equation 3.9(a)), in order to avoid discriminating against nodes that are nested near the nodes to be searched for.

Note in the base case, a keyword k is less important in T-typed nodes if more T-typed nodes contain k. However, now we consider T-typed nodes are more important for keyword k (i.e. larger $C_{via}(T, k)$). These two, which seem contradictory, are in fact the key to accurate relevance based ranking.

Example 3.12. *Consider when searching for customers with query* "customer art road"*, statistics will normally give more weights to* address *than other node types because of the high frequency of keyword "road" in* address*. But if no customer node has address in "art road" but some have address in "art street", then these customer nodes will be ranked before customers with address containing "road" without "art", because the keyword "road" has a lower weight than "art" in* address *nodes due to its much higher frequency.* □

Advantages of XML TF*IDF

Compatibility - The *XML TF*IDF similarity* can work on both semi-structured and unstructured data, because unstructured data is a simpler kind of semi-structured data with no structure, and XML TF*IDF ranking Formula 3.9(a) for *data node* can be easily simplified to the original TF*IDF Formula 3.1 by ignoring the node type.

Robustness - Unlike existing methods which require a query result to cover all keywords [80, 118, 54, 52], we adopt a heuristic-based approach that does not enforce the occurrence of all keywords in a query result; instead, we rank the results according to their relevance to the query. In this way, more relevant results can be found, as a user query may often be an imperfect description of his real information need [64]. Users never expect an empty result to be returned even though no result can cover all keywords; fortunately, our approach is still able to return the most relevant results to users.

3.5 Algorithms

3.5.1 Data Processing and Index Construction

We parse the input XML document, during which we collect the following information for each node n visited: (1) assign a Dewey label $DeweyID$ [105] to n; (2) store the prefix path $prefixPath$ of n as its node type in a global hash table, so that any two nodes sharing the same $prefixPath$ have the same node type; (3) in case n is a leaf node, we create a data node a (mentioned in section 3.2.2) as its child and summarize two basic statistics data $f_{a,k}$ (in Definition 3.7) and W_a (in Formula 3.13) at the same time. Besides, we also build two indices in order to speedup the keyword query processing.

The first index built is called keyword inverted list, which retrieves a list of data nodes in document order whose values contain the input keyword; moreover, an index (e.g. B+-Tree) is built on top of each inverted list for probing purpose. In particular, we have designed and evaluated three candidates for the inverted list: (1) Dup, the most basic index which stores only the dewey id and XML TF $f_{a,k}$; (2) DupType, which stores an extra node type (i.e. its prefix path) compared to Dup; (3) DupTypeNorm, which stores an extra normalization factor W_a (in Formula 3.13) associated with this data node compared to DupType. DupTypeNorm provides the most efficient computation of XML TF*IDF, as it costs the least index lookup time; in contrast Dup and DupType need extra index lookup to gather the value of $W_{a,k}$ (see formula 3.11) to compute W_a online.

Given a keyword k, the inverted list returns a set of nodes a in document order, each of which contains the input keyword and is in form of a tuple $<DeweyID, prefixPath, f_{a,k}, W_a>$. Each term here has been explained as above. In order to facilitate the explanations of the algorithm, we name such tuple as a "*Node*". It supports the following operations:

- getDeweyID(a,k) returns the Dewey id of data node a.
- getPrefix(a,k) returns the prefix path of a in XML data.
- getFrequency(a,k) returns XML TF $f_{a,k}$ of data node a.
- getWeight(a) returns W_a for data node a.

The second index built is called frequency table, which stores the frequency f_k^T for each combination of keyword k and node type T in XML document. Its worst case space complexity is O(K*N), where K is the number of distinct keywords and N is the number of node types in XML database. Since the number of node types in a well designed XML database is usually small (e.g. 100+ in DBLP 370MB and 500+ in XMark 115MB), the frequency table size is comparable to inverted list. It is indexed by keywords using Berkeley DB B+-Tree [1], so the index lookup cost is O(log(K)). It supports getFrequency(T,k) which returns the value of f_k^T. The values returned by these operations are important to compute the result of formulae in Section 3.4.

3.5.2 Keyword Search & Ranking

Algorithm 3.1 presents a flowchart of keyword search and result ranking. The input parameter $Q[m]$ is a keyword query containing m keywords. Based on the inverted lists built after pre-processing the XML document, we extract the corresponding inverted lists $IL[1]$, ..., $IL[m]$ for each keyword in the query. Each inverted list IL contains a set of tuples in form of $<DeweyID$, $prefixPath$, f_a^k, $W_a>$. F is the frequency table mentioned in section 3.5.1. In particular, Algorithm 3.1 executes in two steps.

Algorithm 3.1: KWSearch($Q[m]$, $IL[m]$, $F[m]$)

```
Let max = 0; $T_{for}$ = null
List L$_{for}$ = getAllNodeTypes()
**foreach** $T_n \in L_{for}$ **do**
    $C_{for}(T_n, Q)$ = getSearchForConfidence($T_n$,Q)
    **if** *($C_{for}(T_n) >$ max)* **then**
        max = $C_{for}(T_n)$; $T_{for}$ = $T_n$
LinkedList rankedList
$N_{for}$ = getNext($T_{for}$)
**while** *(!end($IL[1]$)* || ... || *(!end($IL[m]$)))* **do**
    Node $a$ = getMin(IL[1],IL[2],...,IL[m])
    **if** *(!isAncestor($N_{for}$, a))* **then**
        $\rho_s$(Q,$N_{for}$) = getSimilarity($N_{for}$,Q)
        rankedList.insert($N_{for}$, $\rho_s$(Q,$N_{for}$))
        $N_{for}$ = getNext($T_{for}$)
    **if** *(isAncestor($N_{for}$, a))* **then**
        $\rho_s(Q, a)$ = getSimilarity(a,Q)
    **else**
        $\rho_s(Q, a) = 0$
return rankedList;
```

First, it identifies the search intention of the user, i.e. to identify the most desired search for node type (line 1-6). In particular, it first collects all distinct node types in XML document (line 2). Then for each node type, we compute its confidence to be a search for node through Formula 3.6, and choose the one with the maximum confidence as the desired search for node type T_{for} (line 3-6).

Second, for each search for node candidate N_{for}, it computes the XML TF*IDF similarity between n and the given keyword query (line 7-18). We maintain a $rankedList$ to contain the similarity of each search for node candidate (line 7). N_{for} is initially set to the first node of type T_{for} in document order (line 8). The computation of XML TF*IDF similarity between an XML node and the given query is computed recursively in a bottom-up way (line 9-18): for each N_{for}, we first extract node a which occurs first in document order (line 10), then compute the similarity of all leaf nodes a by calling Function $getSimilarity()$, then go one level up to compute the similarity of the lowest *internal node* (line 15-18), until it reaches up to N_{for}, which is actually the root of all nodes computed before. Then it computes the similarity between current N_{for} and the query (line 12), inserts a pair (N_{for}, ρ) into $rankedList$ (line 13), and moves the cursor to next N_{for} by calling function $getNext()$ and calculates the similarity of next N_{for} in the same way (line 14). Function $isAncestor(N_1, N_2)$ returns true if N_1 is an ancestor of N_2.

Lastly, it returns the ranked list of all search for node candidates by their similarity to the query (line 19).

```
Function getSimilarity (Node a, q[n])
1  if (isLeafNode(a)) then
2      foreach $k \in Q \bigcap a$ do
3          $C_{via}(Q, a, k)$ = getKWCo-occur(Q,a,k);
4          $W^{T_a}_{Q,k}$ = getQueryWeight(Q,k,a);
5          $W^{T_a}_{Q,k}$ = $C_{via}(Q, a, k)$ * $W^{T_a}_{Q,k}$;
6          $W_{a,k}$ = 1+log$_e$(f$_{a,k}$);
7          sum += $W^{T_a}_{Q,k}$ * $W_{a,k}$;
8      $\rho_s(Q, a)$ = sum/($W^{T_a}_q$*getWeight(a));
9  if (isInternalNode(a)) then
10     $W^Q_a$ = getQWeight(a,Q);
11     foreach c∈child(a) do
12         $T_c$ = getNodeType(c);
13         $C_{via}$($T_c$,Q) = getSearchViaConfidence();
14         sum += $getSimilarity(c, Q)$ * $C_{via}$($T_c$,Q);
15     $\rho_s(Q, a)$ = sum/$W^Q_a$;
16 return $\rho_s(Q, a)$;
```

Function $getSimilarity()$ presents the procedure of computing XML TF*IDF similarity between a document node a and a given query Q of size n. There are two cases to consider.

`Case 1`: a is a leaf node (line 1-8). For each keyword k in both a and Q, we first capture whether k co-occurs with keyword k_t matching some node type. Line 3-8 present the calculation details of $\rho_s(Q, a)$ in Formula 3.9(a). The statistics in line 3,5,6 are illustrated in Formula 8, 10 and 11 respectively.

`Case 2`: a is an internal node (line 9-15). We compute a's similarity $\rho_s(Q, a)$ w.r.t. query Q by exactly following Formula 3.9(b). $\rho_s(Q, a)$ is computed by a sum of the product of the similarity of each of its child c and the confidence value of c as a search via node (line 11-14). Finally, $\rho_s(Q, a)$ is normalized by a factor W_a^Q (line 15), which is the weight of internal node a w.r.t. Q. Lastly, we return the similarity value (line 16).

The above search method can be gracefully adapted to handle unstructured data, which provide an easy way to incorporate our ranking techniques in a standard text indexing system to handle both unstructured and semi-structured data.

3.6 Experiments

We have performed comprehensive experiments to compare the effectiveness, efficiency and scalability of XReal with SLCA and XSeek, all implemented in Java and run on a 3.6GHz Pentium 4 machine with 1GB RAM running Windows XP. We tested both synthetic and real datasets. XMark [3] is used as synthetic dataset; WSU, eBay from [2] and DBLP are used as real datasets. The size of the data, the three indices and the the connection table CT (proposed in section 3.5.1), and the total indexing time are reported in Table 3.1. Berkeley DB Java Edition [1] is used to store the keyword inverted lists, frequency table and connection table CT.

The effectiveness test contains two parts: (1) the quality of inferring the desired *search for* node; (2) the quality of our ranking approach.

Table 3.1: Data and Index Sizes

Data	Data Size	Dup	DupType	DupTypeNorm	CT	Index Time
DBLP	370MB	1.96GB	2.05GB	2.23GB	2MB	2.3 hours
XMark	115MB	1.26GB	1.3GB	1.32GB	13MB	58 minutes
WSU	15.6MB	13.1MB	13.4MB	14.1MB	0	91 seconds
eBay	350KB	718KB	732KB	803KB	0	10 seconds

Table 3.2: Test on inferring the *search for* node

	Query	Intention	*XReal*	*SLCA* / XSeek
DBLP (370MB)				
QD_1	Java, book	book	book	book; title / book; article
QD_2	author, Chen, Lei	inproceedings	inproceedings	author
QD_3	Jim, Gray, article	article	article	article
QD_4	xml, twig	inproceedings	inproceedings	title / inproceedings
QD_5	Ling, tok, wang, twig	inproceedings	inproceedings	inproceedings
QD_6	vldb, 2000	inproceedings	inproceedings	inproceedings
WSU (16.5MB)				
QW_1	230	place	course; place	room; crs / course
QW_2	CAC, 101	course	course	course
QW_3	ECON	course	course	prefix / course
QW_4	Biology	course	course	title / course
QW_5	place, TODD	course	course	place / course
QW_6	days, TU, TH	course	course	days / course
eBay (0.36MB)				
QE_1	2, days	auction_info	listing	time_left / listing
QE_2	cpu, 933	listing	listing	cpu / listing
QE_3	Hard, drive, CA	listing	listing	description / listing

3.6.1 Evaluation of Search Effectiveness

Infer the Search For Node

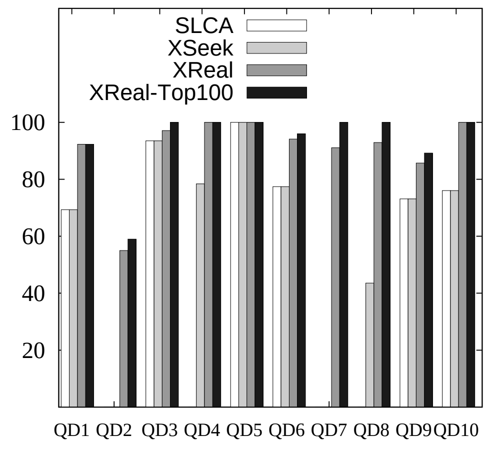
To test XReal's accuracy in inferring the desired search for node, we make a survey of 20 keyword queries, most of which do not contain an explicit search for node. To get a fairly objective view of user search intentions in real world, we post this survey online and ask for 46 people to write down their desired search for and search via nodes. We summarize their answers and choose the queries that more than 80 percentage of users agree on a same search intention. The final queries are shown in Figure 3.2, and some queries contain ambiguities: e.g. QD_1 and QD_3 have both Ambiguity 1 and Ambiguity 2; QD_2, QD_6 and QW_1 have Ambiguity 2. The 4th column contains the search for node inferred by XReal while the 5th column contains the majority node types returned by SLCA and XSeek, as the semantics of SLCA cannot guarantee all results are of the same node type.

We find XReal is able to infer a desired search for node in most queries, especially when the search for node is not given explicitly in the query (e.g. QD_2, QD_4, QW_2, QE_1), or its choice is not unique (e.g. QD_1, QD_3), or both cases such as QW_1. XSeek infers the return nodes of individual keyword matches case by case, rather than addressing the major search intention(s), whereas XReal does so before it goes to find individual matches. In addition, if more than one candidate have comparable confidence to be a search for node, XReal returns all possible candidates (for user to decide), or returns a ranked result list for each such candidate in parallel if user interaction is not preferred. E.g. in QW_1, both place and course can be the return node, as the frequency of “230” in subtrees of course and place are comparable. Note that, the search for node usually models a real world object, so we choose to return sub-trees rooted at the desired search for node, and provide links to the descendants of subtrees for user interested in particular parts of the subtree to explore.

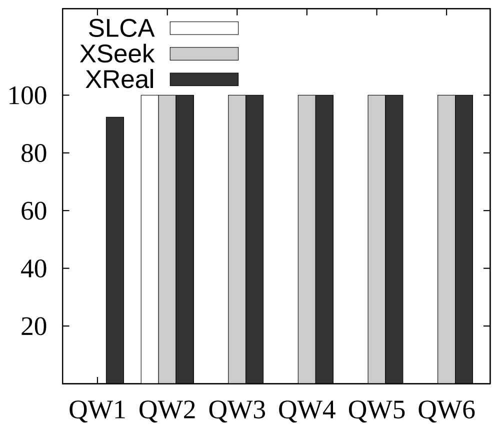
Precision, Recall & F-measure

To measure the search quality, we evaluate all queries in Figure 3.2, and summarize two metrics borrowed from IR field: precision and recall. Precision measures the percentage of the output subtrees that are desired; recall measures the percentage of the desired subtrees that are output. We obtain the correct answers by running the schema-aware XQuery with an additional manual verification. As most queries on DBLP have more than 100 results, we compute XReal’s *top-100* precision and top-100 recall besides the overall ones; since SLCA and XSeek do not provide any ranking function, we only compute their overall precision and recall. Besides, as there are less than 100 results for each query issued on WSU and eBay, we do not show the top-100 precision and recall in Figure 3.2(b)-3.2(c) and Figure 3.3(b)-3.3(c).

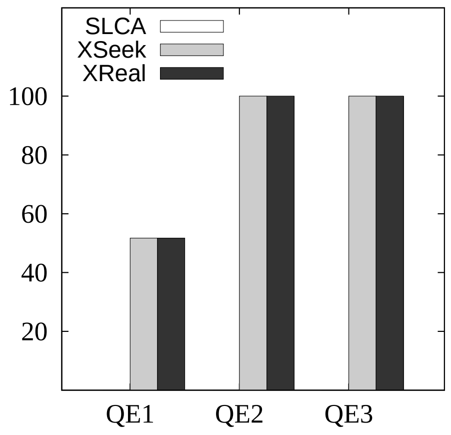
To evaluate XReal’s performance on large real datasets, we include four more queries for DBLP: QD_7 “Philip Bernstein”; QD_8 “WISE”; QD_9 “ER 2005”; QD_{10} “LATIN



(a) DBLP



(b) WSU



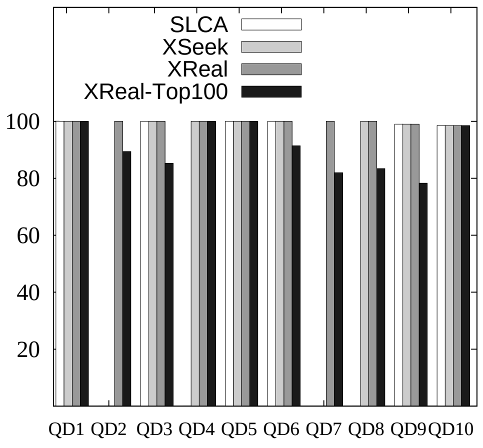
(c) EBAY

Figure 3.2: Precision Comparison(%)

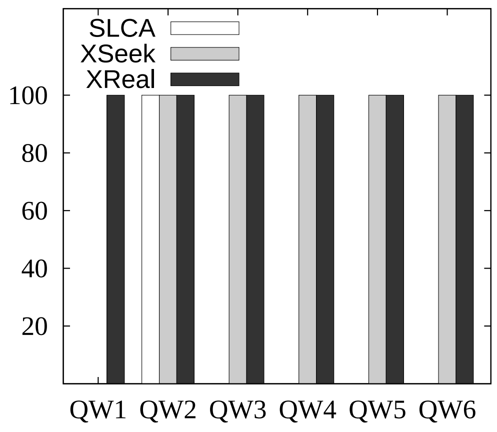
2006". Each of these queries has Ambiguity 2 problem, e.g. "WISE" can be the *booktitle*, *title* of inproceedings, or a value of *author*.

From Figure 3.2 and 3.3, we have four main observations.

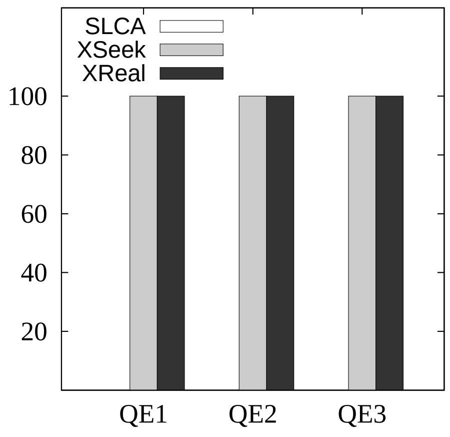
(1) XReal achieves higher precision than SLCA and XSeek for the queries that contain ambiguities (e.g. QD_1-QD_4, QD_6-QD_{10}, QW_1). E.g. in QD_3 which intends to find the articles written by author "Jim Gray", since "article" can be either a tag name or a value of title node, and "Jim" and "Gray" can appear in one author or two different authors, SLCA and XSeek generate some false positive results and lead to low accu-



(a) DBLP



(b) WSU



(c) EBAY

Figure 3.3: Recall Comparison(%)

racy, while XReal addresses these ambiguities well. As another example in QD_9 which intends to find the inproceedings of *ER* conference in year 2005, since "ER" appears in both *booktitle* and *title*, and "2005" appears in both *title* and *year*, XSeek returns not only the intended results, but also other inproceedings whose titles contain both keywords; but XReal correctly interprets the search intention.

(2) SLCA suffers a zero precision and recall from the pure keyword value query, e.g. QD_4, QD_7, QD_8, QW_1, QE_1-QE_3, as the SLCA results contain nothing relevant except the SLCA node. E.g. for QD_8 SLCA returns the *booktitle* or *title* nodes containing

"WISE", while user wants the inproceedings of "WISE" conference. In contrast, XReal correctly captures the search intention. XSeek suffers a zero precision in QD_2 and QD_7, mainly because it mistakenly decides "*author*" as an entity, while the query intends to find the publications.

(3) XReal Performs as well as XSeek (in both recall and precision) when queries have no ambiguity in XML data (e.g. QD_5, QW_4-QW_6, QE_1-QE_3). XReal has a low precision on QD_2, as there are more than one person called Lei Chen in DBLP, while the users are only interested in one of them.

(4) For queries that have more than 100 results on DBLP such as QD_3, QD_6-QD_9, *XReal Top-100* has a higher precision (and lower recall) than overall XReal, which indirectly proves our ranking strategy works well on large datasets.

Table 3.3: F-Measure Comparison

F-measure	*SLCA*	*XSeek*	*XReal*	*XReal top-100*
DBLP	0.272	0.3461	0.4748	0.4799
WSU	0.0083	0.4162	0.4967	0.4967
EBAY	0	0.4002	0.4002	0.4002

Furthermore, we adopt *F-measure* used in IR as the weighted harmonic mean of precision and recall. We compute the average precision and recall of all queries in Figure 3.2 for each dataset (plus QD_7-QD_{10}), adopting formula $F = precision * recall/(precision + recall)$ to get F-measure in Table 3.3. We find XReal beats SLCA and XSeek on all datasets, and achieves almost a perfect value of F which is 0.5 on WSU.

3.6.2 Evaluation of Ranking Effectiveness

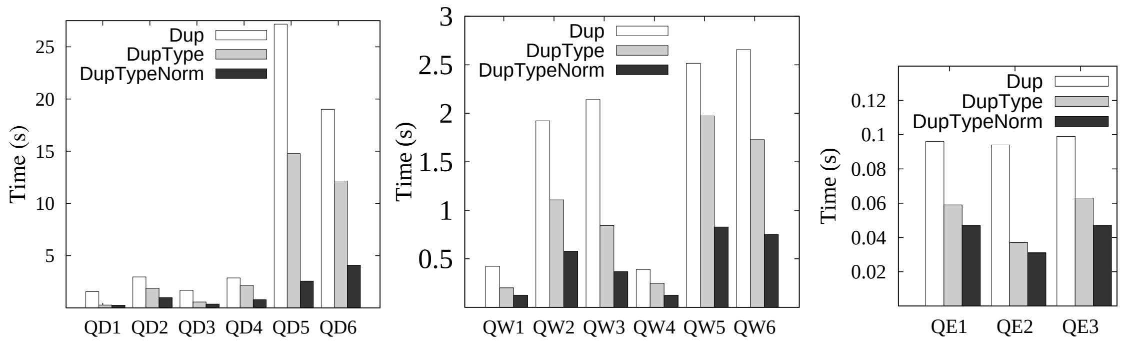
To evaluate the effectiveness of XML TF*IDF alone, we use three measures widely adopted in IR field. (1) *Number of top-1 answers that are relevant.* (2) *Reciprocal rank* (R-rank). For a given query, the reciprocal rank is 1 divided by the rank at which the first correct answer is returned, or 0 if no correct answer is returned. (3) *Mean Average*

Precision (MAP). A precision is computed after each relevant answer is retrieved, and MAP is the average value of such precisions. The first two measure how good the system returns one relevant answer, while the third one measures the overall effectiveness for top-k answers returned, k=40 for DBLP (as DBLP data has very large size) and k=20 for others (if they do exist).

We evaluate a set of 30 randomly generated queries on DBLP, and 10 queries on WSU, eBay and XMark, with an average of 3 keywords. The average values of these metrics are recorded in Table 3.4. We find XReal has an average R-rank greater than 0.8 and even over 0.9 on DBLP. Besides, XReal returns the relevant result in its top-1 answer in most queries, which shows high effectiveness of our ranking strategy.

Table 3.4: Ranking Performance of XReal

Dataset	*Top-1 Number/Total Number*	*R-Rank*	*MAP*
DBLP	27/30	0.946	0.925
WSU	8/10	0.85	0.803
eBay	9/10	0.9	0.867
XMark	7/10	0.791	0.713

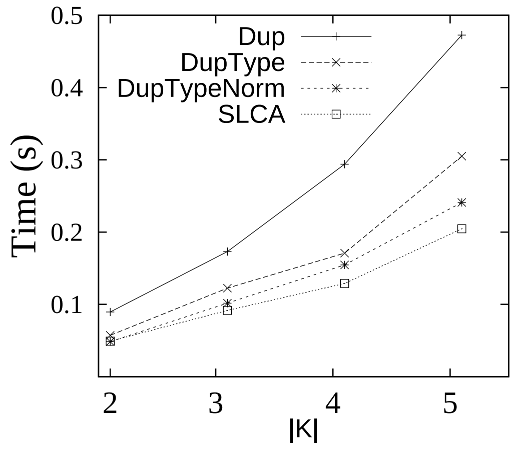


(a) DBLP (b) WSU (c) eBay

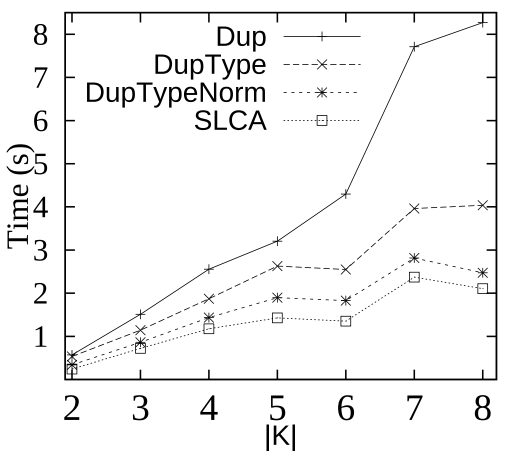
Figure 3.4: Response time on individual queries

3.6.3 Evaluation of Efficiency

We compare the query response time of XReal adopting three indices for keyword inverted list mentioned in section 3.5.1, i.e. *Dup*, *DupType* and *DupTypeNorm*, measured by the timestamp difference between a query is issued and result is returned. Throughout section 3.6, XReal refers to the one adopting *DupTypeNorm*. Figure 3.4 shows the time on hot cache for queries listed in Figure 3.2. *DupTypeNorm* outperforms the other two on all three real datasets, about 2 and 4 times faster than *DupType* and *Dup* respectively. Because *DupTypeNorm* stores the dewey id, node type and normalization factor (for data nodes) together, thus it needs less number of index lookups to fulfill the similarity computation in Formula 3.9. Such advantage is significant when the number of keywords is large or query result size is large, e.g. QD_5 and QD_6 in Figure 3.4(a).



(a) XMark



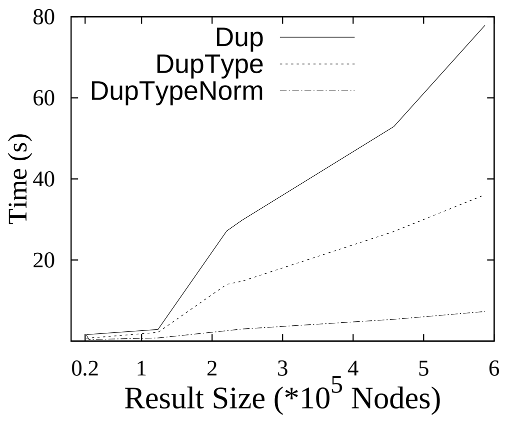
(b) DBLP

Figure 3.5: Response time on different number of keywords |K|

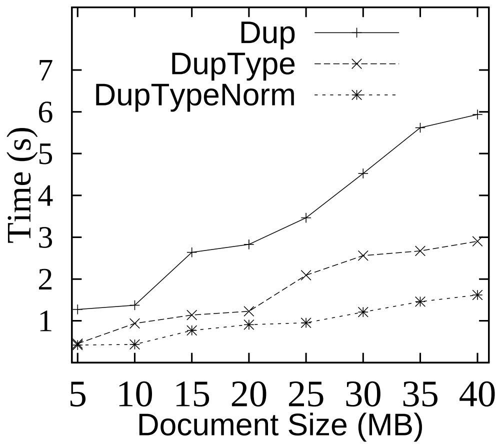
3.6.4 Evaluation of Scalability

Among the existing keyword search methods [118, 54, 38], SLCA is recognized as the most efficient one so far, so we compare XReal with SLCA on DBLP and XMark. For each dataset, we test a set of 50 randomly generated queries, each guarantees to have at least one SLCA result and contains $|K|$ number of keywords, where $|K|$ = 2 to 8 for DBLP and $|K|$ = 2 to 5 for XMark. The response time is the average time of the corresponding 50 queries in four executions on hot cache, as shown in Figure 3.5.

From Figure 3.5(a) and 3.5(b), we find XReal is nearly 20% slower than SLCA on both datasets which is acceptable, because XReal does extra search intention identification, precise result retrieval and ranking; and XReal finds extra results; so this overhead is worthwhile. We also find, the response time of each proposed index increases as $|K|$ increases. In particular, the one with *DupTypeNorm* index costs less time than *DupType*, in turn less than *Dup*. XReal adopting *DupTypeNorm* index scales as well as SLCA, especially when $|K|$ varies from 5 to 8 for DBLP (Figure 3.5(b)).



(a) DBLP



(b) XMark

Figure 3.6: Response time w.r.t. result/document size

Besides, we evaluate the scalability of those indices by drawing the relationship between the response time and query result size (in term of number of nodes returned). A range of 15 queries with various result sizes run over DBLP, and the result is shown in Figure 3.6(a). We can see *DupTypeNorm* again outperforms the other two, and scales linearly w.r.t. the query result size. Similarly, we test the response time of a query "*location united states item*" on XMark data of size 5MB up to 40MB. As shown in Figure 3.6(b), both *DupTypeNorm* and *DupType*'s response time increase linearly w.r.t. the data size.

3.7 Summary

In this chapter, we discovered three critical problems specific to keyword search over XML data tree, which have not been solved by any existing work yet. The first problem, also the most important one, is the keyword ambiguity problems: a keyword can be a tag name or a value of some node, or value of different nodes, or tag name of different nodes, carrying different meanings as they appear in different contexts. Thus, the search intentions (i.e. search targets and search constraints) of a user query is often various. The second problem is how to identify the search target for a keyword query. The third problem is how to rank the results in term of its relevance to user's search intentions in presence of the keyword ambiguity problems.

We find that, as a convinced search target (i.e. a node type to be searched for), its subtree in XML data should cover all keywords, contain as much relevant information while exclude as much irrelevant information as possible. Based on this intuition, we designed a scoring model to quantify the confidence of the search target candidates as the desired search target that a user query intends to *search for*. Then, motivated by the success of TF*IDF ranking model in information retrieval field, we defined XML TF and XML DF, based on which we designed formulae to compute the confidence level of each candidate node type to be a *search via node*, and further proposed a novel *XML TF*IDF similarity* ranking scheme to capture the hierarchical structure of XML data. Lastly, extensive experiments on both real and synthetic data set have been conducted to verify the effectiveness of our search target identification and result ranking methods.

CHAPTER 4

EFFECTIVE KEYWORD SEARCH OVER XML DIGRAPH MODEL

4.1 Introduction

In chapter 3, we mainly talk about how to answer a keyword query over XML data tree. However, the tree model ignores the ID references (denoted by "IDRef") between the elements in XML data. Without considering IDRefs, some relevant results may be missed. Therefore, in this chapter, we would like to investigate how to find meaningful and relevant results of a keyword query over the XML data where IDRefs are taken into consideration.

Motivation

One solution to this problem is: we treat XML data as a general directed graph (i.e. the direction of the edge is considered) such as the one in Figure 4.1, and find all the steiner trees [24], each of which is a directed rooted tree that contains all query keywords. Usually we start from finding the steiner tree with minimal size, then incrementally enlarge the size of the steiner tree found. However, this solution suffers three problems. *First*, similar to LCA semantics for tree data model, the concept of steiner tree itself exploits only the structure of the data while ignoring how to capture user's real search need. *Second*, the problem of finding the results by increasing the sizes of such steiner trees for keyword proximity has been proven to be NP-hard [77], thus such solution is heuristics-based and intrinsically expensive. *Third*, without distinguishing the containment edge (i.e. parent-child edge) and the reference edge (i.e. IDRef edge) in XML data, it may incur a lot overhead in generating the steiner trees that do not meet user's search concern though they contain all query keywords.

Despite the inherent problem of treating XML data with IDRef edges as a general digraph to do keyword query processing, we have another important observation. With the presence of clean and well organized knowledge domains such as Wikipedia, World Factbook, IMDB [5] etc, the future search technology should appropriately help users precisely finding explicit objects of interest. As XML is becoming a standard in data exchange and representation in the internet, in order to achieve the goal of "finding only the *meaningful* and *relevant* data fragments corresponding to the objects (that users really are interested in)", search techniques over XML document need to exploit the matching semantics at *object-level* due to the following two reasons.

First, the information in XML document can usually be recognized as a set of real world objects [80], each of which has attributes and interacts with other objects through relationships. E.g. Course and Lecturer can be recognized as objects in the XML data of Figure 4.1. *Second*, whenever people issue a keyword query, they would like to find information about specific objects of interest, along with their relationships. E.g. when people search Figure 4.1's data by a query "smith", they most likely intend to find the *Lecturer* object about "smith". Therefore, it is desired that the search engine is able to find and extract the data fragments corresponding to the real world objects.

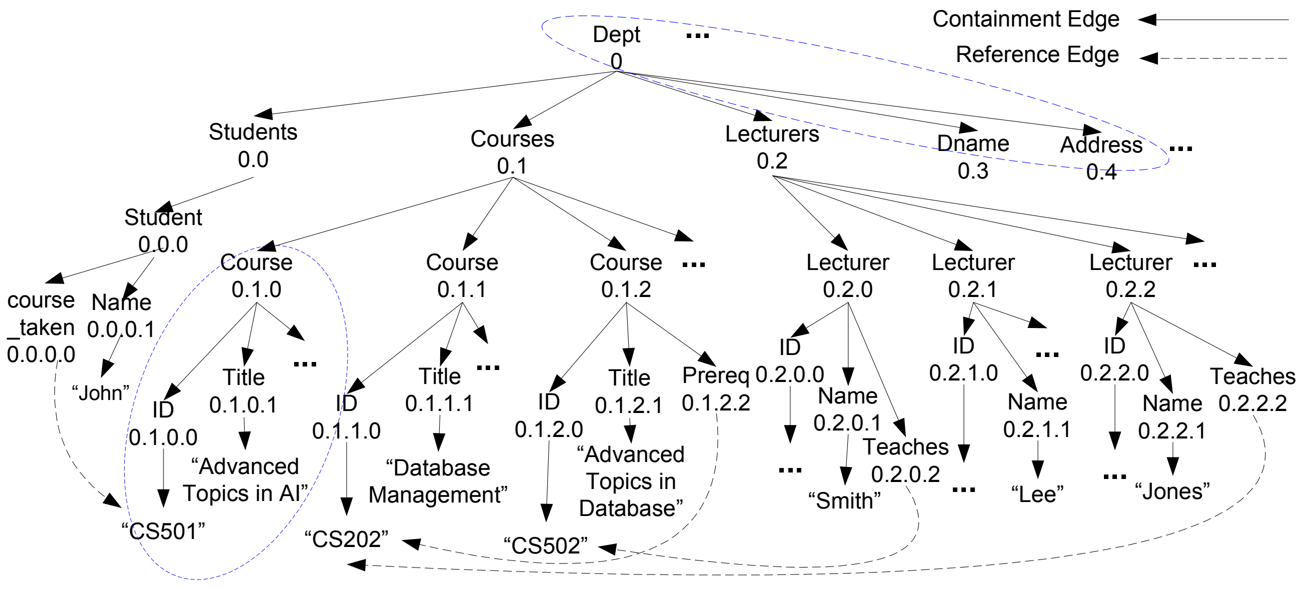


Figure 4.1: Example XML data (with Dewey IDs)

Our Approach

In fact, in most real-life datasets, semantic information about real objects is either explicitly presented or can be inferred without much effort. Therefore, we aim to utilize the semantic information associated with XML data to build an object-level XML keyword search framework, which manifests better result quality in term of result relevance and query answering efficiency.

In particular, we first propose to model XML document as a set of object trees, where each real world object o (with its associated attributes) is encapsulated in an *object tree* whose root node is a representative node of o; two object trees are interconnected via a containment or reference edge in XML data. E.g. The part enclosed by a dotted circle in Figure 4.1 shows an object tree for Dept and Course.

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

Then we design a customized ranking scheme for ISO and IRO results. The ranking function $ISORank$ designed for ISO result not only considers the *content* of result by extending the original TF*IDF [101] to object tree level, but also captures the keyword co-occurrence and specificity of the matching elements. The $IRORank$ designed for an IRO result considers both its self similarity score and the "bonus" score contributed from its interconnected objects. We design efficient algorithms and indices to dynamically compute and rank the matched ISO results and IRO results in one phase. Finally, we experimentally compare ISO and IRO algorithms to the best existing methods XSeek [80] and XReal [16] with real and synthetic data sets. The results reveal that our approach outperforms XReal by an order of magnitude in term of response time and is superior to XSeek in term of recall ratio, well confirming the advantage of our novel semantics and ranking strategies. A search engine prototype incorporating the above proposed techniques is implemented, and a demo of the system on DBLP data is available at http://xmldb.ddns.comp.nus.edu.sg [15].

Our contributions in this chapter are summarized as follows:

- We propose a novel data model for XML data with ID references considered, namely interconnected object-tree model, to well capture user's search concern on real-world objects.

- We propose two major matching semantics based on this data model, to capture user's search concern on a single object of interest or multiple objects of interest that are connected in a meaningful way and help find more relevant results.

- Efficient algorithms are proposed to find the matching results and an elaborate object-level result ranking model is designed.
- Extensive experiments have been conducted to evaluate the efficiency and effectiveness of our approach.

The rest of the chapter proceeds as follows. Section 4.2 presents our interconnected object-tree data model. Section 4.3 presents two major matching semantics, Interested Single Object tree (ISO) and Interested Related Object trees (IRO). Section 4.5 describes the index construction. Section 4.4 presents our ranking schemes. Section 4.6 discusses the search and ranking algorithms. Section 4.7 reports experiment results, and section 4.8 summarizes this chapter.

4.2 Data Model

Definition 4.1. *(**Object Tree**) An object tree* [1] *t in D is a subtree of the XML document, where its root node r is a representative node*[2] *to denote a real world object o, and each attribute of o is represented as a child node of r.*

In an XML document D, a real-world object o is stored in form of a subtree due to its hierarchical inherency. How to identify the object trees is orthogonal to this work; here, we adopt the inference rules in XSeek [80] to help identify the object trees, as clarified in Definition 4.1. As we can see from Figure 4.1, there are 7 object trees (3 Course, 3 Lecturer and 1 Dept), and the part enclosed by a dotted circle is an object tree for Course:0.1.0 and Dept:0 respectively. Note that nodes Students, Courses

[1] Terms *object tree* and *object* are used interchangeably in the rest of this chapter.

[2] To facilitate our discussion in the rest of this chapter, we use the representative node to denote the corresponding object tree.

and Lecturers of Dept:0 are *connection* nodes, which connect the object "Dept" and multiple objects "Student" ("Course" and "Lecturer").

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

Although it is desirable to materialize all the precise relationships from XML data, existing techniques are not able to do so. Fortunately, such relationships can be generalized into two meaningful structural relationships in XML data as below.

Definition 4.2. ***(Reference-connection)*** *Two object trees u and v in an XML document D have a reference-connection (or are reference-connected) if there is an ID reference relationship between u and v in D.*

Definition 4.3. ***(Containment-connection)*** *Two object trees u and v in an XML document D have a containment-connection if there is a P-C relationship between the root node of u and v in D, regardless of the connection node.*

Definition 4.4. ***(Interconnected object-trees model)*** *models an XML document D as a set of object trees, D=(T,C), where T is a set of object trees in D, and C is a set of conceptual connections between the object trees.*

In contrast to the model in XSeek [80], ID references in XML data is considered in our model to find more meaningful results. From Figure 4.1, we can find Dept:0 and Course:0.1.0 are interconnected via a containment connection, and Lecturer:0.2.0 and Course:0.1.2 are reference-connected.

4.3 Object-Level Matching Semantics

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

4.3.1 ISO Matching Semantics

Definition 4.5. *(**ISO**) Given a keyword query Q, an object tree o is the Interested Single Object (ISO) of Q, if o covers all keywords in Q.*

ISO can be viewed as an extension of LCA, which is designed to capture user's interest on a single object. E.g. for a query "database, management" issued on Figure 4.1, LCA returns two subtrees rooted at Title:0.1.1.1 and Courses:0.1, neither of which is an object tree; while ISO returns an object tree rooted at Course:0.1.1.

4.3.2 IRO Matching Semantics

Consider a query "CS502, lecturer" issued on Figure 4.1. ISO cannot find any qualified answer as there is no single object qualified while user's search concern is on multiple objects. However, there is a Lecturer:0.2.0 called "Smith" who teaches Course "CS502" (via a reference connection), which should be a relevant result. This motivates us to design IRO (*Interested Related Object*).

Analysis of User's Search Concern

The difficulty of keyword search is to capture the search intention, as two users may issue the same query for different search intentions, or different queries for the same intention. However, for a given query, the target that users search for (i.e. user's search concern) is usually fixed, which is either objects or relationships. Here, we use a pair of objects (t_1,t_2) to analyze user's search concerns, and such analysis can be easily extended to a group of objects.

Case 1: User knows the relationship R between t_1 and t_2, and information of a particular object (e.g. t_1) ahead before they issue a query. What they intend to find is the detailed information of t_2 which relates to t_1 via relationship R.

Case 1.1: There is a *direct* relationship R between t_1 and t_2.

Example 4.1. *Query "CS202, Lecturer" is issued on Figure 4.1. As user knows "CS202" is the course and the precise relationship* Teach *(captured by reference edge between Lecturer and Course), he intends to find the lecturer teaching "CS202".* □

Case 1.2: A series of relationships R_1,...,R_n connect t_1 to t_2 *indirectly.*

Example 4.2. *Query "Smith, Prereq" is issued on Figure 4.1, intending to find the pre-requisite of the course taught by* Smith. *The relationships between Lecturer* "Smith" *and the result Course* "CS202" *are shown as below:*
Lecturer "Smith"→*(*teach*) Course* "CS502" →*(*pre-requisite*) Course "CS202"* □

Case 2: User knows the information of objects t_1 and t_2 beforehand, and wants to explore whether any relationship R exists between t_1 and t_2.

Example 4.3. *Query "Jones, CS202" is issued on Figure 4.1, which intends to find how Lecturer "Jones" and Course "CS202" are connected.* □

IRO pair & IRO group

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

Definition 4.6. *(**n-hop-meaningful-connection**) Two object trees u and v in an XML document have a n-hop-meaningful-connection (or are n-hop-meaningfully-connected) if there are $n-1$ distinct intermediate object trees $t_1, ...t_{n-1}$, s.t.*

1. *there is either a reference connection or a containment connection between each pair of adjacent objects;*

2. *no two objects are connected via a common-ancestor relationship.*

Definition 4.7. *(**IRO pair**) For a given keyword query Q, two object trees u and v form an IRO pair w.r.t. Q if the following two properties hold:*

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

IRO pair is designed to capture user's search concern on two objects that have a direct or indirect conceptual connection. Intuitively, the larger the upper limit L is, more results can be found, but the relevance of those results decay accordingly. Let us take an example to understand the rational behind Definition 4.7.

Example 4.4. *Consider a query Q "Smith, Database, Management" issued on Figure 4.1's XML data. Object tree Lecturer:0.2.0 and Course:0.1.1 form an IRO pair, because they are connected by a 1-hop-meaningful-connection: Course:0.2.0 "Database Management" is a prerequisite (i.e. label* Prereq *in Figure 4.1) of the Course:0.1.2 that Lecturer:0.2.0 "Smith" teaches.* □

Besides a pair of object trees, a more general case is, a group of interconnected object trees are involved to capture user's search concern. Therefore, *IRO group* is introduced to capture the relationships among three or more connected objects.

Definition 4.8. *(**IRO group**) For a given keyword query Q, a group G of object trees forms an IRO group if:*

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

As an example, for query "Jones, Smith, Database" issued on Figure 4.1, four objects Course:0.1.1, Course:0.1.2, Lecturer:0.2.0 and Lecturer:0.2.2 form an IRO group (with $L' = 2$), where both Course:0.1.1 and Course:0.1.2 can be the hub. The connection is: Lecturer:0.2.2 "*Jones*" teaches Course:0.1.1, which is a pre-requisite of a "*Database*" Course:0.1.2 taught by Lecturer:0.2.0 "*Smith*".

In the rest of the chapter, we call an object involved in IRO semantics as the *IRO object*. Note that, an ISO object o can form an IRO pair (or IRO group) with an IRO object o', but o is not double counted as an IRO object.

4.3.3 Separation of ISO & IRO Results Display

As ISO and IRO correspond to different user search concerns, we separate the results of ISO and IRO in our online demo[1] [15], which is convenient for user to quickly recognize which category of results meet their search concern, thus a lot of user efforts are saved in result consumption.

4.4 Relevance Oriented Result Ranking

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

4.4.1 Ranking for ISO

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

Object-level TF*IOF similarity ($\rho(o, Q)$) Inspired by the extreme success of IR style keyword search over flat documents, we extend the traditional TF*IDF (Term

[1]Note: in our previous demo, ISO was named as ICA, while IRO was named as IRA.

frequency*Inverse document frequency) similarity [101] to our object-level XML data model, where flat document becomes the object tree. We call it as *TF*IOF* (Term frequency*Inverse object frequency) similarity. Such extension is adoptable since the object tree is an appropriate granularity for both query processing and result display in XML. Since TF*IDF only takes the *content* of results into account, but cannot capture XML's *hierarchical structure* we enforce the *structure* information for ranking in the following three factors.

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

$$\rho(o, Q) = \frac{\sum_{\forall k \in o \cap Q} W_{Q,k} * W_{o,k}}{W_Q * W_o} \tag{4.1}$$

The detailed expansion of $W_{Q,k}$ and $W_{o,k}$ is as below.

$$W_{Q,k} = \frac{N}{1+f_k}$$

$$W_{o,k} = \sum_{\forall e \in attr(o,k)} tf_{e,k} * W_e$$

In Equation 4.1, $k \in o \cap Q$ means keyword k appears in both o and Q. $W_{Q,k}$ represents the weight of keyword k in query Q, playing a role of inverse object frequency (*IOF*); N is the total number of objects in xml document, and f_k is the number of objects containing k. $W_{o,k}$ represents the weight of k in object o, counting the term frequency (*TF*) of k in o. $attr(o, k)$ denotes a set of attributes of o that directly contain k; $tf_{e,k}$ represents the frequency of k in attribute e, and W_e is the adjustable weight of matching element e in o, whose value is no less than 1, and W_e is set to 1 for all the experiments conducted in section 4.7.

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

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

Based on the above intuition, we present $c(o, Q)$ in Equation 4.2 (denominator part), which is modeled as *inversely proportional to the minimal number of attributes that are nested in o and together contain all keywords in Q.* Since this metric favors the single-keyword query, we put the number of query keywords (i.e. $|Q|$ in nominator part) as a normalization factor.

$$c(o, Q) = \frac{|Q|}{min(|\{E|E = attrSet(o) \ and \ (\forall k \in Q, \exists e \in E \ s.t. \ e.contain(k))\}|)} \quad (4.2)$$

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

Example 4.5. *When searching for a person by a query "David, Lee", a person p_1 with the exact name should be ranked higher than a person p_2 named* "David Lee Ming*", as*

p_1's name fully specifies the keywords in query, while p_2 does not. □

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

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

So far, we have exploited both the *structure* (i.e. factors F1,F2,F3) and *content* (*TF*IOF* similarity) of an object tree o for our ranking design. Since there is no obvious comparability between structure score and content score, we use product instead of summation to combine them. Finally, the $ISORank(o, Q)$ is:

$$ISORank(o, Q) = \rho(o, Q) * (c(o, Q) + s(o, Q)) \tag{4.3}$$

4.4.2 Ranking for IRO

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

$$IRORank(o, Q) = \rho(o, Q) + Bonus(o, Q) \tag{4.4}$$

where $\rho(o, Q)$ is the *TF*IOF similarity* of object o to Q (Equation 4.1). The reason we do not use $ISORank$ as the choice of the similarity of o is, both $c(o, Q)$ and $s(o, Q)$ in Formula 4.3 is 0, because o does not contain all keywords. $Bonus(o, Q)$ is the extra contribution to o from all its IRO pair/group's counterparts for Q, which can be used as a *relative* relevance metric for IRO objects to Q, especially when they have a comparable *TF*IOF similarity* value. Regarding the design of $Bonus$ score to an IRO object o for

Q, we present three guidelines first.

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

For example, consider a query "interest, painting, sculpture" issued on XMark [3]. Suppose two persons Alice and Bob have interest in "*painting*"; Alice has conceptual connections to many persons about "*sculpture*" (indicated by attending the same auction), while Bob has connections to only a few of such auctions. Thus, Alice is most likely to be more relevant to the query than Bob.

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

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

Example 4.6. *Consider a query "XML, twig, query, processing" issued on DBLP. Suppose a paper p_0 contains "XML" and "twig"; p_1 contains "query" and "processing" and is cited by p_0; p_2 contains the same keywords as p_1; p_3 contains no keyword, but cites p_0 and p_2; p_4 contains "query" and p_5 contains "processing", and both cite p_0. By Definition 4.7-4.8, p_1 forms an IRO pair with p_0; p_2, p_3 and p_0 form an IRO group; p_0, p_4 and p_5 form an IRO group. Therefore, in computing the rank of p_0, the influence from p_1 should be greater than that of p_2 and p_3, and further greater than p_4 and p_5.* $\square$

According to the above two guidelines, the $Bonus$ score to an IRO object o is presented in Equation 4.5. $Bonus(o, Q)$ consists of the weighted connection counts from its IRO pair and group respectively, which manifests Guideline 1. w_1 and w_2 are designed to reflect the weights of the counterparts of o's IRO pair and group respectively, where

$w_1 > w_2$, which manifests Guideline 2.

$$Bonus(o,Q) = w_1 * BS_{IRO\_P}(o,Q) + w_2 * BS_{IRO\_G}(o,Q) \tag{4.5}$$

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

Example 4.7. *Consider a query Q "XML, query, processing" issued on DBLP. The bonus score to a "query processing" paper from a related "XML" conference* inproceedings *should be distinguished from the bonus score coming from a related* book *whose title contains "XML", regardless of the self-similarity difference of this inproceedings and book.* □

Although the distinction of contributions from different object types under a certain matching semantics helps distinguish the $IRORank$ of an IRO object, it is preferable that we can distinguish the precise connection types to o to achieve a more exact $Bonus$ score. However, it depends on a deeper analysis of the relationships among objects and more manual efforts. Therefore, in this work we only enforce Guideline 1 and Guideline 2. As a result, the IRO bonus from the counterparts of o's IRO pair and IRO group is presented in Equation 4.6-4.7:

$$BS_{IRO\_P}(o,Q) = \sum_{\forall o' | (o,o') \in IROPair(Q,L)} \rho(o',Q) \tag{4.6}$$

$$BS_{IRO\_G}(o,Q) = \frac{\sum_{\forall g \in IROGroup(Q,L') | o \in g} BF(o,Q,g)}{|IRO\_Group(o,Q)|} \tag{4.7}$$

In Equation 4.6, $\rho(o',Q)$ is the *TF*IOF similarity* of o' w.r.t. Q, which is adopted as the contribution from o' to o. Such adoption is based on the intuition that, if an object tree o_1 connects to o'_1 s.t. o'_1 is closely relevant to Q, whereas object tree o_2 connects to o'_2 which is not as closely relevant to Q as o'_1, then it is likely that o_1 is more relevant to Q than o_2. In Equation 4.7, $BF(o,Q,g)$ can be set as the self similarity of the object in g

containing the most number of keywords. As it is infeasible to design a one-fit-all bonus function, other alternatives may be adopted according to different application needs. L (in Equation 4.6) and L' (in Equation 4.7) is the upper limit of n in definition of *IRO pair* and *IRO group*.

4.5 Index Construction

As we model the XML document as the interconnected object-trees, the *first* index built is the *keyword inverted list*. An object tree o is in the corresponding list of a keyword k if o contains K. Each element in the list is in form of a tuple (Oid, DL, $w_{o,k}$), where Oid is the id of the object tree containing k (here we use the dewey label of the root node of object tree o as its oid, as it serves the purpose of unique identification); DL is a list of pairs containing the dewey labels of the exact locations of k and the associated attribute name; $w_{o,k}$ is the term frequency in o (see Equation 4.1). $c(o, Q)$ (in Equation 4.2) can be computed by investigating the list DL; $s(o, Q)$ is omitted in index building, algorithm design and experimental study later due to the high complexity to collect. Therefore, the $ISORank$ of an object tree can be efficiently computed. A B+ tree is built on top of each inverted list to facilitate fast probing of an object in the list.

The $ISORank$ of an object tree can be efficiently computed, as *TF* $W_{o,k}$ and *IOF* $W_{Q,k}$ are pre-computed in index building, and the keyword co-occurrence $c(o, Q)$ (in Equation 4.2) of the matching elements can be easily derived from the keyword inverted lists.

Without loss of generality, we assume there is only one object class and one inverted list for each keyword in the following since the keyword query processing of multiple interested object class can be performed independently.

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

4.6 Algorithms

The backbone workflow compute and rank the ISO and IRO results is in Algorithm 4.1. Its main idea is to scan the shortest keyword inverted list IL_s, check the objects in the list and their connected objects, then compute and rank the ISO and IRO results. The details are: for each object tree o in IL_s, we find the keywords contained in o by calling function $getKeywords()$(line 5). If o contains all query keywords, then o is an ISO object, and we compute the $ISORank$ for o by calling $initRank()$, then store o together with its rank into hash table HT(line 6-7). If o contains some keywords, then o is an IRO object, and all its IRO pairs and groups are found by calling functions $getIROPairs()$ (Algorithm 4.2) and $getIROGroups$ (Algorithm 4.3) (line 8-10). The results for ISO and IRO are inserted into RL for output(line 11).

Function $computeRank()$ computes/updates the ranks of objects o' in $oList$, each forming an IRO pair with o. For each such o', it probes all inverted lists with o' to check two cases: (1)If o' is an ISO object containing all query keywords, then its $ISORank$ is computed and it is added into ISO_Result (line 3-4). (2)If both o and o' are IRO objects, their *TF*IOF* similarity are initialized (if not yet), and their $IRORank$s are updated accordingly (line 5-9). Function $initRank()$ computes its $ISORank$ by Equation 4.3 if o is an ISO object; otherwise computes its TF*IOF similarity by Equation 4.1.

Algorithm 4.1: KWSearch

```
   Input: Keywords: $KW[m]$; Keyword Inverted List: $IL[m]$; Connection Table: $CT$;
          upper limit: $L$, $L'$ for IRO pair and group
   Output: Ranked object list: $RL$
1  let $RL$ = $ISO\_Result$ = $IRO\_Result$ = {};
2  let $HT$ be a hash table from object to its rank;
3  let $IL_s$ be the shortest inverted list in $IL[m]$;
4  for each object $o \in IL_s$ do
5      let $K_o$ = getKeywords($IL, o$);
6      if ($K_o == KW$)                /* $o$ is an ISO object */
7          initRank($o$,$K_o$,$KW$,$HT$); $ISO\_Result$.add($o$);
8      else if ($K_o \neq \emptyset$)
9          $IRO\_Pair$ = getIROPairs($IL, o, o, CT, L$)          /* Algorithm 2 */
9          $IRO\_Group$ = getIROGroups($IL, o, o, CT, L', K_o$) /* Algorithm 3 */
10 $RL$ = $ISO\_Result \cup IRO\_Pair \cup IRO\_Group$;
Function initRank($o, K_o, KW, HT$)
1  if ($o$ not in $HT$)
2      HT.put(o.id, computeISORank(o,Q,KW));
Function computeRank($o, oList$)
1   foreach object $o' \in oList$
2     $K_{o'}$ = getKeywords($IL, o'$) ;
3     if ($K_{o'} == KW$)              /* $o'$ is an ISO object */
4        initRank($o'$,$K_o$,$KW$,$HT$);    $ISO\_Result$.add($o'$);
5     else if($K_{o'} \neq \emptyset$ AND ($K_{o'} \cup K_o == KW$))     /* $o'$ is IRO object    */
6        initRank($o'$,$K'_o$,$KW$,$HT$);
7        $IRO\_Pair$.add($o$,$o'$);
8        initRank($o$,$K_o$,$KW$,$HT$);    /* $o$ is an IRO object also */
9        updateIRORank($o$, $o'$,$oList$, $HT$);
Function updateIRORank ($o$, $o'$, $oList$, $HT$)
1  update the $IRORank$ of $o$ based on Equation 4.5−4.7;
2  put the updated ($o$, $IRORank$) into $HT$;
```

Algorithm 4.2: getIROPairs ($IL[m]$, src, o, CT, L)

```
   /* find all counterparts of $o$ captured by IRO pair        */
1   if $L == 0$ then   return ;
2   let oList = getConnectedList(o,CT) ;
3   computeRank($o, oList$) ;
4   let ancList = getParent(o) ;
5   computeRank($o$, $ancList$) ;
6   let desList = getChildren(o) ;
7   computeRank($o$, $desList(o)$) ;
8   $L$ = $L$ - 1 ;
9   foreach $o' \in$ ($oList \cup ancList \cup desList$) s.t. $o'$ is not IRO object yet
10      getIROPairs($IL, src, o', CT, L$) ;
```

Algorithm 4.2 shows how to find all objects that form IRO pair with an IRO object src. It works in a recursive way, where input o is the current object visited, whose initial value is src. Since two objects are connected via either a reference or containment connection, line 2-3 deal with the counterparts of o via reference connection by calling $getConnectedList()$; line 4-7 deal with containment connection. Then it recursively finds such counterparts connecting to src indirectly in a depth limited search(line 8-10). $getIROGroups()$ in Algorithm 4.3 works in a similar way.

Algorithm 4.3: getIROGroups ($IL[m]$, o, CT, L', K_o)

```
/* find all counterparts of o captured by IRO group     */
1   let $KS = \emptyset$; $count$ = 0;
2   cList = getConnectedList($o, CT, L'$);
3   **for** $n$= 1 to $L'$ **do**
4     **foreach** $o' \in cList$ **do**
5       $KS$ = getKeywords($IL, o'$) $\cup$ $KS$;
6       **if** ($KS \subset KW$) **then**
7         count++; continue;
8       **elseif** ($count$>2) **then**
9         initialize group $g$ containing such $o$ and $o'$;
10        $IRO\_Group$.add($o$,$g$);
```

The time complexity of $KWSearch$ algorithm is composed of three parts: (1) the cost of finding all IRO pairs is: $\sum_{o \in L_s} \sum_{i=1}^{L} |cList_i(o)| * \sum_{j=1}^{k} \log |L_j|$, where L_s, o, $|cList_i(o)|$, k and $|L_j|$ represent the shortest inverted list of query keywords, an object ID in L_s, length of the list of objects forming an IRO pair with o with chain length = i (limited to L), the number of query keywords, and the length of the jth keyword's inverted list respectively. (2) the cost of finding all IRO groups is: $\sum_{o \in L_s} \sum_{i=1}^{L'} |Q_{L'}| * \sum_{j=1}^{k} \log |L_j|$), where the meaning of each parameter is same as part (1), and $|Q_{L'}|$ denotes the maximal number of object trees reached from o by depth limited search with chain length limit to L'. (3) the cost of finding all ISO objects is: $O(\sum_{o \in L_s} \sum_{j=1}^{k-1} \log |L_j|)$. The formation of each cost can be easily derived by tracing Algorithm 4.1-4.3. The overall complexity of *KWSearch* algorithm is $\sum_{o \in L_s} \sum_{i=1}^{L} (|cList_i(o)| + |Q_L|) * \sum_{j=1}^{k} \log |L_j|$, assuming $L > L'$.

4.7 Experimental Evaluation

Experiments run on a PC with Core2Duo 2.33GHz CPU and 3GB memory, and all codes are implemented in Java. Both real dataset DBLP(420 MB) and synthetic dataset XMark(115 MB) [3] are used in experiments. The inverted lists and connection table are created and stored in the disk with Berkeley DB [1] B+ trees, a summary of indices built is shown in Table 4.1. An online demo [15] of our system on DBLP, namely ICRA, is available at http://xmldb.ddns.comp.nus.edu.sg.

Table 4.1: A summary of Indices

Data	File size	Keyword inverted lists		Connection table	
		creation time	size	creation time	size
DBLP	420MB	783 sec	214MB	94 sec	1.91MB
XMark	114MB	315 sec	141MB	270 sec	13.8MB

Table 4.2: Recall Comparison

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

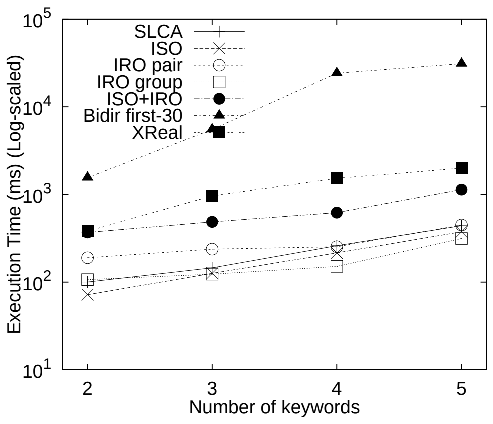
4.7.1 Effectiveness of ISO and IRO Matching Semantics

In order to evaluate the quality of our proposed ISO and IRO semantics, we investigate the overall recall of ISO, ISO+IRO[3] with XSeek [80], XReal [16] and SLCA [118] on both DBLP and XMark. 20 queries are randomly generated for each dataset, and the result relevance is judged by five researchers in our database group. From the average recall shown in Table 4.2, we find: (1) ISO performs as well as XReal and XSeek, and is much better than SLCA. It is consistent with our conjecture that the search target of a user query is usually an object of interest, because the concept of object indeed is implicitly considered in the design of ISO, XReal and XSeek. (2) ISO+IRO has a higher recall than ISO alone, especially for queries on XMark, as there are more ID references in XMark that bring more relevant IRO results. In general, IRO semantics do help find more user-desired results while the other semantics designed for tree data model cannot.

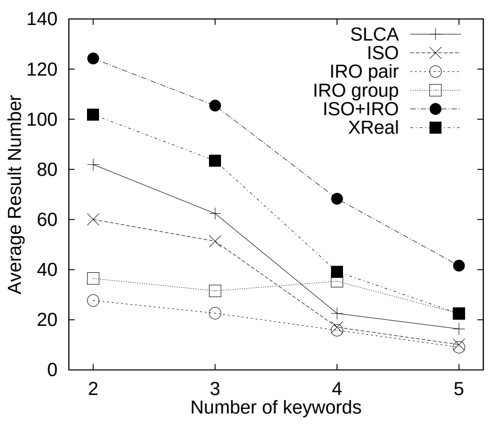
4.7.2 Efficiency & Scalability Test

Next, we compare the efficiency of our approach with SLCA and XReal [16] in tree model, and Bidirectional expansion [65] (Bidir for short) in digraph model. For each dataset, 40 random queries whose lengths vary from 2 to 5 words are generated, with 10 queries for each query size. The upper limit of connection chain length is set to 2 for IRO pair and 1 for IRO group, and accordingly we modify Bidir to not expand to a node of more than 2-hops away from a keyword node for a fair comparison. Besides, since Bidir searches as small portion of a graph as possible and generates the result during expansion, we only measure its time to find the first 30 results. The average response time on cold cache and the number of results returned by each approach are recorded in Figure 4.2 and 4.3.

[3]ISO+IRO represents a result set, each of which is a matching result under either ISO semantics or IRO semantics.



(a) Execution time



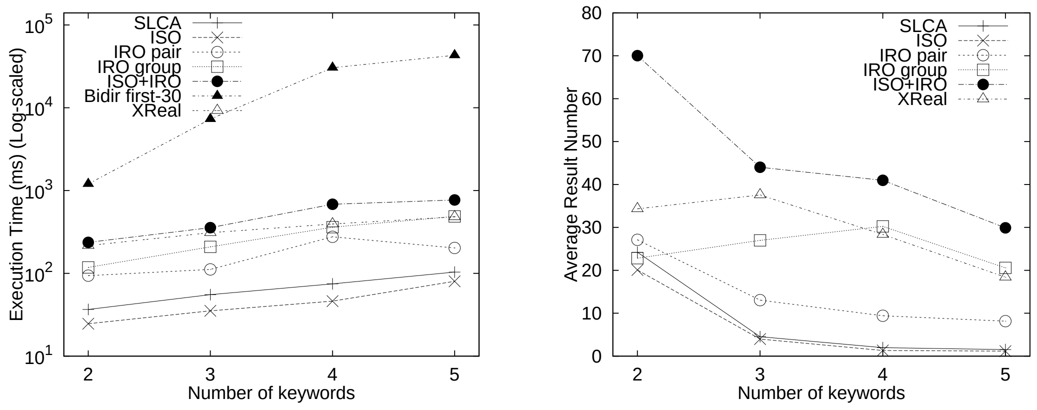
(b) Total result number

Figure 4.2: Efficiency and scalability tests on DBLP

The log-scaled response time on DBLP is shown in Figure 4.2(a), and we find: (1) Both SLCA and ISO+IRO are about one order of magnitude faster than XReal and Bidir for queries of all sizes. SLCA is twice faster than ISO+IRO, but considering the fact that ISO+IRO captures much more relevant results than SLCA (as evident from Table 4.2), such extra cost is worthwhile and ignorable. (2) ISO+IRO scales as well as SLCA w.r.t the number of query keywords, and ISO alone even has a better scalability than SLCA.

(3) Bidir approach (where we only count the response time of getting the first-30 results) has the least efficiency.
The reasons is threefold: *First*, at each expansion, Bidir needs to find the best node to expand among all expandable nodes in order to quickly find the next result. *Second*, when Bidir computes or updates the goodness score of a node, it has to recursively propagate the goodness to all neighbors to improve their goodness until no nodes' goodness can be improved. *Third*, Bidir involves the floating point numbers in computing and comparing the goodness of expandable nodes.

From Figure 4.2(b), we find the result number of ISO is a bit smaller than that of SLCA, as ISO defines qualified result on (more restrictive) object level. Besides, ISO+IRO finds more results than SLCA and XReal, because many results that are connected by ID references can be identified by IRO.



(a) Execution time (b) Total result number

Figure 4.3: Efficiency and scalability tests on XMark

The efficiency result of each approach on XMark is shown in Figure 4.3, which is similar to that on DBLP; thus we ignore the detailed discussion. One observation worth notifying is that, the result number increases when number of keywords increases from 2 to 4 and then drops for IRO group. It is because the 40 random queries we choose for each keyword number differ with each other, and in average the 40 queries of 4 keywords have more matching groups of object trees that are meaningfully connected by IDRefs than other batches of query by coincidence.

4.7.3 Effectiveness of the Ranking Schemes

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

Here, we compute the R-rank and MAP for top-30 results returned by ISO, IRO and XReal, by issuing the same 20 random queries as describe in section 4.7.1 for each dataset. Specificity factor $s(o, Q)$ is ignored in computing $ISORank$; in computing the $IRORank$, w_1 = 1 and w_2 = 0.7 are chosen as the weights in Equation 4.5. The result is shown in Table 4.3. As ISO and XReal do not take into account the reference connection in XML data, it is fair to compare ISO with XReal. We find ISO is as good as XReal in term of both R-rank and MAP, and even better on DBLP's testing. The ranking strategy for IRO result also works very well, whose average R-rank is over 0.88.

Table 4.3: Ranking Performance Comparison

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

Besides the random queries, we choose 7 typical sample queries as shown in Table 4.4, which contain various search needs: $Q1$ intends to search for publications co-authored by two authors; $Q2$-$Q4$ intends to search for publications on a certain topic by a certain author; $Q5$ intends to search for publications of a particular author on a certain conference; $Q6$-$Q7$ search for publications about a certain topic.

Table 4.4: Sample queries on DBLP

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

In particular, we compare our system [15] with some academic search engines such as Bidir in digraph model [65], XKSearch employing SLCA [118] in tree model, with commercial search engines, i.e. Google Scholar and Libra[4]. Since both Scholar and Libra can utilize abundant of web data to find more results than ours whose data source only comes from DBLP, it is infeasible and unfair to compare the total number of relevant results. Therefore, we only measure the number of top-k relevant results, where k=10, 20 and 30.

Table 4.5: sample query result number

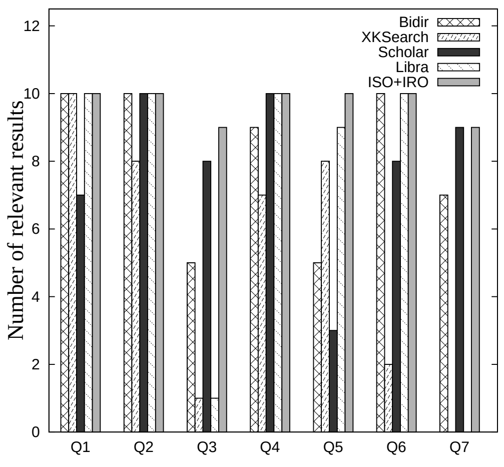
Query	ISO result	IRO Result
Q1	16	42
Q2	14	58
Q3	1	56
Q4	14	230
Q5	8	1238
Q6	3	739
Q7	0	93

Since our system separates ISO results and IRO results (as mentioned in section 4.3.3), top-k results are collected in the way that, all ISO results are ordered before the IRO results. The total number of ISO results and IRO results are shown in Table 4.5, and the comparison for the top-30 results is shown in Figure 4.4.

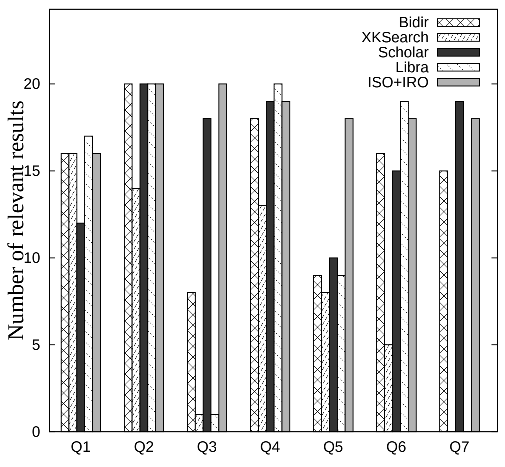
First, we compare ISO+IRO with Bidir and XKSearch. For queries that have both ISO and IRO results (e.g. $Q1$-$Q6$), our approach can find more relevant results, and rank them in most of the top-30 results. E.g. for $Q5$, Bidir does not work well, because it treats two authors coauthored in one paper, and one author writes a VLDB paper, as a result; however, our approach puts citation relationship between papers as an important matching condition. Note that, there is no ISO result for $Q7$, XKSearch also returns nothing; but 26 IRO results are actually relevant.

Second, we compare ISO+IRO with Libra and Scholar. From Figure 4.4, we find our approach is comparable with Scholar and Libra for all sample queries. In particular, ISO+IRO is able to rank the most relevant ones in top-10 results for most queries, because its top-10 precision is nearly 100% for most queries, as evident in Figure 4.4(a). In

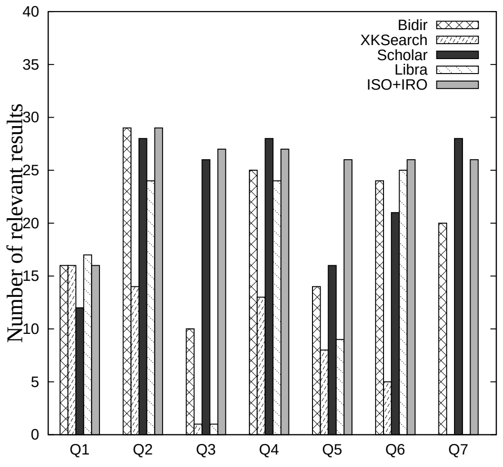
[4]Google Scholar: http://scholar.google.com. Microsoft Libra: http://libra.msra.cn



(a) Top-10 Results



(b) Top-20 Results



(c) Top-30 Results

Figure 4.4: Result quality comparison

addition, as Libra only supports keyword conjunction (similar to our ISO semantics), it does not work well for $Q3$ and $Q7$, as there is only 1 and 0 result containing all keywords for $Q3$ and $Q7$. As shown in Figure 4.4(a), Scholar only finds 3 relevant results for $Q5$ in its top-10 answers, probably because keywords “Jim” and “Gray” appear in many web pages causes many results that don’t contain “VLDB” to still have a high rank, which is undesired. Moreover, due to information loss of citation and content of publication in DBLP data file, it may cause some relevant result not to be found by ISO+IRO. Scholar may find such results due to its power to search the whole web information. E.g. the top-30 relevant result for $Q4$ by ISO+IRO is smaller than that by Scholar.

Thirdly, as shown in Figure 4.4, the average recall for each query generated by our ISO+IRO is above 80% at each of the three top-k levels, which confirms its advantage over any other approach.

4.8 Summary

In this chapter, we have built a preliminary framework for object-level keyword search over XML data, by taking into account the ID references missed in tree data model. In particular, based on the semantic information about the objects of interest available to us, we model XML data as the interconnected object-trees, where each object-tree is in form of a subtree representing a single object of interest. Then, based on this model, we propose two object-level matching semantics that are close to user's search concern, namely ISO (*Interested Single Object*) and IRO (*Interested Related Object*). ISO is to capture user's search concern on a single object-tree, while IRO is to capture user's search concern on multiple object-trees which are connected by either containment or reference edge in somehow related way. A customized ranking scheme is proposed by taking both the structure and content of the results into account. Efficient algorithms are designed to compute and rank the query results in one phase, and extensive experiments have been conducted to show the effectiveness and efficiency of our approach. As a future work, we would like to investigate how to distinguish the relationship types among objects and utilize them to define more precise matching semantics.

Although our approach targets at keyword query processing over XML data with IDRef edges, it can nonetheless be easily adapted to processing XML data tree (without taking IDRef into consideration), to solve the problem in chapter 3. The difference is, solutions proposed in chapter 3 can work without knowing the semantic information, while solutions here works assuming the semantic information is known.

CHAPTER 5

CONTENT-AWARE QUERY REFINEMENT IN XML KEYWORD SEARCH

5.1 Introduction

Chapter 3 and 4 focus on how to find relevant and meaningful data fragments in XML data without and with IDRef edges being considered respectively to answer a keyword query, assuming each query keyword is correct and intended as part of it. However, user queries may contain irrelevant or mismatched terms, typos etc, which may easily lead to empty or meaningless results. As reported by [103], web search users have to reformulate their queries at least once 40% to 52% of the time in order to find their desired results, and 10-15% of the queries sent to search engines contain spelling errors. Similarly, keyword search over XML data suffers from the same problems, which draws our attention to the problem of query refinement in the context of XML keyword search.

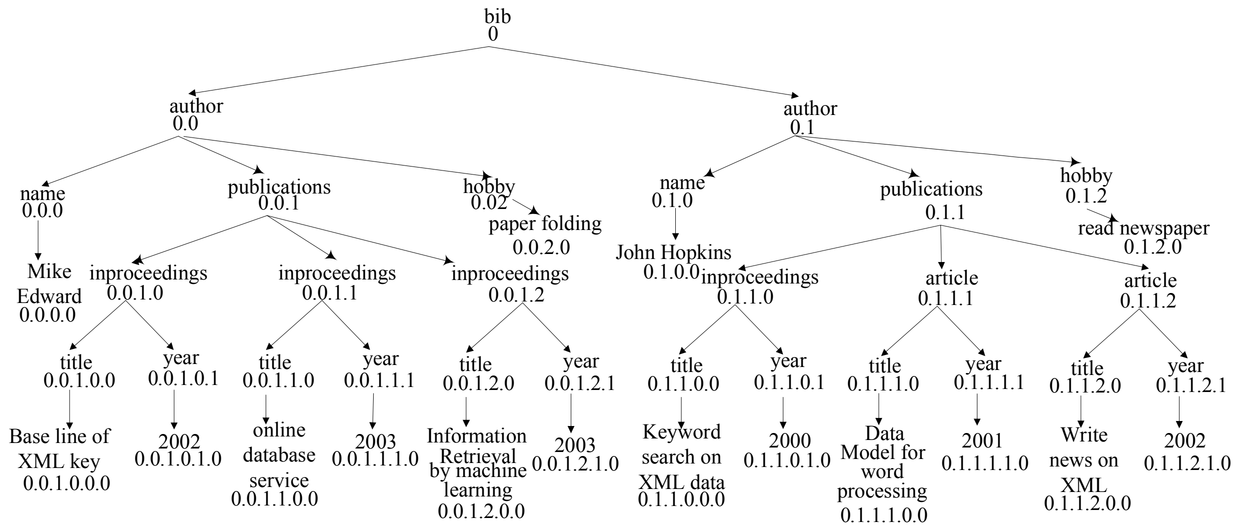


Figure 5.1: Example XML document

A user query may often be an imperfect description of their real information need, which may easily cause an empty matching result or wrong result. Even when the information need is well described, a search engine may not be able to return the results matching the query as expected possibly due to term mismatch, keyword ambiguity or unintentional spelling error etc, as shown in Example 5.1.

Example 5.1. *Consider Q_3={keyword, paper} in Table 5.1 issued on a bibliographic XML document in Figure 5.1 (where both the tags and their content nodes are labeled using dewey labeling scheme [105]), intending to find "papers" about "keyword". By SLCA, the whole XML tree rooted at* bib:0 *is returned (as lca(0.0.2.0, 0.1.1.0.0.0)= 0) due to the occurrence of the ambiguous "paper" at node 0.0.2.0, which contains "paper folding" as a hobby of an author. However, such result is meaningless and irrelevant to user's search intention; moreover, the result is the is overwhelmingly large (i.e. the whole document) for user to consume. R_{Q_3}={keyword, inproceedings} is a potential refinement of Q_3, as inproceedings is a synonym of paper in this context.* □

Besides the above scenarios that necessitate the query refinement, another frequently encountered scenario in XML keyword search is, a user query may be too restrictive to have a meaningful matching result. E.g. consider Q_4={*XML, John, 2003*} in Table 5.1 issued on Figure 5.1, intending to find John's publication about XML in year 2003. However, the only result covering all the keywords is the root node of XML data, which is meaningless to the user. A possible refinement is to delete "2003" from the original query.

Table 5.1: Query before and after refinement

Initial query	**Suggested Refined query**
Q_1: IR, 2003, Mike	R_{Q_1}: Information Retrieval, 2003, Mike
Q_2: Mike, publication	R_{Q_2}: Mike, publications
Q_3: keyword, paper	R_{Q_3}: keyword, inproceedings
Q_4: XML, John, 2003	R_{Q_4}: XML, John
Q_5: mechin, learn	R_{Q_5}: machine, learning
Q_6: hobby, news, paper	R_{Q_6}: hobby, newspaper
Q_7: on, line, data, base	R_{Q_7}: online, database

In the scenario of web search, there are often a large number of documents to (partially) match query keywords, and query refinement is carried out to make the query result more specific. In contrast, XML keyword search focuses on finding only few meaningful and relevant fragments of an XML document, and a conjunctive search semantics is enforced which may easily lead to nonsensical or empty result. Therefore, in this chapter we focus on a particular direction - the initial query has no meaningful matching result over XML data, and needs to be refined to a closely related query that has meaningful matching results.

Now, the question becomes whether we can offer a solution during search, which peruses the content of XML data being queried and refines the queries that have no (meaningful) matching result, in order to better represent users' search needs and help users more easily find the relevant information, without an initial result retrieval or any intervention on user part. This is called *XML keyword query refinement* as addressed in this work. In particular, there are four critical issues to be addressed.

Issue 1: It should be able to adaptively and judiciously decide whether the initial query Q needs to be refined during the processing of Q.

Issue 2: It should find a list of refined query (RQ) candidates, where each RQ candidate is aware of the contents of the corresponding query answers and assured to have meaningful matching results over the XML data.

Issue 3: *Effectiveness* - It should be able to provide a query ranking model that closely relates the refined query to the XML data being queried in evaluating the quality of the RQ candidates found in Issue 2.

Issue 4: *Efficiency* - In addressing the above three issues especially Issue 2, it should be able to scan the corresponding keyword inverted lists as few times as possible (optimally only once).

In resolving Issues 1 and 2, there are three challenges. The *first* challenge is to define what a *meaningful query result* should be for an XML keyword query, as it will be used to judge whether a query needs to be refined. Compared to the traditional IR-style keyword search whose search target is usually the flat documents, the search target of an XML keyword query is usually implicit or unknown [16], which makes the problem more difficult to solve. The *second* challenge is, it cannot decide in advance whether query refinement is required or not before processing the initial query. A brute force approach needs to submit a query for an initial result retrieval, before deciding whether the refinement should be used [64]. However, it is a prohibitively expensive operation to answer one query by evaluating several potential RQs one at a time, as it has to scan the related keyword inverted lists multiple times, which defeats the primary efficiency goal as requested in Issue 4. The *third* challenge is how to generate appropriate RQ candidates in term of the closeness to user search intention.

Regarding Issue 3, no previous work has touched on building an intuitive query ranking model to evaluate the quality of a refined query in XML keyword search yet. E.g. consider a query Q={database,publication} issued on Figure 5.1, at least two candidates R_{Q_1} = {database,article} and R_{Q_2} = {database,inproceedings} seem to be of equal rank as both are the synonym of "publication", but to know which one has the best match w.r.t Q needs further exploration of the content of XML data being queried.

At first glance people may think there is no big difference for query refinement between web search and XML search, and consider extending the methods for web search to XML. But when one actually attempts to implement such extension, one is faced with myriad options and difficult decisions every step of the way, because of the following three reasons.

1. In web search, result is usually computed in an IR style by adopting a scoring model to judge the similarity between the query and result; while in XML search a strictly conjunctive search semantics and tree structure-preserved result form (which enforces each query keyword to appear in a subtree) are widely adopted.

2. The search target is usually unknown or implicit in an XML keyword query.
   Despite the limitations of existing IR methods in addressing Issues 1-3, a unique challenge for XML keyword search that limits the extension of the above IR methods is: compared to web search whose search target are flat documents, an XML keyword search engine needs to identify the *target node* in XML data (for a keyword query), which is usually implicit and unfixed [16], and it is becoming even more challenging for queries that have no matching result and need to be refined. For example, without figuring out the search target of a query, there seems no way to judge whether a RQ has a meaningful matching result as mentioned in Issue 2, such as Q_4 in Example 5.1. Moreover, it prevents us from extending the machine learning methods (in web search context) to predict high-quality RQs in XML context; and how to incorporate the semi-structured nature of XML in building

such extension remains an open problem.

3. Most previous works designed for web search improve the search relevance by machine learning from a large set of users' search log data [63, 114, 50]. However, a lack of such widely-used commercial XML keyword search engine (with abundant user search activities) prevents us from acquiring a thesaurus of user log data for model training.

Therefore, query refinement in XML keyword search is not just a trivial extension of its counterpart in web search, which motivates us to start this work.

5.1.1 Our Approach

In a nutshell, towards building an automatic query refinement framework that addresses Issues 1-4 as a whole, we novelly integrate the job of looking for the desired RQs and finding the matching results of such RQs as a single problem, namely as the *content-aware* solution. Basically, we achieve such *content-aware* feature in two ways: (1) the RQ candidates are materialized during the procedure of finding the matching results of original query rather than before processing the original query, so that we can ensure each RQ candidate found so far has non-empty matching result. (2) We keep track of a set of RQ candidates and their up-to-date results on the fly, during traversing the related inverted lists for keywords that belong to either initial query Q and its RQ candidates. An immediate benefit of this integration is that user's search experience is significantly enhanced, because from user's perspective, when judging the quality of a RQ, his/her concern is on checking the results of RQ over the XML data, rather than judging from the literal meaning of RQ itself, as even a good refinement may not have any meaningful matching result in the XML data being queried.

As the first step towards our goal, four major refinement operations, namely term deletion, merging, split and substitution are defined in a rule based way, where each particular operation is associated with a basic *dissimilarity* score. In order to judge whether a query needs to be refined, we propose an enhanced notion of SLCA as a criteria by taking the search target node of a query into account, we call it *meaningful SLCA*.

In the second step, we adopt a basic metric to judge the quality of a refined query RQ by computing the accumulated dissimilarity from initial query Q to RQ. As the RQ that has both minimum dissimilarity and non-empty meaningful SLCA result over XML data is unknown ahead, it is prohibitively expensive to infer all RQ candidates and find the one with minimum dissimilarity. Instead, we design a dynamic programming solution to find the RQ with minimum dissimilarity, and the potential Top-K RQ candidates are also produced as its side product.

In the third step, we investigate how to rank the refined queries, especially for those with the same dissimilarity. Basically, we investigate the quality of a RQ from two complementary aspects: the relevance of RQ w.r.t the initial query Q and the dependency of keywords of RQ on XML data D, both of which are able to capture the hierarchical structure of XML data. In particular, we utilize the keyword frequency, co-occurrence, textual similarity between Q and RQ etc. in qualifying the relevance and dependency score.

In the last step, we move to design *efficient* algorithms to find RQs and their matching results, which is another core part. Based on the observation that the document root is never counted as a meaningful SLCA, we propose a *partition-based* approach, in which an XML document tree is logically divided into a sequence of subtrees (in document order) which we call as partitions, and query refinement is sequentially executed on each partition. In this way, the Top-K RQ candidates in each partition can be identified before employing existing methods to find their SLCA results (and extracting meaningful SLCA from those SLCA results, as the meaningful SLCA results of a query Q is a subset of the SLCA results of Q), so unnecessary computation for the RQ candidates that have no meaningful SLCA result can be totally skipped. Besides, those RQ candidates found in the subsequent partitions that have larger dissimilarity than the current Top-K RQs will also be pruned for their SLCA computation, which is an important optimization. Lastly, this approach needs only one-time keyword inverted list scan, manifesting good I/O performance.

Furthermore, we take into account the distribution of keyword frequency, and propose a *short-list eager* approach to start the exploration of Top-K RQs from those that contain the keyword with the shortest inverted list, in order to avoid the full scan of long keyword lists as much as possible. This approach works well when the frequency distribution of query keywords is skewed.

A salient feature of partition-based and short-list eager approach is, they are orthogonal to the methods of finding matching results of a query. If the initial query Q does have meaningful matching result, both of the above algorithms will stop finding refinements and their matching results immediately after the first matching result of Q in XML data is found.

To our best knowledge, this is the first work towards an effective query refinement in XML keyword search, and our major contributions are summarized as below.

- We formally define the problem of *keyword query refinement* in XML keyword search, propose an enhanced notion of the widely adopted SLCA matching semantics to judge whether a query has meaningful matching result, and define four typical refinement operations in a rule-based way.
- We build a query ranking model to evaluate the quality of RQ candidates from a statistical perspective based on tree structural data, by considering the semantical and morphological similarity between RQ and Q, together with the dependency of keywords of RQ in XML data.
- We design a dynamic programming solution to efficiently find the optimal RQ.
- We propose two solutions to achieve an efficient XML keyword query refinement

and results generation: partition-based approach and short-list eager approach, both of which are orthogonal to any existing SLCA computation methods. Moreover, the partition-based approach needs only one-time scan of the corresponding keyword inverted lists.

- We conduct extensive experiments to show the efficiency and effectiveness of our refinement framework, by using the real-life data sets and real-life user queries.

The rest of this chapter is organized as below. Section 5.2 defines the meaningful SLCA. Section 5.3 presents our query ranking model. Section 5.4 presents an efficient approach to find the optimal RQ. Section 5.5 presents two dynamic query refinement approaches. Section 5.6 discusses the index construction for efficient refinement. Experimental result is reported in section 5.7 and we summarize our work in section 5.8.

5.2 Preliminaries

Same as Chapter 3, we model XML data as a rooted, labeled tree using dewey labeling scheme, and a keyword query Q=$\{k_1$, k_2, ..., $k_n\}$ is treated as an ordered *sequence* of terms separated by commas.

5.2.1 Meaningful SLCA

In recent literature, the notion of Smallest Lowest Common Ancestor (SLCA) [118, 80] has been suggested as an effective way to identify the segments of interest from XML data for a keyword query. However, a unique feature of XML keyword search is to identify the target that user intends to search for [16] while SLCA and its variations cannot resolve thoroughly, as mentioned in Chapter 3.

Therefore, we define the concept of *meaningful SLCA*, which is the SLCA that is aware of the search targets. A typical non-meaningful SLCA is the document root, as it makes no sense to return the whole XML document to user. Consider a query “database, model" issued on XML data in Figure 5.1, the only SLCA is the root node *bib:0*, which user never expects to have.

Regarding how to identify the target that a user desires to *search for*, recall that in Chapter 3 we have defined the concept of *node type*, *search for node* and *XML DF* (XML Document Frequency). These definitions will be utilized in this chapter again, and please refer to section 3.2 for details.

Formula 5.1 is designed to measure the confidence of a node type T to be the desired *search for node* w.r.t a given query Q, with same intuition as Formula 3.6, except that we use *Sum* of XML DF (f_k^T) to combine the statistics of all keywords for each node type T, as some query keywords may not appear in the XML data for ill-formed query. r is a reduction factor ranging in (0,1); $depth(T)$ represents the depth of T-typed nodes in XML data.

$$C_{for}(T, Q) = \log_e(1 + \sum_{k \in Q} f_k^T) * r^{depth(T)} \tag{5.1}$$

Choosing the desired search for node

Due to the keyword ambiguity problem as mentioned in chapter 3, different people may issue the same query for different search intentions, so the search for node may not be unique. In the worst case, the number of search for node candidates can be as large as the number of distinct *node types* of an XML document, an appropriate threshold δ should be set to filter those that cannot be a promising candidate. We choose the desired search for node in two steps.

In the first step, we believe there should be at least one subtree (in the XML data tree) that contains at least one keyword of the initial query. Thus, the lower bound of Formula 5.1 is δ=ln(1+1)*$r^{depth(D)}$, where $depth(D)$ denotes the depth of the XML data tree D. As a result, those T whose search for confidence is above δ will be chosen as the *search for node candidates*.

In the second step, we try to get only the promising ones from those resulted in step 1. We first sort all the above candidates and pick the node type with the highest confidence, say T_{max}. Then we compute the relative difference percentage of the confidence scores of the remaining node types with $C_{for}(T_{max}$,$Q)$, and choose those whose difference percentage is within a given threshold σ as one of the final *desired* search for node candidates. Note that there is no one-fit-all threshold value, and our empirical study demonstrates that a value of σ=30% has an overall reasonable and effective performance.

As a result, those *desired* search for node candidates will be used to further constrain the meaningfulness of an SLCA result of a query, which is shown in Definition 5.1. Regarding the tunable value of r in Formula 5.1, there is no one-fit-all choice, and through our empirical study, a choice of r=0.8 works well in general.

Definition 5.1. *A node n is a* meaningful SLCA *of query Q on XML document D, if all the following properties hold:*

1. *n∈SLCA(Q,D)[118] (i.e. n contains all the query keywords in either its labels or the labels of its descendants, and has no descendant that also contains all the query keywords).*
2. *n is not the root node of XML document D.*
3. *n is a self or descendant of one of the* search for node *candidates T inferred by Formula 5.1 and above a give threshold δ.*

We can find that, if we return user the meaningful SLCA and its subtree, it may not provide enough relevant information, as it is not closely related to user's search target. What user expects to be returned is as below.

Definition 5.2. ***(Meaningful Result)*** *For a certain meaningful SLCA n of query Q over XML document D, the corresponding meaningful result (output to user) should be the subtree rooted at node m, where among all promising search for node candidates T satisfying property 3 of Definition 5.1, m is the structurally nearest T to n.*

As a quick example, consider Q_6={hobby,news,paper} in Table 5.1 issued on Figure 5.1. By formula 5.1, *author* is a promising search for node candidate of Q_6. The only SLCA result of Q_6 is the root node *bib:0*, which violates property 2 in Definition 5.1. Thus, Q_6 does not have any meaningful SLCA. In contrast, the only SLCA of R_{Q_6}={hobby,newspaper} is *hobby:0.1.2* which is a descendant of *author*; so it is a meaningful SLCA, and the output of R_{Q_6} is the subtree rooted at *author:0.1* (by Definition 5.2).

Now, we would like to argue the rationality of the properties as described in Definition 5.1 to trigger a query refinement. For a given query Q, whether Q needs refinement may vary from users, as different people may have different search intentions even when they issue the same query. However, despite of the *subjective* search intention issue, we observe that no matter which user issues a keyword query on an XML data tree, she is interested in particular fragments, rather than the whole XML data tree that is overwhelmingly large for user to consume. Therefore, if all the matching results of Q are the root node of the XML tree, it is certain that Q needs refinement. It naturally drives us to impose property 2 in Definition 5.1. Moreover, when a user issues a query, she usually has her search target in mind ahead, though the search condition (that is used to constrain the resulted instances of the search target) may have various interpretations. Therefore, if all the potential search targets that a user query may intend to search for can be inferred, we can further constrain the condition to trigger the refinement. As a result, property 3 is specified to ensure the result is related to one of the potential search targets. In other words, Definition 5.1 indeed describes an *objective* bottom line that necessitates a query refinement.

Furthermore, we would like to discuss the *flexibility* of Definition 5.1 in two aspects. (1) In this work, Definition 5.1 is defined to fit the SLCA matching semantics [80], but it is not confined to SLCA only. Indeed, it can be easily adapted to accommodate to

any other matching semantics proposed for XML data tree model, such as LCA [52], MSLCA [79] etc, because only property 1 in Definition 5.1 needs to be adjusted.
(2) Besides the above two mandatory properties for meaningful SLCA, an optional property is: "The element denoted by n is of either entity category or attribute category[1] according to the *Entity inference rule* (with the aid of DTD of XML data) defined in XSeek [80]." This property is designed to coincide with the fact that user's search target is normally at real entity level. E.g. in Figure 5.1, people's concern is on *author*, *inproceedings* and *article* which are of entity category, while *publications* is not.

Lastly, people usually mistakenly take the (meaningful) SLCA node as the output of a query. Indeed, as claimed in Definition 5.2, a desired output of a query is the subtree rooted at the search for node, which is the self-or-ancestor of a meaningful SLCA node.

Definition 5.3. *A keyword query Q is said to* need refinement *if Q does not have any* meaningful SLCA *on XML document D.*

5.2.2 Refinement Operations

As reported by the web query logs tracing users' search modifications [63], a frequently used strategy by users is deleting terms, presumably to obtain greater coverage; while term substitution is the major strategy adopted by search engines. Besides, we observe that there are four potential sources that frequently cause ill-formed queries: (1) queries may contain misspelled or mismatched words (e.g. Q_1-Q_3,Q_5 in Table 5.1), (2) mistakenly split words (e.g. Q_6,Q_7 in Table 5.1), (3) mistakenly merged words (such as queries in Table 5.5), (4) queries that contain strong conjunctive constraints have no match against a small corpus (e.g. Q_4 in Table 5.1). These four ill-forms cause the results of the initial query either empty or nonsensical.

[1]Same as [80], when the SLCA of a query is an attribute node, we return its associated entity as SLCA result instead.

As a result, four refinement operations, i.e. *term substitution*, *term merging*, *term split* and *term deletion*, are defined to increase the query coverage. E.g. a query {online, newspaper} may often be written as {on,line,news,paper} by user, which needs term merging. Term deletion is employed presumably to obtain greater coverage, as queries with no matches can have words deleted till a match is obtained, such as Q_4 in Table 5.1. *Term substitution* is wide-ranging, which mainly includes spelling error correction (Q_5 in Table 5.1), synonym substitution (Q_3), acronym expansion (Q_1) and word stemming (Q_2). Note that, we can include more refinement operations if needed, as it is an orthogonal problem to our query refinement methods as introduced in Section 5.5; however, we believe the above four operations are enough to serve the general purpose to cover most refinement jobs in real world.

Definition 5.4. *A refinement rule instance*[2] r *associated with a refinement operation* op *is in form of:* $S_1 \underset{op}{\rightarrow} S_2$*, where* S_1 *and* S_2 *are two keyword sequences, and* r *has an associated dissimilarity score* ds_r*, which models the dissimilarity between* S_1 *and* S_2*.*

Table 5.2 lists some refinement rule instances. In particular, for term merging, term split and spelling error correction, ds_r can be a variant of the morphological metric such as the string edit distance between the LHS and RHS of rule r. E.g. the dissimilarity ds_r of a one-time term merging or split is 1, as a single space is removed/added, such as r1, r2, r4 and r7 in Table 5.2. For r5, ds_{r5} = 2 as two string edits are needed to correct the spelling error.

Since the *term substitution* is wide-ranging, which may include synonym substitution, typo correction, acronym expansion etc, the dissimilarity varies correspondingly. For synonym substitution rule such as r3 in Table 5.2, ds_r can be the similarity score provided by checking against a corpus of the known database tokens or the semantic lexical database such as WordNet [44]. For instance, in WordNet the dissimilarity score between two words a and b is the length of the path from a's belonging synset to b's

[2]Without ambiguity, we use the term "rule" to denote rule instance in the rest of this chapter.

Table 5.2: Sample Refinement Rule Instances with its dissimilarity score

	Operation	Example	ds_r
r1	Merging	on,line → online	1
r2	Merging	data,base → database	1
r3	Substitution	article → inproceedings	1
r4	Merging	learn,ing → learning	1
r5	Substitution	mechin → machine	2
r6	Substitution	WWW → world,wide,web	1
r7	Split	online → on,line	1

belonging synset; two synonyms in the same synset has a dissimilarity score of 1, such as r3. For acronym expansion such as r6 in Table 5.2, a score of 1 is designated.

It is out of the scope of this work to study a normalized measurement scheme that can well handle the dissimilarity caused by either the semantic similarity or the literal similarity. Therefore, in order to minimize the effect of the dissimilarity assignment in evaluating the effectiveness of the query ranking model (as proposed in section 5.3), we designate a uniform dissimilarity score for all refinement rule instances (that invoke a single refinement operation) except for term deletion. Besides, in this work we do not consider the recursive refinement which further applies refinement rule(s) on the newly generated keywords.

Since term deletion has the greatest potential in changing the meaning of initial query, we adopt the principle that its dissimilarity score is greater than any other three rules throughout this chapter[3]. The refinement rules can be obtained from data mining, query log analysis [63] or manual annotation [50]. However, how to generate these rules is orthogonal to this work. Lastly, we define the dissimilarity between an initial query Q and a refined query RQ.

Definition 5.5. *Given a set R of refinement rule instances, the dissimilarity between Q and a RQ, denoted as $dSim(Q, RQ)$, is the minimum of the sum of the cumulated ds_r*

[3]To facilitate our discussion, the dissimilarity score of a single term deletion rule is 2 throughout all examples in this chapter.

among all possible sequences of application[4] *of the rule instances in R to transform Q into RQ.*

5.3 Ranking of Refined Queries

In section 5.2.2, *dSim(Q,RQ)* is defined as a preliminary quality metric for a RQ, mainly based on its lexical and morphological similarity w.r.t Q. However, it is inadequate without considering the local context (i.e. the XML data) being queried, especially for those RQs that have the same dissimilarity. Motivated by the ability of statistics in modeling patterns or drawing inferences about the underlying data, we aim to utilize the statistic knowledge of underlying XML data to build an in-depth *content-aware* query ranking model. In general, the overall quality of a RQ can be evaluated in two complementary aspects: (1) the *similarity score* of RQ which captures the relevance of RQ w.r.t the initial search intention, and (2) the *dependency score* of RQ which captures the keyword dependencies of RQ in XML data D.

We begin with introducing some statistic notations used in later discussion. $tf(k, T)$ denotes the *term count* of k in all the subtrees rooted at node type T; F_T denotes the number of distinct terms contained in either the values or tags of all the subtrees rooted at node type T, while the stop words are omitted. For example in Figure 5.1, tf(”XML”,*author*)=3, as “XML” appears 3 times within the subtrees rooted at *author*; $F_{article}$=14 as there are 14 distinct keywords within the subtrees rooted at *article*. Note that these statistics data can be pre-computed in parsing the XML data.

5.3.1 Similarity Score of a RQ

Given a query Q issued on an XML document D and a RQ candidate, we propose four intuitive guidelines in an incremental way to compute the similarity of RQ w.r.t the initial search intention.

[4]Recursive application of rule instances are not considered.

Guideline 1: Keyword frequency. The more frequently the keyword in RQ appears within a search for node type T, the more important RQ is. □

Following Guideline 1, Formula 5.2 is designed to accumulate the term frequencies of all keywords in RQ, and F_T is chosen as a normalization factor to prevent a bias to a search for node type T whose subtrees are of large size.

$$Imp(RQ, T) = \sum_{k \in RQ} \frac{tf(k, T)}{F_T} \tag{5.2}$$

In addition, we notice that each keyword in a query has its own ability to discriminate the query results, i.e. each $k \in Q$ as a constraint of Q actually has different importance. Take term deletion as an example (Example 5.2), k_i is one of the keywords that are deleted from Q to form a RQ. Thus, the less frequent k_i appears in the subtrees rooted at T-typed nodes, the more discriminative k_i is to Q, i.e. the RQ resulted from deleting this k_i from Q is less favored.

Example 5.2. *Consider Q=\{XML, twig, pattern, join\} issued on DBLP, where no matching result is found. Suppose* inproceedings *is a search for node candidate. Then, by deleting "join" and "pattern" respectively, we get two candidates R_{Q_1}=\{XML, twig, pattern\} and R_{Q_2}=\{XML, twig, join\}, where dSim(R_{Q_1},Q)=dSim(R_{Q_2},Q) as only one term deletion is adopted. Though, $f_{pattern}^{inproceedings}$=17297*[5] *is much larger than $f_{join}^{inproceedings}$ = 946, i.e. "join" and "pattern" have different importance as a constraint of Q, which also should affect the similarity of the resulted RQ.* □

Guideline 2: Importance of keyword. The more *discriminative* a keyword k_i (that is either deleted from the initial query Q or newly generated by the term merging, term split and/or term substitution rules) is w.r.t the initial query Q, the lower the rank of RQ, which is resulted from the corresponding refinement involving k_i, should be assigned. □

[5]These statistics can be validated by an online demo of our previous work [15] at http://xmldb.ddns.comp.nus.edu.sg

Recalling Definition 3.8, the *XML DF* f_k^T provides an effective way to measure the discriminative power of a keyword k, as Guideline 2 is in line with the design intuition of document frequency. In other words, the less *XML DF* of a keyword k_i (i.e. $f_{k_i}^T$) is, the more discriminative this k_i is w.r.t. Q. As a result, Formula 5.3 is designed to address Guideline 2 alone.

$$Imp_{k_i}(Q,T) = \log_e \frac{N_T}{1+f_{k_i}^T} \tag{5.3}$$

where N_T is the total number of nodes of type T in XML document D. The *log* function is applied to normalize the raw ratio.

Guideline 1 favors the RQ whose keywords have large term frequencies, which is analogous to the intuition of *TF* part in *TF*IDF* definition; while Guideline 2 favors the RQ whose deleted or newly generated keyword has the largest importance in the initial query, which is analogous to the intuition of *IDF* part. Therefore, we define the similarity $\rho(RQ, Q|T)$ of a RQ w.r.t Q, for a given search for node type T in Formula 5.4, where the first multiplier manifests Guideline 1, and the second multiplier manifests Guideline 2 by accumulating the importance of k_i involved in refining Q to RQ.

$$\rho(RQ, Q|T) = Imp(RQ,T) * \sum_{k_i \in (RQ \triangle Q)} Imp_{k_i}(Q,T) \tag{5.4}$$

Here, $RQ \triangle Q$ denotes a set of keywords that are either deleted from Q or newly generated by term merging, term split or term substitution rules to produce RQ.

So far, we have only considered the case that the search for node candidate T of a query is unique. However, the keyword ambiguity problem [16] may cause more than one T to have comparable and promising search for confidence by Formula 5.1, as discussed in section 5.2.1. Motivated by this fact, Guideline 3 is proposed.

Guideline 3: Confidence as a desired search target. For an initial query Q that has multiple desired search for node candidates T, the confidence $C_{for}(T,Q)$ of each such T should be taken into account. The higher the confidence of T as a desired *search for node* is, the more important its associated similarity $\rho(RQ, Q|T)$ is. □

Therefore, we incorporate the confidence of T as the desired *search for node* (i.e. Formula 5.1) and its similarity as below.

$$\rho(RQ, Q) = \sum_{T \in T_{for}} C_{for}(T, Q) * \rho(RQ, Q|T) \tag{5.5}$$

where T_{for} denotes a set of candidates of the desired search for node. Note that, Guideline 3 holds based on the principle that both the initial and refined query share the same search for node(s).

We argue that the design of Formula 5.5 is reasonable, because firstly the literal difference between Q and RQ are usually small (1-2 keyword difference); secondly, the use of logarithm function in search for confidence computation (in Formula 5.1) ensures a small confidence difference between Q and RQ, as validated by empirical study in section 5.7.

Lastly, by taking the semantic and morphological dissimilarity $dSim(Q, RQ)$ between Q and RQ into account, whose intuition is mentioned in Guideline 4, the similarity of a RQ w.r.t the initial query Q is presented in Formula 5.6.

Guideline 4: Textual/Semanitc dissimilarity. The smaller the dissimilarity between Q and RQ is, RQ is closer to Q in term of the search intention. $\square$

Continuing with Example 5.2, if two keywords are deleted from Q, there are 6 more RQs to be generated. Although these queries have more results than R_{Q_1} and R_{Q_2}, they are farther to Q in term of search intention. Thus, we also incorporate the dissimilarity score into our query ranking model.

As a result, we define Formula 5.6, using the weighted sum of the dissimilarity between Q and RQ w.r.t a given search for node candidate T and the confidence of T as the desired search for node as the final similarity between RQ and Q.

$$\rho(RQ, Q) = w^{(dSim(Q,RQ))} * \sum_{T \in T_{for}} C_{for}(T, Q) * \rho(RQ, Q|T) \tag{5.6}$$

where $dSim(Q, RQ)$ denotes the dissimilarity between Q and RQ as described in Definition 5.5. w is a decay factor ranging in (0,1) to enforce Guideline 4, and w=0.7 is a good choice as evident by our empirical study in section 5.7.4.

5.3.2 Dependency Score of a RQ

In evaluating the quality of a RQ, the above similarity function emphasizes the relevance between Q and RQ, which has a limitation: the query terms are assumed to be mutually independent. As a complementation of the similarity score, if RQ contains more than one keyword, the dependency between the keywords in RQ over the XML data being queried should also be captured.

Guideline 5: Keywords' co-occurrence. A refined query candidate RQ is effective for a certain search for node T, if RQ has as many keywords as possible that co-occur frequently in the subtrees of type T. $\square$

Since the desired search for node candidate T may not be unique, for convenience we first discuss the case that T is unique. In order to quantify the dependency of keywords in a refined query RQ, we utilize a variant of *association rule* [9]. For each keyword $k_i \in RQ$, we measure how often another keyword $k \in RQ$ appears in the subtrees of type T that contain k_i, as shown in Formula 5.7:

$$C(k_i \Rightarrow k) = f^T_{k,k_i} / f^T_{k_i} \tag{5.7}$$

where $f^T_{k_i}$ represents the number of subtrees (rooted at node type T) that contain keyword k_i, and f^T_{k,k_i} denotes the number of subtrees (rooted at node type T) that contain both k_i and k. Note that f^T_{k,k_i} is computed offline at the expense of large index stored.

The dependency score of a RQ is shown in Formula 5.8. The inner sum is a cumulation of how often each other keyword $k_i \in RQ$ appear together with k, while the outer sum cumulates such score for each keyword in RQ. In addition, as Guideline 5 usually favors the RQ with large size, a normalization factor $|RQ|$ is introduced to prevent such bias.

$$Dep(RQ, Q|T) = \sum_{k \in RQ} \frac{\sum\limits_{k_i \in RQ, k_i != k} C(k_i \Rightarrow k)}{|RQ|} \tag{5.8}$$

Once again, when Q is inferred to have multiple desired search for node candidates, we adopt Guideline 3 again to get the overall dependency score $Dep(RQ, Q)$ as below.

$$Dep(RQ, Q) = \sum_{T \in T_{for}} (C_{for}(T, Q) * Dep(RQ, Q|T)) \tag{5.9}$$

At last, the overall rank of a refined query RQ (w.r.t the initial query Q) is completed by a weighted sum of its similarity score and dependency score in Formula 5.10.

$$Rank(RQ, Q) = \alpha * \rho(RQ, Q) + \beta * Dep(RQ, Q) \tag{5.10}$$

where α and β are tunable weights that reflect the importance of each metric. α=β=1 is the default choice, and the effectiveness impact of different choices will be evaluated in section 5.7.4.

All the above statistic data can be collected during the pre-processing of XML document, and we refer readers to section 5.6 for the index construction on collecting all the above statistic data for efficient ranking computation.

5.4 Exploring the Refined Query

Given a query Q and a set R of refinement rules at term level, a static query refinement is to infer all potential RQ candidates and sort them by their respective dissimilarities. However, it is not guaranteed that which RQ candidates have meaningful matching results over the XML data D, until they are tried on D. Furthermore, it is quite expensive to infer all such RQ candidates, as the amount of such RQs may be quite large, especially for those derived by applying term deletion. Lastly, it is also expensive to conduct multiple-times scan of the keyword inverted lists in order to find those RQ candidates that have results over D.

Since the RQs that have both minimum $dSim(Q, RQ)$ and meaningful matching result over XML data D is unknown ahead of processing Q, we should adaptively explore those RQs during the processing of Q. As can be seen in any refinement algorithm in section 5.5 later, a fundamental problem encountered in between is: we can obtain a set T of keywords, each of which is from a rule set R or the initial query Q (e.g. see line 1 of Algorithm 5.2), and does exist in XML data D (see line 9 of Algorithm 5.2). However, it remains a challenge to efficiently materialize a RQ from T, such that RQ has the minimum $dSim$(Q,RQ). This is what we mean the *exploration of optimal RQ*, as defined below:

Problem Formulation: Given a keyword sequence S={k_1,k_2,...,k_s} (S denotes the initial query Q), a set T={k'_1,k'_2,...,k'_t} of keywords and a set R of refinement rules. We aim to find a RQ, which is a subset of T and $\forall RQ' \subseteq T$, $dSim$(Q,RQ)$\leq dSim$(Q,RQ') (by Definition 5.5).

We develop a bottom-up dynamic programming method, namely *getOptimalRQ(S,T)*, to resolve this problem.

Sub-problems: We create subproblems as below. Let 0<$i \leq s$ be an integer. Let $S[1, i]$= {k_1,k_2,...,k_i} be a sub-sequence of S. Let C be an array of length (|S|+1), where $C[i]$ is the minimum dissimilarity between $S[1, i]$ and some $RQ \subseteq T$. Our final goal is to compute a value for $C[|S|]$, which is the minimum dissimilarity between S and some $RQ \subseteq T$.

Notations: Each $k_i \in S$ is associated with a set of refinement rules, denoted as $R(k_i)$, $R(k_i)$={r | r=<*$k_i \rightarrow k'_m$,...,k'_n >}, where $*k_i$={k_j, k_{j+1},...,k_{i-1}, k_i} is a sub-sequence of S ended with k_i and {k'_m,...,k'_n} $\subseteq T$. For each rule r, its left and right hand side are denoted as $LHS(r)$ and $RHS(r)$ respectively.

Initialization: $C[0]$ = 0, which means the dissimilarity between an empty query and any other query is 0.

Recurrence Function: Regarding the subproblem of computing $C[i]$ for 0<i≤$|S|$, we have three options to consider.

Option 1: when the ith keyword $k_i \in S$ also appears in T, then the dissimilarity score remains unchanged.

Option 2: when k_i does not appear in T, and term deletion is applied to delete k_i.

Option 3: for a refinement rule r, if $LHS(r) = *k$ and $RHS(r) \subseteq T$, C[i] should be equal to a sum of $C[i - |LHS(r)|]$ and ds_r. If more than one rule can be applied here, the one with the minimum sum is selected. This case is used to handle term merging, split and/or substitution.

Among these three options, the one with the minimum value is assigned to $C[i]$, as summarized in Formula 5.11.

$$C[i] = min \begin{cases} C[i-1] & \text{if } k_i \in T \\ C[i-1] + cost\ of\ deleting\ k_i & \text{if } k_i \notin T \\ C[i - |LHS(r)|] + min\{ds_r\} & \\ & \text{if } k_i \notin T \text{ AND } LHS(r) = *k_i \text{ AND} \\ & RHS(r) \subseteq T, \text{for each } r \in R(k_i) \end{cases} \tag{5.11}$$

As we can see, *getOptimalRQ* is insensitive to the order of keywords in T, but sensitive to the order of keywords in S, because S denotes the original query which is a sequence of keywords (as defined in section 5.2). However, this property does not limit its applicability and correctness w.r.t the four refinement operations defined. For instance, if a user mistakenly splits a term to two terms in his query S, then these two neighboring terms must also appear next to each other in the same order in the LHS of a term merging rule.

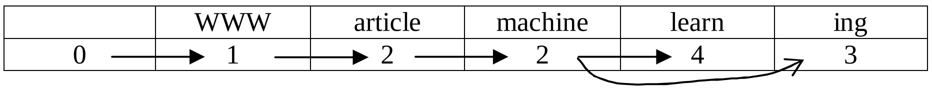
A formal presentation of the above core idea of *getOptimalRQ* is shown in Function

```
Function getOptimalRQ($S[1...s]$,$T[1...t]$)
Let C be an array of length s+1, C[0]=0;
Let RQ be an empty set;
Hashtable R = readInRules();
for (i = 1 to s) do
    /* Option 1                                                  */
    if (S[i]$\in$T) then C[i] = C[i-1];
    RQ.add(S[i]);
    else if (S[i]$\notin$T) then
        /* Option 2                                              */
        C[i] = C[i-1] + cost of deleting S[i];
        /* Option 3                                              */
        foreach (*S[i]=S[j,...,i] for j$\in$[i,1]) do
            if ($\exists$r$\in$R, s.t. *S[i]=LHS(r) $\cap$ RHS(r)$\subseteq$T) then
                temp = C[i-$|LHS(r)|$]+cost of applying rule r;
                if (temp $<$ C[i]) then
                    C[i]=temp;
        RQ.add(RHS of the r with minimum dissimilarity);
return C[s] and its associated RQ;
```

6, which runs exactly as what we have described above.

Time Complexity: *getOptimalRQ* runs in $|Q|$ loops, where in the worst case, each subsequence of Q is related to a certain refinement rule r. In addition, suppose a B-tree index is built upon the refinement rule set R, so the cost of locating such r is O($log|R|$). As a result, the worst case time complexity of *getOptimalRQ* is: O($\sum_{i=1}^{|Q|}\sum_{j=1}^{i} log|R|$) = O(($\frac{|Q|(|Q|+1)(2|Q|+1)}{12} + \frac{|Q|(|Q|+1)}{4}$)$log|R|$) = O($|Q|^3 log|R|$).

A running example of *getOptimalRQ* is shown as below.



	WWW	article	machine	learn	ing
0 →	1 →	2 →	2 →	4	3

Figure 5.2: A running example of finding the optimal RQ

Example 5.3. *Given a query Q=\{WWW,article,machine,learn,ing\} and a keyword set T=\{machine,inproceedings, learning,worldwide, Web,World,Wide\}, three relevant rule instances r3, r4 and r6 in Table 5.2 are identified. Figure 5.2 shows how array C is filled* *during the process of* getOptimalRQ(Q,T). *To compute C[1], option 2 offers a cost of C[0]+cost of deleting "WWW"=0+2=2, while option 3 offers a cost of C[0]+*ds_{r6}*=1. So C[1]=min(2,1)=1. Similarly, C[2]=C[1]+1=2, C[3]=2 as "machine" exists in* T*; C[4]=2+2=4, and C[5]=min(C[3]+1, C[4]+2)=3. Finally, the optimal* RQ*=* $\{$*World, Wide, Web, inproceedings, learning*$\}$*, and* $dSim(RQ, Q)$*=5.*□

Summary *getOptimalRQ* serves two purposes. *First*, it generates the optimal RQ in term of $dSim(Q,RQ)$. *Second*, as a side product, a ranked list of some (but not all) non-optimal RQ candidates by $dSim(Q, RQ)$ can also be obtained, as they are indeed the intermediate results kept during executing *getOptimalRQ*. They will be used as the candidates for Top-K RQs later in section 5.5.

5.5 Content-aware Query Refinement

The main challenge towards an effective query refinement is, it is unknown whether any refinement is needed ahead of processing the initial query, as each RQ must have meaningful SLCA results over the XML data by Definition 5.1. A straightforward solution is to try the initial query first, and if no matching result is found, we go to infer all potential RQ candidates based on the given refinement rule set, and try them one by one until the desired RQ is found. However, it may involve the evaluation of multiple queries, which has to scan the corresponding keyword inverted lists multiple times; even worse, many top-ranked RQs may not have any matching result.

Therefore, we propose to integrate the job of looking for the refined queries of Q and generating their matching results together to guarantee the existence of meaningful SLCA result for each RQ found; and meanwhile accompany the refinement job with the job of processing the initial query Q, in order to scan the related keyword inverted lists as few times as possible (optimally only once). This is what we call *content-aware XML* *keyword query refinement.*

As a result, two separate solutions, namely partition-based approach and short-list eager approach, are designed to find the approximate Top-K RQs and their matching results in a flow of document order. The main procedure is: we first maintain a ranked list to store the approximate Top-$2K$ RQ candidates in term of $dSim(Q, RQ)$ during answering Q. In the end, we apply the complete query ranking model (proposed in section 5.3) to generate the final Top-K RQs from the $2K$ candidates.

5.5.1 Partition-based Algorithm

As evident by Definition 5.1, the root node of an XML data tree is a typical meaningless SLCA, because users are only interested in the fragments of XML data. Therefore, we can partition the XML data tree into a list of *ordered* partitions as defined below:

Definition 5.6. ***(Document Partition)*** *Given an XML data tree D, a subtree D_i is a document partition of D if the root node R_{D_i} of D_i is the ith child of D's root node.*

Document partition is a logical partition in that, it is a virtual view of the XML data tree D by ignoring its root node without modifying the structure and order of the nodes in D. In this way, our algorithm proceeds from one partition to another partition in document order, thus avoids all SLCA computations leading to the meaningless root node of D. In Figure 5.1, there are 2 document partitions of D: D_1 rooted at author:0.0 and D_2 rooted at author:0.1, and all the meaningful SLCA nodes are either the self or descendants of the root node of D_1 or D_2.

As most users concern on the Top-K RQs, we aim to support Top-K query refinement. The main procedure is as follow: we first maintain a ranked list to store the approximate Top-$2K$ RQ candidates in term of $dSim(Q, RQ)$ during the processing of Q. In the end, we apply the complete query ranking model (proposed in section 5.3) to generate the final Top-K RQs from the $2K$ candidates.

Before we introduce the main algorithm, we would like to describe some important data structures used as a preliminary step. *RQSortedList* is developed to store the up-to-date Top-2K RQs during the procedure of query refinement. It is implemented as a sorted list with a B-tree index built on the dissimilarity of RQ, where method *insert* and *remove* can be done in O(log2K) time. Besides, method *hasRQ*, which is used to check whether a RQ to be inserted is already in the list, can be done in O(1) time by maintaining a separate hashtable whose key is RQ itself.

Algorithm 5.2: Partition-based Top-K query refinement

```
   input  : Q={$k_1$,...,$k_n$}, refinement rule set $R$, XML document $D$, $K$
   output : $result$={($R_{Q_1}$,SLCA($R_{Q_1}$)),...,($RQ_K$,SLCA($RQ_K$))}
1  Let result ← ∅;  Let $KS$ = getNewKeywords(Q) + Q
2  Let $RQSortedList$ = a list of $RQ$s sorted by $dSim(Q, RQ)$
3  {$S_1$,$S_2$,...,$S_m$} ← getInvertedLists($KS$)
4  while (!end($S_i$) for each i∈[1,m]) do
5      $v_s$ = getSmallestNode();                              /* 1≤s≤m      */
6      $D_{pid}$ = getDocPartition($v_s$)
7      {$S'_1$, $S'_2$,..., $S'_m$} ← getKLPartition(pid)
8      move cursor of $S_i$ to the node next to the end of each $S'_i$
9      Let T = {$k_i$ | $S_i$' is not empty}
10     {<$RQ_i$,$dSim(Q, RQ_i)$>|i∈[1,2$K$]}=getOptimalRQ(Q,T,2$K$)
11     foreach $RQ_i$ do
12         if ($dSim(Q, RQ_i) < RQSortedList.max$) then
13             if (!$RQSortedList$.hasRQ($RQ_i$)) then
14                 $RQSortedList$.remove(2$K$)
15                 $RQSortedList$.insert(<$RQ_i$,$dSim(Q, RQ_i)$>)
16             $slca_{RQ_i}$ = computeSLCAs({$S'_1$, $S'_2$,..., $S'_m$},$RQ_i$)
17         $result$.add($RQ_i$,$slca_{RQ_i}$)
18     reset $S'_1$, $S'_2$,..., $S'_m$ to empty
19 Apply Formula 5.10 on result to get final Top-$K$ $RQ$s
```

Algorithm 5.2 presents the details of partition-based approach. The input is an initial user query Q, a value of K, an XML document D and a given refinement rule set R. The output is a list of Top-K refined queries and their corresponding matching results over the XML data. Initially, it finds a set KS of keywords that appear in either R or Q via a consultation on a given pertinent refinement rule set R (line 1). A ranked list called $RQSortedList$ is developed to store the up-to-date Top-2K RQs during the procedure of query refinement (line 2). It supports three major operations: *insert* a RQ into the list, *remove* the lowest-ranked RQ from the list, and *hasRQ* checking whether a RQ to be inserted is already in the list.

A cursor is maintained for each keyword inverted list S_i. The algorithm runs in an iterative way: as long as the end of all the related keyword lists haven't been reached, the smallest node v_s in document order is selected (line 5), and the document partition that contains v_s is located by Definition 5.6, denoted as D_{pid}, where pid is the label of this partition's root node (line 6). Function *getKLPartition* is responsible for identifying the corresponding sublist S_i' of each keyword list S_i within partition D_{pid}, based on the property that pid is the prefix of the dewey label of each node in each S_i' (line 7). Accordingly, the cursor of each S_i is moved to the node next to the end of S_i' (line 8).

An extension of Function *getOptimalRQ(Q,KS,2K)* (proposed in section 5.4) is invoked to find the Top-$2K$ RQ candidates (if they do exist) within partition D_{pid} (line 9-10); this extension is easy to achieve, as those RQ candidates are in fact preserved as the intermediate results during the exploration of optimal RQ. For each RQ_i in the Top-$2K$ RQs found, if the dissimilarity of RQ_i is smaller than that of the lowest-ranked query in $RQSortedList$ and RQ_i has not been inserted before, then RQ_i is inserted into $RQSortedList$ (line 13-15), and any existing SLCA computation method (such as [118, 104]) can be employed to find the SLCAs of RQ_i within partition D_{pid} (line 16) and add them into *result* (line 17). Lastly, the overall query ranking model (i.e. Formula 5.10) is applied on the $2K$ RQ candidates to get the final Top-K RQs (line 19).

A running example of Algorithm 5.2 is shown in Example 5.4.

Example 5.4. *Consider a query Q=$\{$article,online,data,base$\}$ issued on the XML data in Figure 5.1, and the Top-1 RQ is expected if Q needs to be refined. Rules r2, r3 and r7 in Table 5.2 are found to be relevant to Q. For illustrative purpose, we list only five typical RQ candidates in an ascending order of its dissimilarity w.r.t Q.*

R_{Q_1}*: $\{$article,online,database$\}$ (2 merges)*

R_{Q_2}: *{article,on,line,database} (1 merge)*

R_{Q_3}: *{inproceedings,online,database} (1 merge, 1 substitution).*

R_{Q_4}: *{inproceedings,on,line,data,base} (1 split, 1 substitution).*

R_{Q_5}: *{inproceedings,online,base} (1 deletion, 1 substitution).*

......

There are two document partitions D_1 and D_2 in Figure 5.1. Partition D_1 in Figure 5.1 is identified to contain part of the related keywords, and the partitioned keyword lists are: S'_{online}= $S'_{database}$={0.0.1.1.0.0}, S'_{on}=S'_{data}= $S'_{article}$={}, $S'_{inproceedings}$= {0.0.1.0, 0.0.1.1, 0.0.1.2}, S'_{line}= S'_{base}={0.0.1.0.0.0}. getOptimalRQ *returns R_{Q_3} (with $dSim(Q, RQ_3)$=2) and R_{Q_5} (with $dSim(Q, RQ_5)$=3) as Top-2 RQs, as D_1 doesn't cover all keywords for any of Q, R_{Q_1}, R_{Q_2} and R_{Q_4}. As $RQSortedList$ is empty, both R_{Q_3} and R_{Q_5} are inserted and their SLCA results are computed.*

Then, we move to next partition D_2, where S'_{on}={0.1.1.0.0.0}, S'_{data} = {0.1.1.0.0.0, 0.1.1.1.0.0}, S'_{line}=S'_{base}=S'_{online}=$S'_{database}$ ={}, $S'_{article}$ = {0.1.1.1, 0.1.1.2} and $S'_{inproceedings}$ = {0.1.1.0}. Now, the optimal RQ found by getOptimalRQ *is RQ = {article, data} with $dSim(Q, RQ)$ = 4 (as two term deletions are applied on Q), which is even larger than the dissimilarity of the current 2nd-ranked RQ (in $RQSortedList$), i.e. 3. Therefore, we can skip computing the SLCA results for any new RQ (other than those in $RQSortedList$) found in partition D_2.*

Lastly, $result$={<R_{Q_3}, inproceedings:0.0.1.1>} is returned as the Top-1 RQ. □

In summary, Algorithm 5.2 reveals two major *advantages*: *(1)* Within each partition, it is able to decide the current Top-2K RQ candidates before computing their SLCA results. As evident in line 12-17, for a partition D_j whose associated RQ candidates have larger dissimilarity than that of the lowest ranked RQ in $RQSortedList$, we can skip computing the SLCA results of such RQ candidates on D_j (as they never can be the top-K RQ), which is an important optimization. *(2)* It follows in a flow of the document order, so for a query Q that needs no refinement, the refinement will immediately stop once the first meaningful SLCA result of Q in XML data is found, thus the extra cost spent on finding its RQs is minimized. Lastly, Lemma 5.1 and Theorem 5.1 show the exclusive features of Algorithm 5.2.

Lemma 5.1. Algorithm 5.2's query refinement is orthogonal to any existing method of computing the SLCA results of a query on a certain XML document.

Proof Sketch. Algorithm 5.2 proceeds from one partition to another in document order, where in each partition P, the RQ candidates are determined before finding their SLCA results within P, as evident in line 10-15. Thus, it is orthogonal to the concrete methods of computing the SLCA results of these RQ candidates. $\square$

Note that, without loss of generality, Algorithm 5.2 is orthogonal to any LCA computation methods, where the only modification is to relax the criteria of triggering a query refinement as defined in property 1 of Definition 5.1.

Theorem 5.1. *Given a query Q issued on an XML document D, Algorithm 5.2 is able to return the Top-K RQs according to their dissimilarity $dSim(Q, RQ)$, and meanwhile generate their matching results within a one-time scan of related keyword inverted lists.*

Proof Sketch. In Algorithm 5.2, line 4 guarantees a one-time scan of the related keyword inverted lists. Besides, for each partition visited, function *getOptimalRQ* is able to find the top-K RQs, the $RQSortedList$ can guarantee to store the up-to-date top ranked RQs. Lastly, by Lemma 5.1 the correctness and completeness of the matching results of each RQ are guaranteed. $\square$

Time Complexity: If indexed lookup in [118] is adopted for SLCA computation, Algorithm 5.2 costs O(F*$KlogK$*$|S'_1|md\log|S'|$), where $S'(S'_1)$ is the max (min) size throughout lists S'_1 to S'_m; F is the fanout of document root node; m is number of keywords involved and d is the document depth. The total cost by *getOptimalRQ* is O(F*m^3). Thus, the total cost is O(F*($KlogK$*$|S'_1|mdlog|S'|$+m^3)).

5.5.2 Short-List Eager Algorithm

As we can see, Algorithm 5.2 requires a full scan of the related keyword inverted lists, though it needs only one-time scan. In practice, however, the frequencies of query keywords typically vary significantly [118]. Therefore, during the exploration of Top-K RQs, if we can start from the RQ candidates that contain the keyword of the shortest inverted list first, it is possible to skip the full scan of all the other inverted lists involved, as shown in Example 5.5.

Example 5.5. *Consider the Top-1 query refinement of* Q=*{XML, database,2002} issued on Figure 5.1.* $S_{database}$=*{0.0.1.1.0.0},* S_{XML} =*{0.0.1.0.0.0, 0.1.1.0.0.0, 0.1.1.2.0.0},* S_{2002}=*{0.0.1.0.1.0, 0.1.1.2.1.0}. If we start from the shortest inverted list* $S_{database}$*, partition* D_1 *with* pid*=0.0 is found to cover the first occurrence of* database. *Since* D_1 *contains all the keywords of* Q*, there is no need to find any refinement for* Q *in* D_1 *and all the subsequent partitions. Therefore, the sequential scan of* S_{XML} *and* S_{2002} *can be avoided.* □

This idea is presented in Algorithm 5.3, which runs in two main steps. In step 1, the Top-K RQs are found (line 5-19). In step 2, any existing method is employed to compute the SLCA matching results for each RQ found in step 1 (line 20-21). The input and output in Algorithm 5.3 are same as those in Algorithm 5.2. The variables that have the same name with those in Algorithm 5.2 share the same meaning.

The core part of Algorithm 5.3 is how to set the stop condition for the Top-$2K$ RQ exploration, i.e. whether the potentially minimum dissimilarity is larger than the dissimilarity of the $2K$-th query in current $RQSortedList$ when $RQSortedList$ is already

Algorithm 5.3: Short-List Eager Algorithm

```
input  : Q={$k_1$,...,$k_n$}, refinement ruleset $R$, XML document $D$, $K$
output: result={($R_{Q_1}$, SLCA($R_{Q_1}$)),..., ($RQ_K$,SLCA($RQ_K$))}
1  Let $RQSortedList$ be a list of $RQ$s sorted by dissimilarity;
2  Let $KS$ = getNewKeywords(Q) + Q;  Let $C_{potential}$=0;
3  {$S_1$,$S_2$,...,$S_m$} ← getInvertedLists(allKeywords);
4  Let $KS_{pid}$ denote a set of keywords appearing in partition $pid$;
5  while ($C_{potential} \leq RQSortedList.max$) do
6      $k_i$ = the keyword in $KS$ with the shortest inverted list $S_i$;
7      foreach partition $D_{pid}$ in $S_i$ do
8          foreach keyword $k \in KS$ other than $k_i$ do
9              if ($k$ appears in Partition $D_{pid}$) then
10                 insert $k$ into $KS_{pid}$;
11         {<$RQ_i$,$dSim(Q, RQ_i)$>|i∈[1,2$K$]} = getOptimalRQ(Q,$KS_{pid}$,2$K$);
12         foreach $RQ_i$ do
13             if ($dSim(Q, RQ_i) < RQSortedList.max$) then
14                 if (!$RQSortedList.hasRQ(RQ_i)$) then
15                     $RQSortedList$.remove(2$K$);
16                     $RQSortedList$.insert(<$RQ_i$,$dSim(Q,RQ_i)$>);
17     $KS$ = $KS$ - $k_i$;  remove $S_i$ from {$S_1$,...,$S_m$};
18     Compute $C_{potential}$ = getOptimalRQ($Q$,$KS$,2$K$);
19 Apply Formula 5.10 on $RQ$s in $RQSortedList$ to get final Top-$K$ $RQ$s;
20 foreach $RQ_i \in RQSortedList$ do
21     result.add($RQ_i$, computeSLCAs($RQ_i$));
```

full (line 5). $C_{potential}$ denotes the potentially minimum dissimilarity for those RQ candidates *unexplored* yet. If it is greater than the dissimilarity of the $2K$-th RQ in current $RQSortedList$, then any RQ candidate found later can never be one of the final Top-K RQs, and we can safely stop step 1. Otherwise, the current shortest list S_i is selected, and for each partition D_{pid} containing k_i, keyword sequence KS_{pid} will collect all keywords covered in D_{pid} by random accessing the inverted list of each other related keyword (line 8-10). Then *getOptimalRQ* is invoked to find Top-$2K$ RQs within D_{pid}, and qualified RQs are put into $RQSortedList$ (line 11-16).

A salient feature of short-list eager approach is line 17-18: at the end of each iteration, all refined queries that contain k_i have been identified, so the shortest list S_i is removed, and k_i is removed from KS accordingly; lastly, the potentially minimum dissimilarity $C_{potential}$ between Q and some RQ (which is a subset of the updated KS) is computed, which will be used in the stop condition checking of next iteration.

In order to better understand Algorithm 5.3, a running example is shown as below.

Example 5.6. *Consider a query Q_4 = {XML, John, 2003} (in Table 5.1) issued on the XML data in Figure 1, and the user expects the Top-2 refined query to be returned if Q_4 needs refinement. Initially, the inverted list for each keyword is: S_{XML} =<0.0.1.0.0.0, 0.1.1.0.0.0, 0.1.1.2.0.0>, S_{John}=<0.1.0.0>, S_{2003}=<0.0.1.1.1.0, 0.0.1.2.1.0>.*

In the first iteration, the shortest inverted list is S_{John}, which is contained in the partition with pid 0.1, denoted as $D_{0.1}$. Then we access S_{XML} and S_{2003} to find whether any occurrence of these two keywords is within $D_{0.1}$. Then $getOptimalRQ$ computes the Top-4 RQ candidates (if any), each of which should contain keyword "John". As a result, RQ_a={XML, John} (where $dSim(Q, RQ_a)$=2 as a term deletion is enforced) and RQ_b={John} (where $dSim$(Q, RQ_b)=4) are found and inserted into $RQSortedList$.

In the second iteration, keyword "2003" has the shortest inverted list and is contained in partition $D_{0.0}$, where keywords "2003" and "XML" are found to exist in partition 0.0. Thus, RQ_c={2003, XML} and RQ_d={2003} are the candidates in $D_{0.0}$ and are inserted into $RQSortedList$.

Finally, assuming the query ranking model gives equal rank to each RQ, RQ_a and RQ_c are returned as the Top-2 refined queries for Q_4, as both of them have the smallest dissimilarity. □

Time Complexity. In the worst case, each keyword in KS is involved in Top-K RQ exploration, and let m=$|KS|$. In each loop, the cost of finding all partitions covering k_j is $|S_j|$ (line 7), so let P_{k_j} denote the number of partitions containing k_j; random accesses to other keyword lists cost $\sum_{i=j+1}^{|KS|} log|S_i|$ (line 8-10) (assuming keyword lists are sorted by length ahead, i.e. $|S_j| \leq |S_i|$, $\forall j<i$); *getOptimalRQ* costs $2|Q|^3$(line 11,17); all the operations supported by $RQSortedList$ is O(1). Thus, its time complexity is $\sum_{j=1}^{m}(|S_j| + P_{k_j} * (\sum_{i=j+1}^{m}(|Q|^3 + log|S_i|) + T_{slca}))$, where T_{slca} denotes SLCA computation time for Top-K RQs, depending on the concrete algorithm adopted.

Discussion. *First*, Lemma 5.1 and Theorem 5.1 also hold for Algorithm 5.3. *Second*, the performance of Algorithm 5.3 depends on two factors: (1) whether the RQs that cover the keyword with the shortest inverted list are among the final Top-K RQs.
(2) how early the first match of each RQ in the final Top-K RQs appears in XML data.

Based on this analysis, we can have a smarter choice of k_i and S_i in each iteration (line 6): the k_i which either appears in the RHS of the refinement rules related to Q or never appears in the LHS of any rule related to Q (i.e. the keyword that does not need any refinement), and also has the shortest inverted list should be chosen first. In this way, the RQ containing such k_i should have a high probability to be one of the final Top-K RQs, and thus the exploration of Top-K RQs can finish earlier.

As a summary, Algorithm 5.2 achieves a one-time scan of the related keyword lists at the expense of a full scan for each related keyword list; while Algorithm 5.3 avoids the full scan at the expense of scanning the related keyword lists multiple times. The practical performance of these two approaches are query and data dependent.

5.5.3 Summary

We can find, the performance of the partition-based approach and short-list eager approach depends on two important factors:

1. How many times the related keyword inverted lists are traversed.

2. Whether a full scan of the related inverted lists is needed.

There is a tradeoff between the above two factors; and these two algorithms are designed to focus on one factor at the expense of the other factor. Therefore, there is no such algorithm that can beat the other one all the time; the performance of both algorithms are query and data dependent, as illustrated in section 5.7 later.

5.6 Index Construction

In this section, we describe the indices built for an efficient content-aware query refinement framework.

The first index built is the traditional keyword inverted list. For each keyword k, it stores a list of nodes that directly contain k in document order. Such order is in accordance with the query processing order in both Algorithm 5.2 and 5.3. A B-tree index (on the keyword) is built over all the inverted lists to accelerate the lookup.

As another core part of our query refinement framework, how to efficiently calculate the ranking score of a RQ is also important. Among all the statistics needed in the whole ranking model, F_T (in Formula 5.2) and N_T (in Formula 5.3) can be easily collected when parsing the input XML document, and stored in two separate tables, where each entry of the table corresponds to a unique node type T.

Furthermore, some statistics such as the XML term frequency $tf(k,T)$ (in Formula 5.2) and document frequency f_k^T (in Formula 5.3) involve both the node type and the keyword. Therefore, we build another index called *keyword stats table*, which stores both $tf(k,T)$ and f_k^T for each combination of keyword k and node type T in the XML document D. In the worst case, each node type may directly contain all the distinct keywords in D; so the space is O($m * t$), where m denotes the number of distinct keywords in D and t denotes the number of node types in D (i.e. the number of distinct prefix paths of the XML data tree). Similar to the above keyword inverted list, a B-tree index is built on the keyword stats table for an efficient support on the following two operations:

- $getTF(T,k)$, which returns the XML TF $tf(k,T)$.
- $getDF(T,k)$, which returns the XML DF f_k^T.

Lastly, a *keyword dependency table* is built to store the statistics f_{k,k_i}^T (in Formula 5.7). For each combination of any two distinct keywords k and k_i and any node type T, we have an entry storing f_{k,k_i}^T.

5.7 Experiments

In the experimental study, we investigate the efficiency and scalability of the two refinement algorithms (i.e. the partition-based approach and short-list eager approach) proposed in section 5.5, and the effectiveness of our query ranking model proposed in section 5.3. Note that, we do not include the evaluation of the most related work [96] which is designed for keyword query cleaning in relational database, because it is nearly infeasible to extend it to fit into XML document, and it cannot guarantee the existence of the matching result of the cleaned keyword query.

Equipment. All experiments are performed on a 1.9 GHz AMD DualCore PC running Windows XP with 3GB memory. All codes are implemented in Java, and Berkeley DB Java Edition [1] is used to store the keyword inverted lists.

Notations.

(1) SLCA refers to the scan-eager approach proposed in [118] for SLCA computation.

(2) The short-list eager and partition-based algorithm proposed in section 5.5 are called SLE and Partition respectively. Both Partition and SLE employ the scan-eager approach [118] in computing the SLCA results of a query.

Data set and Query Set. Since our work is an empirical study closely related to user's real search experience, we use real data set and real-world user queries instead of the synthetic data sets and queries. To our best knowledge, a common problem that all existing works in the field of XML keyword search have encountered in studying the practicability of their approaches is the lack of real-world data sets and user queries.

Due to the lack of real-world data sets, only two real data sets DBLP [70] (420MB, depth =2, up to 2007/12/10) and Baseball[6] (1MB, depth=5) are used in our experiments. DBLP contains publications in computer science; Baseball contains information on teams and players of North American baseball league. These two real data sets differ

[6]*http://www.ibiblio.org/xml/books/biblegold/examples/baseball/*

from each other in terms of the data-organization and data-application: DBLP is shallow and wide, while Baseball is deep and narrow. Our goal in choosing these diverse data sources is to understand the usefulness of our refinement strategies in different real-world environments.

In order to minimize the subjectivity in experimental evaluation, the most recent 1000 real-world user queries are selected from the query log of an DBLP online demo[7] of our previous work [15], out of which 219 queries (with an average length of 3.92 keywords) that have empty result are selected to form a pool of queries that need to be refined, which coincides with the primary motivation of this work. Besides, we randomly pick 100 queries that have meaningful matching results and add them into the query pool, in order to increase the variety of queries. The refinement rules come from either WordNet [44] or human annotation, and we adopt the same metrics for measuring the dissimilarity score of each rule (except for term deletion) as described in section 5.2.2.

Same as IR query refinement approaches such as [50], we build the refinement rule set at term-level for the four refinement operations adopted in this work by asking two human annotators to manually refine the above 219 queries. However, the refinement is not immediate, as there can be many possible refined queries resulted from the application of different rules. Regarding the dissimilarity score ds_r of a refinement rule r, we adopt the same metrics as described in section 5.2.2, and ds_r = 2 is assigned for a single term deletion.

5.7.1 Sample Query Set

Sample queries with a refinement using a typical operation are shown in Table 5.3, 5.4, 5.5 and 5.6 respectively, where in each table the 3rd column shows the refinements returned by our method, and the 4th column shows the cardinality of results based on the

[7] *http://xmldb.ddns.comp.nus.edu.sg*

Table 5.3: Sample Query Sets for Term Deletion

ID	Initial Query	Suggested Refinements	Results
Q_{D1}	Ling,Tok,Wang,twig,pattern,join	delete "pattern" or "join"	2 or 5
Q_{D2}	Yufei,Tao,skyline,2000	delete "2000"	5
Q_{D3}	Tan,Kian,Lee,keyword,search	delete "keyword"	8
Q_{D4}	XML,view,model,1995	delete "XML" or "1995"	4 or 8
Q_{D5}	XML,graph,keyword,search	delete "XML" or "graph"	1 or 22
Q_{D6}	Ooi,Beng,Chin,Jagadish,index	delete "Jagadish" or "index"	8 or 11
Q_{D7}	Yannis,graph,keyword,search	delete "Yannis" or "graph" or "keyword"	1 or 10 or 1

Table 5.4: Sample Query Sets for Term Merging

ID	Initial Query	Suggested Refinements	Results
Q_{M1}	Jia,wei,han,2006	Jiawei	35
Q_{M2}	Xiao,fang,zhou,2005	Xiaofang	16
Q_{M3}	on,line,news,paper	online,newspaper	6
Q_{M4}	electronic,text,book	textbook	6
Q_{M5}	xml,key,word,search	keyword	21
Q_{M6}	online,hand,writing	handwriting	47
Q_{M7}	work,shop,data,management,korea	workshop	2
Q_{M8}	net,work,routing,protocol	network	59
Q_{M9}	micro,array,gene,classification,selection	microarray	21
Q_{M10}	over,lay,routing,cost	overlay	3

corresponding RQ. Besides, queries involving multiple mixed refinements, i.e. Q_{X1}-Q_{X6}, are shown as below.

Q_{X1}:{eficient, key, word, search}, which can be refined by substituting "efficient" for "eficient", followed by a merging of "key" and "word".

Q_{X2}:{eficient, sky, line, computation}, where a desired refinement is {efficient, skyline, computation}.

Q_{X3}:{worldwide, web, search, engine} can be refined by either adopting worldwide→world,wide or www→ worldwide web.

Q_{X4}:{inproceeding, xml, twig, match} can be refined by substituting "inproceedings" for "inproceeding", "matching" for "match".

Q_{X5}:{suficient, bundary, values} can be refined by a series of substitutions: suficient→

Table 5.5: Sample Query Sets for Term Split

ID	Initial Query	Suggested Refinements	Results
Q_{P1}	adhoc,search	ad,hoc	14
Q_{P2}	webpage,filtering,2006	web,page	2
Q_{P3}	fulltext,search,networks	full,text	3
Q_{P4}	floatingpoint,function	floating,point,function	10
Q_{P5}	multiquery,processing	multi,query	24
Q_{P6}	realtime,application,analysis	real,time	11
Q_{P7}	hengtao,shen,video,2007	heng,tao	5

Table 5.6: Sample Query Sets for Term Substitution

ID	Initial Query	Suggested Refinements	Results
Q_{S1}	Jagadish,VLBD	VLDB	41
Q_{S2}	machin,learning,technique	machine	9
Q_{S3}	Jim,Gary,VLDB	Gray	8
Q_{S4}	principle,component,neural,network	principal	18
Q_{S5}	xml,document,object,model	DOM	11
Q_{S6}	extensible,markup,language,application	XML	71
Q_{S7}	privacy,preserving,cluster	clustering	24
Q_{S8}	fuzy,database,search	fuzzy	4
Q_{S9}	DASFA,2007,XML	DASFAA	11
Q_{S10}	distributed,allocation,chanel	channel	42
Q_{S11}	search,bundary,constraints	boundary	2

sufficient, bundary→boundary, values→value.

Q_{X6}:{private, data, preserve} can be refined by a series of substitutions: private→privacy, preserve→preservation.

5.7.2 Efficiency

In this section, we evaluate the efficiency of SLE and Partition by measuring the latency between a query is issued and its Top-K RQs with their matching SLCA results are returned.

Efficiency on sample queries We first evaluate SLE and Partition for Top-1 query refinement on all *sample queries* in Table 5.3 to 5.6 plus Q_{X1}-Q_{X6}. We also compare them with a naive approach, where we first process the initial query and then enumerate all the RQ candidates if necessary, and try them one by one (in a descending order of its query rank) until a user is satisfied with the query results; while the time spent on user judgement is not counted. Besides, we record the time spent on processing the initial query by SLCA [118] to understand the extra cost brought by the exploration of RQs. Note that directly processing the initial query without refinement may return either empty or meaningless results.

Figure 5.5(a)-5.5(e) show the elapsed time for all sample queries that need refinement (on hot cache), where we have four observations.

(1) Both Partition and SLE outperform the naive approach for all sample queries; Partition is about twice faster than SLE.

(2) SLCA spends the least evaluation time, as it is only responsible for processing the initial query which even has no meaningful matching result. In contrast, Partition brings a very small extra cost (about 30% in average), but serves both the purpose of producing the top-1 RQ and finding its matching result in XML data tree.

(3) SLE outperforms Partition for Q_{D2} and Q_{X3}, because the keyword with the shortest inverted list is also in the final Top-1 RQ, so that the full scan of corresponding inverted lists is avoided.

(4) Interestingly, we find for Q_{M10}, Q_{S3}, Q_{S11} and Q_{D7}, Partition is even more efficient than SLCA which does not perform any refinement operation. This can be explained that the extra cost spent by Partition on computing the ranking scores of the RQ candidates is even smaller than the cost by SLCA in computing the meaningless LCAs (i.e. the document root node) for those queries.

Lastly, we randomly pick 10 queries that do not need refinement and test the elapsed time by SLE, Partition and SLCA. As shown in Figure 5.5(f), in average both SLE and Partition spend about 20% extra time as compared to SLCA, which is acceptable.

5.7.3 Scalability

In order to test the scalability of SLE and Partition in Top-K query refinement, we design two experiments.

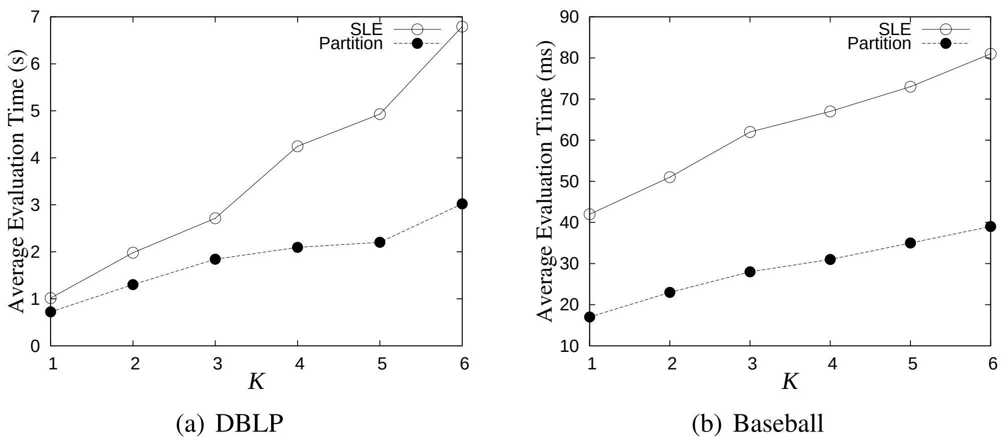


Figure 5.3: Effects of K on Top-K Query Refinement

Firstly, we measure the effects of different choices of K on the evaluation time of Top-K query refinement, where K∈[1,6]. A batch of 40 random queries with an average length of 3.71 for DBLP and 20 random queries with an average length of 3.18 for Baseball are tried, and the average time of those queries in five executions are shown in Figure 5.3. As evident from Figure 5.3(a), Partition scales well all the way, while SLE's time increases much faster when K>3. Since SLE has to find all Top-K RQs before evaluating them, the larger the K is, the more extra time on dissimilarity computation is, and more times of keyword lists scan are needed in employing existing methods to find SLCA results. In contrast, for Partition approach, the larger the K is, the higher possibility that the lower-ranked RQs and their SLCA results (that are detected before the higher-ranked queries) are preserved (rather than pruned away), so less extra cost is introduced. For Baseball data, both algorithms scale equally well, as shown in Figure 5.3(b).

Secondly, we measure the response time of Top-3 query refinement by SLE and Partition over the data sets of different size, which are obtained from DBLP (420MB), and a

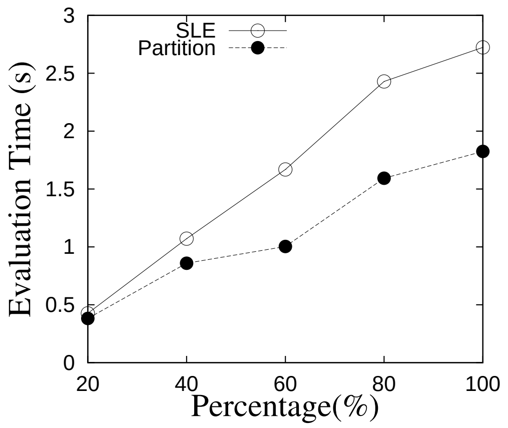


Figure 5.4: Effects of Data Size on Top-3 RQ Computation

batch of 40 random queries are used again. As shown in Figure 5.4, both approaches have a good scalability over the data size. Note that SLE has a significant increase from 60% to 80%, as SLE's efficiency relies heavily on how early the Top-K RQs are detected, which consequently affects the number of random accesses to the keyword inverted lists.

5.7.4 Effectiveness of Query Refinement

Having verified the efficiency of our algorithms, in the sequel we assess the effectiveness of our query ranking model.

Evaluation method

Traditional IR evaluation methods include precision, recall, F-measure [80], reciprocal rank [16] etc; however, all of them are based on a binary judgement (which judges a result to be either relevant or irrelevant). In contrast, by taking into account the fact that all results are not of equal relevance to users, *Cumulated Gain-based evaluation* (CG) [59] is proposed to combine the degree of relevance of the results and their ranks (affected by their possibility of relevance) in a coherent way, no matter what the recall base size is. In particular, given a ranked result list, [59] turns the list to a gained value vector G[i], which denotes the relevance score of the ith result retrieved; then a cumulated gain vector CG is defined recursively as shown in Formula 5.12, where CG[i] is computed by summing G[1] up to G[i]. Discounted CG (DCG) is designed to model user persistence to weigh down the gain from results found later in examining long ranked result lists, we refer interested readers to [59] for details. In our experiment, we adopt CG rather than DCG to evaluate the effectiveness of our query ranking model, as the ranked query list is usually not too long and all users participated in experiment are patient.

$$CG[i] = \begin{cases} G[i] & \text{if } i = 1 \\ CG[i-1] + G[i] & \text{otherwise} \end{cases} \tag{5.12}$$

Table 5.7: Top-4 ranked RQs with their result number

Q	RQ_1	RQ_2	RQ_3	RQ_4
Q_{M1}	jiawei,h,2006; 35	h,w,2006; 45	j,w,2006; 29	h,j,2006; 9
Q_{M2}	xiaofang,z,2005; 16	xiaofang,z; 91	x,z,2005; 27	f,z,2005; 7
Q_{M9}	microarray,g,c,s; 21	microarray,g,s; 60	array,g,c,s; 2	m,a,c,s; 1
Q_{S3}	J,Gray,VLDB; 8	J,Gary;21	J,VLDB; 11	G,VLDB; 4
Q_{S5}	XML,DOM; 11	d,o,m;9	XML,o,m; 5	XML,d,m;8
Q_{S6}	XML,a; 71	m,l,a;6	e,m,l; 22	l,a; 189
Q_{P3}	full,text,s,n; 3	t,s,n; 7	f,t,n; 5	f,s,n; 3
Q_{P6}	real,time,ap,an; 11	ap,an; 1187	realtime,ap;5	realtime,an; 2
Q_{X1}	efficient,keyword,s; 19	efficient,k,s; 4	word,s; 21	key,w,s; 1
Q_{X2}	efficient,skyline,c; 8	skyline,c; 13	eff,skyline; 17	efficient,l,c;4
Q_{X3}	world,wide,w,s,e;9	www,s,e;39	web,s,e;156	w,w,w,s;43

Effectiveness study

In order to study the empirical effects of our query ranking model, all queries tested are real-world user queries as logged in our XML keyword search engine [15]. For each query, we extract its Top-4 RQs. Six researchers are invited for relevance judgement of query refinement on DBLP, as DBLP is one of the few large real XML data sets, and the six researchers use DBLP to find papers frequently, which helps make their judgement more reliable. They are asked to look into each RQ and its matching results carefully, and judgements are done on a four-point scale as: (1) irrelevant, (2) marginally relevant,

(3) fairly relevant, (4) highly relevant. As mentioned in [59], a proper choice of relevance score depends on the evaluation context. Thus, we use *moderate relevance scores* (say, 0-1-2-3) for the above four-point scale, as we assume that our users are patient enough to dig down the results of low-ranked RQs.

Table 5.7 shows the Top-4 RQs and their result numbers (separated by semicolon) for some queries in Table 5.3-5.6. For simplicity, each keyword is denoted by its first letter if no ambiguity is caused. For each query in Table 5.7, all six users have an agreement that its Top-1 refined query RQ_1 is the most appropriate refinement.

Next, we make an *in-depth* analysis of the query ranking model. As the overall rank of a RQ consists of two complementary parts, i.e. *similarity score* and *dependency score*, we conduct two sets of experiments to test their respective effects individually.

Table 5.8: Query Statistics

Refinement	Num
Term Merging	18
Term Split	10
Term Substitution	31
Term Deletion	4

In the *first* experiment, we investigate the query ranking model that takes the similarity score into account alone. As Guidelines 1-4 (in section 5.3.1) contribute to the similarity score of a RQ, we test how each of them contributes to the overall quality. Let RS_0 denote the original ranking scheme, and RS_i denote a variant of RS_0 by removing `Guideline` i from consideration for $i \in [1,4]$. In our experiment, we adopt the above CG evaluation but the input now is a ranked list of RQs (associated with their matching results). 50 queries that have no meaningful result on DBLP, involve various refinement(s) and have at least 4 possible RQ candidates, are chosen from our query pool. Table 5.8 shows a summary of the number of queries that involve the four refinement operations respectively. The decay factor w in Formula 5.6 is set to 0.7 here.

Table 5.9: CG@4 by different ranking models

Variants	$CG[1]$	$CG[2]$	$CG[3]$	$CG[4]$
RS_0	2.631	3.562	4.233	4.539
RS_1	2.343	3.491	4.127	4.516
RS_2	2.416	3.525	4.161	4.525
RS_3	2.427	3.509	4.058	4.497
RS_4	2.305	3.456	4.16	4.521

Table 5.9 shows the results of the average CG values judged by the above 6 users for Top-K RQs, for K=1 to 4. *(1)* From column 2 of Table 5.9, we find the original ranking model, i.e. RS_0 is the most effective one that can capture the most relevant result as Top-1 RQ, compared to all its four variants. *(2)* By comparing CG[i] of each model for i∈[0,4], we find the original ranking model outperforms all its four variants in finding the Top-K RQs for any K∈[1,4]. *(3)* In finding the Top-1 RQ, Guideline 4 plays a much more important role than other guidelines, as CG[1] of RS_4 has the smallest value. *(4)* From the last column of Table 5.9, we find both the original ranking model and its four variants have similar value for CG[4], which means all of them are able to find the desired Top-4 RQs, although the relative ranks of these RQs vary in each variant.

Table 5.10: CG@4 by different weights

[α,β]	CG[1]	CG[2]	CG[3]	CG[4]
[1,2]	2.626	3.56	4.217	4.532
[2,1]	2.64	3.565	4.241	4.537
[1,1]	2.675	3.569	4.236	4.543
[1,0]	2.631	3.562	4.233	4.539

In the *second* experiment, we test a combined effect of the *similarity* score and the *dependency* score of a RQ. The importance of these two factors are investigated by varying the choice of the tunable parameters α and β in Formula 5.10. From the result as shown in Table 5.10, we have the following observations: (1) By comparing variants [1,1] and [1,0], we find the consideration of the dependency score does improve the overall effectiveness of our query ranking model. (2) By comparing the CG[1] for all the variants, we find the similarity score is more effective than the dependency score in contributing to infer the Top-1 RQ.

Conclusion In summary, the naive query refinement approach is not adequate due to its costly query time; the original SLCA algorithm without refinement functionality is not reliable because it fails to report meaningful answers for many queries. In contrast, SLE and Partition can detect and produce high quality refined queries and their matching results in an efficient way. Overall the best solution is the Partition algorithm, which offers the best-fit refinement and scales better than SLE. Furthermore, the comprehensive CG evaluation demonstrates the effectiveness of our query ranking model.

5.8 Summary

In this chapter, we introduced the problem of content-aware XML keyword query refinement, aiming to integrate the job of finding the desired refined queries and generating their matching results as a single problem, with no intervention on user part. We first described the criteria to trigger a query refinement and introduce the concept of dissimilarity as a preliminary quality metric of a refined query RQ. As a core part of this work, we proposed a statistics-based query ranking model which takes into account of both the keyword dependencies in RQ and the relevance of RQ w.r.t original search intention. We further proposed two adaptive query refinement algorithms. Lastly, experiments have shown the efficiency and effectiveness of our approach. In future, we would like to study another extreme - how to refine a query which has “too many” matching results over the XML data.

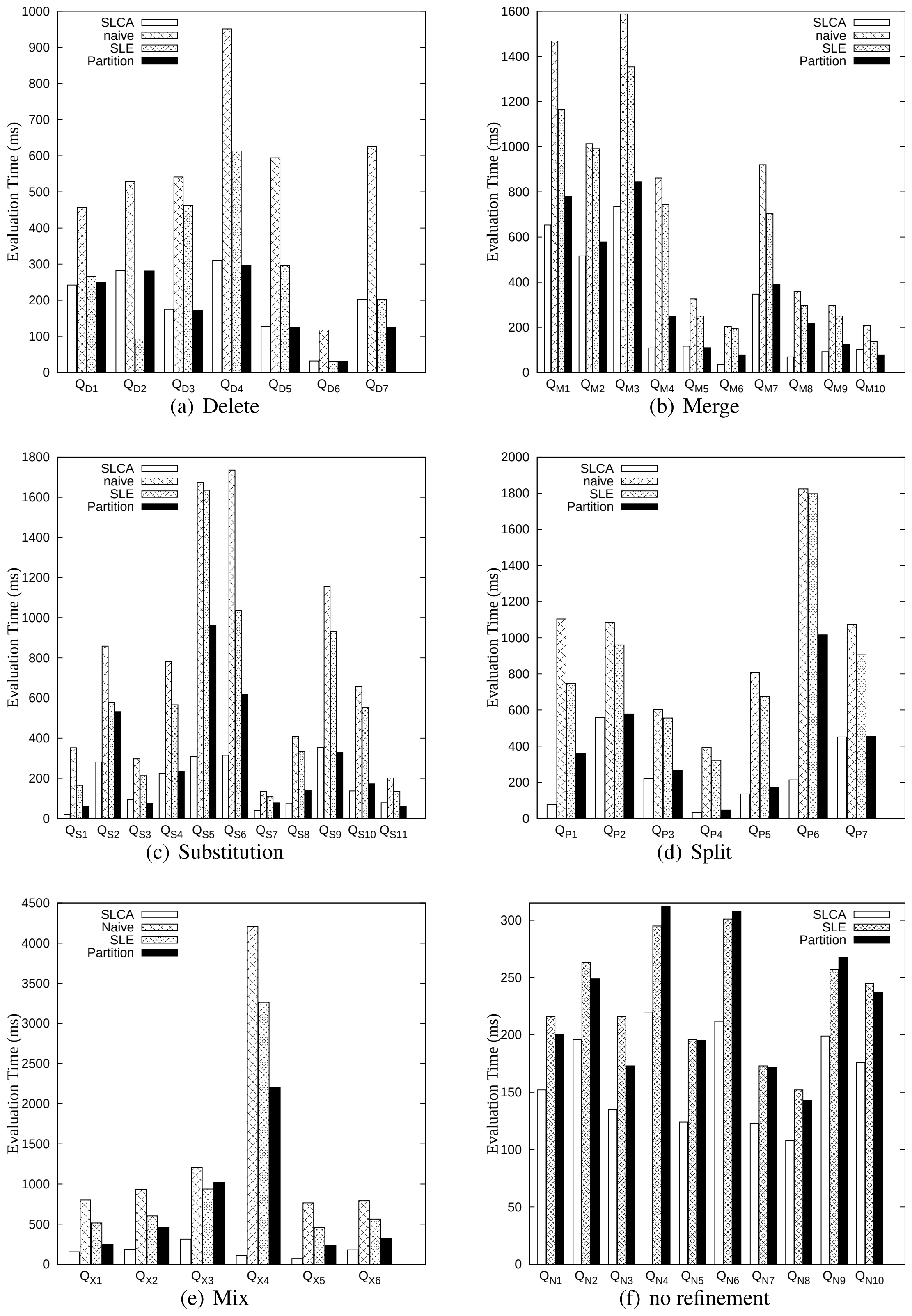


Figure 5.5: Top-1 sample query refinement on DBLP

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

Keyword search over semi-structured and structured data offers users great opportunities to explore more well-organized data. XML, as a kind of semi-structured data, has enabled data exchange over the internet. Therefore, there is rapidly growing awareness of the needs for providing effective and efficient XML keyword search method in both academic and commercial communities. However, traditional information retrieval style keyword search methods (designed for unstructured documents) are inappropriate or inadequate in the context of XML keyword search, due to its inability to take the hierarchical structure of XML data into consideration.

Building an effective XML keyword search engine has revealed various research problems, such as the design of appropriate matching semantics, the design of effective result ranking scheme such that most users can find their desired ones in top-k results returned, the design of appropriate indexing method for efficient computation of both result retrieval and result ranking, result snippet generation, keyword query refinement etc. This thesis mainly resolves the effectiveness issue by combining the contemporary DB and IR technologies, which is line with the trend of DB&IR integration [34, 12]. In particular, this thesis contributes on the identification of user search intention, the design of result ranking scheme and the keyword query refinement, as listed below.

- **Keyword query processing over tree-structured XML data** We discovered the keyword ambiguity problem in XML keyword search: a query keyword can appear as the tag name of or (part of) the value of a node in XML data, or (part of) the value of different nodes in XML data to express different meanings. In particular, we first proposed to identify the *type* of a node by its prefix path, and defined document frequency (DF) and term frequency (TF) in XML context. XML TF $f_{a,k}$ counts the frequency of a term k in leaf node a, and XML DF f_k^T counts how many nodes of type T (in XML data) containing a particular keyword k. Then in inferring the most promising search target, we have found: the desired search target should be a node type T, whose associated subtrees can cover as many distinct query keywords as possible, and should maximize other relevant information and minimize the irrelevant information. Based on this observation, we carefully designed a formula to quantify the confidence of a certain node type in XML data to be the desired search target w.r.t. a user query. Similarly, since the search constraint of a user query is usually not unique due to the occurrence of the above keyword ambiguities, we further capture the structural distance of neighboring query keywords in XML data and the proximity between keyword and the node type (as a search constraint) in measuring the confidence of potential search constraints w.r.t the query. Then we incorporated them into the design of our XML TF*IDF result ranking scheme to rank the individual matches of all possible search intentions. Lastly, we built a system prototype called XReal [18], and experimentally showed the effectiveness of our approach on XML real data set.

- **Keyword query processing over digraph-structured XML data** We observed that when XML data is modeled as a directed graph where ID references between XML elements are considered, more relevant results can be found. We also observed that whenever user issues a query, what he/she is really interested in is either a single object of interest or a list of objects of interest that interact in somehow a meaningful relationship, though users may not know such relationship explicitly. Therefore, we modeled XML document as a set of interconnected object-trees, where each object tree is a subtree representing a concept in real world, and are interconnected by either the containment edge (i.e. p-c edge) or reference edge (i.e. the edge between nodes of type ID and IDRef). Based on this model, we propose object-level matching semantics called *Interested Single Object* (ISO) and *Interested Related Object* (IRO) to capture single object and multiple objects (related via IDRef or containment edges) as user's search target respectively. An immediate benefit is that, the matching result is of finer granularity to user's search needs than previous works. Moreover, we designed efficient algorithms to find ISO and IRO matching results, and customized ranking schemes for ISO and IRO results respectively.

- **Query refinement for XML keyword search** User queries may contain irrelevant or mismatched terms, typos etc, which may easily lead to empty or meaningless results under the widely adopted conjunctive matching semantics (such as SLCA) in the context of XML keyword search. Therefore, we issued the problem of XML keyword query refinement, where the search engine should judiciously decide whether a user query Q needs to be refined during the processing of Q, and find a list of promising refined query candidates which guarantee to have meaningful matching results over the XML data, without any user interaction for all cases. To achieve this goal, we built a novel content-aware XML keyword query refinement framework which consists of two core parts: (1) we devised a statistics-based query ranking model to evaluate the quality of a refined query RQ of Q, which captures the morphological and/or semantical similarity between Q and RQ and the dependency of keywords in RQ over the XML data; (2) we integrate the exploration of RQ candidates and the generation of their matching results as a single problem, to guarantee the meaningfulness of the refined query found w.r.t. the XML data being queried. Moreover, it can be fulfilled within a one-time scan of the related keyword inverted lists optimally. Experiments on real-world data set by queries of real users have verified the efficiency and effectiveness of our refinement framework.

6.2 Future Work

How to resolve the keyword ambiguity in interpreting a user query has always been the most concerned problem in XML keyword search. While this thesis has presented several solutions, there are several directions we would like to work on this problem in the future.

- *Personalization of Search Results*. Existing XML keyword search methods focus on providing a relevance-oriented result ranking scheme to every user; however, different users may issue the same query with different search intentions, where the objective relevance score may not address the subjective individual search intention problem. In our first work as described in Chapter 3, besides automatically selecting the most promising search target for user, we may achieve a better result by offering a user interaction: i.e. before finding the final matching results, we first compute all the promising search target candidates, from which we allow user to select his/her desired search target(s), as done in [18]. Web search has tried to build user profile by looking into user's long term search history, click-through streams and data usage, in order to provide a high-quality user-oriented search to satisfy various information needs from different individuals. As XML is deployed to represent more and more information and data in internet, it demands for a similar search result personalization and proactive support for user's information need. In contrast to the unstructured document on web, keyword search on semi-structured data (such as XML) poses more challenges on analyzing user preferences, where not only the content of results, but also the structure of results should be considered. Furthermore, we plan to exploit the personalization techniques to enhance the effects of our query refinement work in Chapter 5, as it can help provide a customized suggestion w.r.t each specific user, which can alleviate the machine efforts in enumerating all possible suggestions at query-level only.

- *Improvement on Query Form.* Most of the time, the keyword ambiguity problem is attributed to the free form of keyword query itself. In contrast, the structured query language (e.g. XQuery) is expressive and leads to a unique search intention. Therefore, how to add some structured constraints on keyword query (e.g. user can roughly specify the ancestor-descendant relationship between any two keywords in the query, or specify those keywords that must appear together as part of the value by enclosing them by double quotes, etc.) according to user's own knowledge-level while alleviating user efforts in learning much syntax of structured query languages is a promising research direction. So far, several preliminary solutions in the context of XML search have been proposed: XSEarch [38] requires user to differentiate the tag name and value in his/her keyword query; DaNaLIX [78] provides users a generic natural language interface to specify their information need and translate it into XQuery expressions. However, how to improve the precision of interpretation of the search intention is still a long way to go. We believe one possible solution is to design an easy-to-use user interaction in clearing the ambiguities of query keywords.

- Result Diversification. When a user's underlying information need cannot be unambiguously determined from an initial query, an effective approach is to diversify the search results of this query, where diversification aims to find k items which are subset of all relevant results that contain both the most relevant and the most diverse results. However, increasing the diversity leads to a decrease in relevance, and it has been proven to be NP-hard [49] to find the optimal trade-off between diversity and relevance in the context of web search. It is an issue orthogonal to the result ranking scheme design, where diversification aims to display the results representing as many user search intents as possible in top-k results, while result ranking work solves the problem at individual result level and aims to display the results with as high relevance score as possible to satisfy most users' search intentions (as most users have the same intention for a particular query). It is also complementary to the issue of improvement on query form, because it improves the search quality from the perspective of internal implementation of search engine, whereas the above achieves so from the perspective of user-interface design of search engine. In particular, we find there are three future works to do: (1) How to define the dimension of diversity and features of diversity specific to XML database. (2) How to find a greedy approximation solution to strike a well balance between diversity and relevance of a result for most user information needs. (3) How to define appropriate metric to evaluate the effectiveness of the result diversification for XML keyword query.

Another independent problem we would like to investigate is how to support keyword search over probabilistic XML databases. In web 2.0 period, many data are generated either by automated information extraction which usually brings unexpected errors, or by integration from various data sources that may be uncertain to a certain degree. Since XML is able to represent data uncertainty of different degrees more naturally (by its hierarchical structure) and its semi-structured nature is tailored for the above information extraction and data integration applications, abundant uncertain data is being stored in XML format, which is called as probabilistic XML database formally [91]. In existing data models for probabilistic XML database, each node is associated with a probability assigned conditionally based on the probability of its parent node. We believe that there is a demanding need for querying on such probabilistic XML data for ordinary user in future, where keyword query will remain the most popular way to explore such uncertain data. Here, we list three challenges to be addressed. *First*, in deciding what a qualified result should be, instead of only enforcing the occurrences of all query keywords, how to incorporate the probability of each individual matching node for a matching result is a very critical problem. *Second*, it calls for an intuitive and appropriate combination of the relevance scoring function with the probability of the matching nodes in computing the ranking score of a matching result. In particular, it should adapt the traditional probabilistic data model and information retrieval model to tailor for XML context in order to have a strong theoretical guarantee for the resulted ranking scheme over uncertain XML data. *Third*, as compared to the keyword query processing over certain XML data which can skip the computation towards nodes which can not be the SLCA result [118] or contributing to the same SLCA result [104], now for each keyword, we may need to access all its related nodes and even all its ancestor nodes to compute the probability of the SLCA result under nowadays probabilistic XML data models. An efficient method that caters for both result finding and probability computation is demanded.

BIBLIOGRAPHY

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

[2] http://www.cs.washington.edu/research/xmldatasets.

[3] http://www.xml-benchmark.org/.

[4] INEX. initiative for the evaluation of xml retrieval. http://inex.is.informatik.uni-duisburg.de/.

[5] The internet movie database. http://www.imdb.com/interfaces.

[6] Information processing – text and office systems – standard generalized markup language (sgml), 1985. International Organization for Standardization.

[7] XRel: a path-based approach to storage and retrieval of xml documents using relational databases. *ACM Trans. Internet Technol.*, 1(1):110–141, 2001.

[8] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The lorel query language for semistructured data. In *International Journal on Digital Libraries 1(1)*, pages 68–88, 1997.

[9] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, 1993.

[10] 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.

[11] Shurug Al-Khalifa, H. V. Jagadish, Jignesh M. Patel, Yuqing Wu, Nick Koudas, and Divesh Srivastava. Structural joins: A primitive for efficient xml query pattern matching. In *ICDE*, pages 141–152, 2002.

[12] Sihem Amer-Yahia, Djoerd Hiemstra, Thomas Roelleke, Divesh Srivastava, and Gerhard Weikum. Db&ir integration: report on the dagstuhl seminar. *SIGIR Forum*, 42(2):84–89, 2008.

[13] Sihem Amer-Yahia, Laks V. S. Lakshmanan, and Shashank Pandit. Flexpath: flexible structure and full-text querying for xml. In *SIGMOD conference*, 2004.

[14] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[15] Zhifeng Bao, Bo Chen, Tok Wang Ling, and Jiaheng Lu. Demonstrating effective ranked XML keyword search with meaningful result display. In *DASFAA*, 2009.

[16] Zhifeng Bao, Tok Wang Ling, Bo Chen, and Jiaheng Lu. Effective XML keyword search with relevance oriented ranking. In *ICDE*, 2009.

[17] Zhifeng Bao, Tok Wang Ling, Jiaheng Lu, and Bo Chen. Semantictwig: A semantic approach to optimize XML query processing. In *DASFAA*, pages 282–298, 2008.

[18] Zhifeng Bao, Jiaheng Lu, and Tok Wang Ling. XReal: An interactive XML keyword searching. In *In Proceedings of the 19th CIKM Conference*, 2010.

[19] Zhifeng Bao, Jiaheng Lu, Tok Wang Ling, and Bo Chen. Towards an effective XML keyword search. *IEEE Trans. Knowl. Data Eng.*, 2010.

[20] Zhifeng Bao, Jiaheng Lu, Tok Wang Ling, Liang Xu, and Huayu Wu. An effective object-level XML keyword search. In *DASFAA (1)*, pages 93–109, 2010.

[21] Zhifeng Bao, Huayu Wu, Bo Chen, and Tok Wang Ling. Using semantics in XML query processing. In *ICUIMC*, pages 157–162, 2008.

[22] Doug Beeferman and Adam Berger. Agglomerative clustering of a search engine query log. In *KDD*, 2000.

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

[24] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proc. of ICDE Conference*, pages 431–440, 2002.

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

[26] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Tim Bray Textuality. Extensible markup language (xml) 1.0 (second edition), 2000.

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

[28] Nicolas Bruno, Nick Koudas, and Divesh Srivastava. Holistic twig joins: optimal xml pattern matching. In *SIGMOD Conference*, pages 310–321, 2002.

[29] Chris Buckley. Automatic query expansion using smart: Trec 3. In *TREC*, pages 69–80, 1995.

[30] David Carmel, Yoëlle S. Maarek, Matan Mandelbrod, Yosi Mass, and Aya Soffer. Sea- rch xml documents via xml fragments. In *SIGIR*, pages 151–158, 2003.

[31] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. Xml-gl: a graphical language for querying and restructuring xml documents. In *In Proc. of the Eighth Int'l World Wide Web Conference*, May 1999.

[32] D. D. Chamberlin, J. Robie, and D. Florescu. uilt: An xml query language for heterogeneous data sources. In *Proc. of the Third Int'l Workshop on the Web and Databases*, pages 53–62, 2000.

[33] Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. In *SODA Conference*, pages 192–200, 1998.

[34] Surajit Chaudhuri, Raghu Ramakrishnan, and Gerhard Weikum. Integrating db and ir technologies: What is the sound of one hand clapping? In *CIDR*, pages 1–12, 2005.

[35] Liang Jeff Chen and Yannis Papakonstantinou. Supporting top-k keyword search in xml databases. In *ICDE*, pages 689–700, 2010.

[36] Ting Chen, Jiaheng Lu, and Tok Wang Ling. On boosting holism in xml twig pattern matching using structural indexing techniques. In *SIGMOD Conference*, pages 455–466, 2005.

[37] Sara Cohen, Yaron Kanza, Benny Kimelfeld, and Yehoshua Sagiv. Interconnection semantics for keyword search in xml. In *CIKM*, pages 389–396, 2005.

[38] Sara Cohen, Jonathan Mamou, Yaron Kanza, and Yehoshua Sagiv. XSEarch: A semantic search engine for XML. In *VLDB*, pages 45–56, 2003.

[39] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to algorithms, second edition, 2001.

[40] A. Deutsch, M. F. Fernndez, and D. Florescu. A query language for xml. In *World Wide Web Consortium*, 1998.

[41] Bolin Ding, Jeffrey Xu Yu, Shan Wang, Lu Qin, Xiao Zhang, and Xuemin Lin. Finding top-k min-cost connected trees in databases. *Data Engineering, International Conference on*, pages 836–845, 2007.

[42] Ahmad El Sayed, Hakim Hacid, and Djamel Zighed. Mining semantic distance between corpus terms. In *PIKM*, 2007.

[43] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 102–113, 2001.

[44] Christiane Fellbaum. Wordnet: an electronic lexical database.

[45] Alan Feuer, Stefan Savev, and Javed A. Aslam. Evaluation of phrasal query suggestions. In *CIKM*, pages 841–848, 2007.

[46] Norbert Fuhr and Kai Großjohann. Xirql: A query language for information retrieval in xml documents. In *SIGIR*, pages 172–180, 2001.

[47] J. Teevan G. Murray. Query log analysis: social and technological challenges. In *SIGIR forum*, 2007.

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

[49] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 381–390, 2009.

[50] Jiafeng Guo, Gu Xu, Hang Li, and Xueqi Cheng. A unified and discriminative model for query refinement. In *SIGIR*, pages 379–386, 2008.

[51] Lin Guo, Jayavel Shanmugasundaram, and Golan Yona. Topology search over biological databases. In *ICDE*, pages 556–565, 2007.

[52] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD*, 2003.

[53] Hao He, Haixun Wang, Jun Yang, and Philip S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD*, 2007.

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

[55] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 850–861, 2003.

[56] Vagelis Hristidis and Yannis Papakonstantinou. Discover: Keyword search in relational databases. In *Proc. of VLDB Conference*, pages 670–681, 2002.

[57] Vagelis Hristidis, Yannis Papakonstantinou, and Andrey Balmin. Keyword proximity search on XML graphs. In *ICDE*, pages 367–378, 2003.

[58] Yu Huang, Ziyang Liu, and Yi Chen. Query biased snippet generation in xml search. In *SIGMOD Conference*, pages 315–326, 2008.

[59] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4), 2002.

[60] Haifeng Jiang, Wei Wang 0011, Hongjun Lu, and Jeffrey Xu Yu. Holistic twig joins on indexed xml documents. In *VLDB*, pages 273–284, 2003.

[61] Haifeng Jiang, Hongjun Lu, and Wei Wang 0011. Efficient processing of twig queries with or-predicates. In *SIGMOD Conference*, pages 59–70, 2004.

[62] Haifeng Jiang, Hongjun Lu, Wei Wang 0011, and Beng Chin Ooi. Xr-tree: Indexing xml data for efficient structural joins. In *ICDE*, pages 253–263, 2003.

[63] Rosie Jones and Daniel Fain. Query word deletion prediction. In *SIGIR*, 2003.

[64] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *WWW*, pages 387–396, 2006.

[65] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.

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

[67] Lingbo Kong, Rémi Gilleron, and Aurélien Lemay Mostrare. Retrieving meaningful relaxed tightest fragments for xml keyword search. In *EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology*, pages 815–826, 2009.

[68] Reiner Kraft and Jason Zien. Mining anchor text for query refinement. In *WWW*, pages 666–674, 2004.

[69] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, 2001.

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

[71] Changqing Li and Tok Wang Ling. Qed: a novel quaternary encoding to completely avoid re-labeling in xml updates. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 501–508, 2005.

[72] Changqing Li, Tok Wang Ling, and Min Hu. Efficient processing of updates in dynamic xml data. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, 2006.

[73] Guoliang Li, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. Effective keyword search for valuable lcas over xml documents. In *CIKM*, pages 31–40, 2007.

[74] Guoliang Li, Beng Chin Ooi, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. Ease: Efficient and adaptive keyword search on unstructured, semi-structured and structured data. In *SIGMOD*, 2008.

[75] Mu Li, Yang Zhang, Muhua Zhu, and Ming Zhou. Exploring distributional similarity based models for query spelling correction. In *ACL*, pages 1025–1032, 2006.

[76] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In *Proc. of VLDB*, pages 361–370, 2001.

[77] Wen. Syan Li, K. Selcuk Candan, Quoc Vu, and Divyakant Agrawal. Retrieving and organizing web pages by information unit. In *WWW*, pages 230–244, 2001.

[78] Yunyao Li, Ishan Chaudhuri, Huahai Yang, Satinder Singh, and H. V. Jagadish. Danalix: a domain-adaptive natural language interface for querying xml. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1165–1168, 2007.

[79] Yunyao Li, Cong Yu, and H.V. Jagadish. Schema-free XQuery. In *VLDB*, pages 72–83, 2004.

[80] Ziyang Liu and Yi Chen. Identifying meaningful return information for xml keyword search. In *SIGMOD*, 2007.

[81] Ziyang Liu and Yi Chen. Reasoning and identifying relevant matches for xml keyword search. *PVLDB*, 1(1):921–932, 2008.

[82] Ziyang Liu, Peng Sun, and Yi Chen. Structured search result differentiation. *PVLDB*, 2(1):313–324, 2009.

[83] Jiaheng Lu, Zhifeng Bao, Tok Wang Ling, and Xiaofeng Meng. Content-aware query refinement in xml keyword search. *Submitted to IEEE rans. Knowl. Data Eng.*

[84] Jiaheng Lu, Zhifeng Bao, Tok Wang Ling, and Xiaofeng Meng. XML keyword query refinement. In *KEYS*, pages 41–42, 2009.

[85] Jiaheng Lu, Tok Wang Ling, Zhifeng Bao, and Chen Wang. Extended xml tree pattern matching: Theories and algorithms. *IEEE Trans. Knowl. Data Eng.*, 2010.

[86] Jiaheng Lu, Tok Wang Ling, Chee Yong Chan, and Ting Chen. From region encoding to extended dewey: On efficient processing of xml twig pattern matching. In *VLDB*, pages 193–204, 2005.

[87] Yi Luo, Xuemin Lin, Wei Wang 0011, and Xiaofang Zhou. Spark: top-k keyword query in relational databases. In *SIGMOD Conference*, pages 115–126, 2007.

[88] Eve Maler, Steve DeRose, Eve Maler (arbortext, and Steve Derose (inso Corp. Xml pointer language (xpointer), 1998.

[89] Alexander Markowetz, Yin Yang, and Dimitris Papadias. Keyword search on relational data streams. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 605–616, 2007.

[90] Yosi Mass and Matan Mandelbrod. Component ranking and automatic query refinement for xml retrieval. In *INEX*, 2004.

[91] Andrew Nierman and H. V. Jagadish. Protdb: Probabilistic data in xml. In *In Proceedings of the 28th VLDB Conference*, pages 646–657. Springer, 2002.

[92] Hanglin Pan, Anja Theobald, and Ralf Schenkel. Query refinement by relevance feedback in an xml retrieval system. In *ER*, 2004.

[93] Desislava Petkova, W. Bruce Croft, and Yanlei Diao. Refining keyword queries for xml retrieval by combining content and structure. In *ECIR*, 2009.

[94] David Pinto, Andrew McCallum, Xing Wei, and W. Bruce Croft. Table extraction using conditional random fields. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 235–242, 2003.

[95] Ken Q. Pu. Keyword query cleaning using hidden markov models. In *KEYS '09: Proceedings of the First International Workshop on Keyword Search on Structured Data*, pages 27–32, 2009.

[96] Ken Q. Pu and Xiaohui Yu. Keyword uery cleaning. In *VLDB*, volume 1, pages 909–920, 2008.

[97] Yonggang Qiu and Hans-Peter Frei. Concept based query expansion. In *SIGIR*, pages 160–169, 1993.

[98] Lawrence R. Rabiner. Readings in speech recognition. chapter A tutorial on hidden Markov models and selected applications in speech recognition, pages 267–296. 1990.

[99] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. Html 4.01 specification, 1999. W3C Recommendation, http:// www.w3c.org/TR/html401.

[100] Ian Ruthven. Re-examining the potential effectiveness of interactive query expansion. In *SIGIR*, pages 213–220, 2003.

[101] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1986.

[102] Albrecht Schmidt, Martin L. Kersten, and Menzo Windhouwer. Querying xml documents made easy: Nearest concept queries. In *ICDE*, pages 321–329, 2001.

[103] Amanda Spink, Bernard J. Jansen, Dietmar Wolfram, and Tefko Saracevic. From e-sex to e-commerce: Web search changes. *IEEE Computer*, 35(3):107–109, 2002.

[104] Chong Sun, Chee Yong Chan, and Amit K. Goenka. Multiway slca-based keyword search in xml data. In *WWW*, 2007.

[105] Igor Tatarinov, Stratis D. Viglas, Kevin Beyer, Jayavel Shanmugasundaram, Eugene Shekita, and Chun Zhang. Storing and querying ordered xml using a relational database system. In *SIGMOD*, pages 204–215, 2002.

[106] Anja Theobald and Gerhard Weikum. The index-based xxl search engine for querying xml data with relevance ranking. In *EDBT*, pages 477–495, 2002.

[107] Martin Theobald, Holger Bast, Debapriyo Majumdar, Ralf Schenkel, and Gerhard Weikum. Topx: efficient and versatile top-k query processing for semistructured data. *The VLDB Journal*, 17(1):81–115, 2008.

[108] Quang Hieu Vu, Beng Chin Ooi, Dimitris Papadias, and Anthony K. H. Tung. A graph method for keyword-based selection of the top-k databases. In *SIGMOD Conference*, pages 915–926, 2008.

[109] Huayu Wu, Tok Wang Ling, Gillian Dobbie, Zhifeng Bao, and Liang Xu. Reducing graph matching to tree matching for XML queries with id references. In *DEXA (2)*, pages 391–406, 2010.

[110] Huayu Wu, Tok Wang Ling, Liang Xu, and Zhifeng Bao. Performing grouping and aggregate functions in XML queries. In *WWW*, pages 1001–1010, 2009.

[111] X. Wu, M. Lee, and W. Hsu. A prime number labeling scheme for dynamic ordered XML trees. In *Proc. of ICDE*, pages 66–78, 2004.

[112] Yuqing Wu, Jignesh M. Patel, and H. V. Jagadish. Structural join order selection for xml query optimization. In *ICDE*, pages 443–454, 2003.

[113] XSLT. http://www.w3.org/Style/XSL/.

[114] Jinxi Xu and W. Bruce Croft. Improving the effectiveness of information retrieval with local context analysis. *ACM Trans. Inf. Syst.*, 18(1):79–112, 2000.

[115] Liang Xu, Zhifeng Bao, and Tok Wang Ling. A dynamic labeling scheme using vectors. In *DEXA '07: Proceedings of the 18th international conference on Database and Expert Systems Applications*, pages 130–140, 2007.

[116] Liang Xu, Tok Wang Ling, Zhifeng Bao, and Huayu Wu. Efficient label encoding for range-based dynamic XML labeling schemes. In *DASFAA*, 2010.

[117] Liang Xu, Tok Wang Ling, Huayu Wu, and Zhifeng Bao. Dde: from dewey to a fully dynamic XML labeling scheme. In *SIGMOD*, pages 719–730, 2009.

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

[119] Yu Xu and Yannis Papakonstantinou. Efficient lca based keyword search in xml data. In *EDBT*, pages 535–546, 2008.

[120] Bei Yu, Guoliang Li, Karen Sollins, and Anthony K. H. Tung. Effective keyword-based selection of relational databases. In *SIGMOD*, pages 139–150, 2007.

[121] Xiaohui Yu and Huxia Shi. Query segmentation using conditional random fields. In *KEYS '09: Proceedings of the First International Workshop on Keyword Search on Structured Data*, 2009.

[122] C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, and G. M. Lohman. On supporting containment queries in relational database management systems. In *Proc. of SIGMOD Conference*, pages 425–436, 2001.

[123] Junfeng Zhou, Zhifeng Bao, Tok Wang Ling, and Xiaofeng Meng. MCN: A new semantics towards effective xml keyword search. In *DASFAA*, pages 511–526, 2009.