Interactive Search in XML Data

Guoliang Li Jianhua Feng Lizhu Zhou
Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China
{liguoliang,fengjh,dcszlz}@tsinghua.edu.cn

ABSTRACT

In a traditional keyword-search system in XML data, a user composes a keyword query, submits it to the system, and retrieves relevant subtrees. In the case where the user has limited knowledge about the data, often the user feels "left in the dark" when issuing queries, and has to use a try-and-see approach for finding information. In this paper, we study a new information-access paradigm for XML data, called "INKS," in which the system searches on the underlying data "on the fly" as the user types in query keywords. INKS extends existing XML keyword search methods by *interactively* answering keyword queries. We propose effective indices, early-termination techniques, and efficient search algorithms to achieve a high interactive speed. We have implemented our algorithm. The experimental results show that INKS achieves high search efficiency and result quality.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*query formulation, search process*

General Terms

Algorithms, Experimentation, Performance

Keywords

XML, Keyword Search, Interactive Search, Autocomplete

1. INTRODUCTION

Keyword search provides a simple and user-friendly query interface to access XML data in web and scientific applications [4, 2, 8, 7, 5, 6]. In an existing XML keyword search system, a user composes a query, submits it to the system, and retrieves relevant answers. This information-access paradigm requires the user to have certain knowledge about the content of the underlying data. In the case where the user has limited knowledge about the data, often the user feels "left in the dark" when issuing queries, and has to use a try-and-see approach for finding information. Many systems are introducing various features to solve this problem. One of the commonly used methods is *autocomplete*, which predicts a word or phrase that the user may type in based on the partial string the user has typed.

In this paper, we extend *autocomplete* and propose an interactive keyword-search method in XML data, called INKS. INKS searches XML data on the fly as users type in queries and provides a friendly interface for users exploring XML data. INKS can significantly save users typing effort. In contrast, one limitation of autocomplete is that the system treats a query with multiple keywords as a single string, thus it does not allow keywords to appear at different places. For instance, consider the search box on Apple.com. Although a query "itunes" can find a record "itunes wi-fi music store," a query "itunes music" cannot find this record, because the two keywords appear at different places. CompleteSearch [1] interactively searches on a set of documents. INKS extends autocomplete and CompleteSearch to find *relevant subtrees* in XML data by supporting multiple keywords.

We give an example to show how INKS works. Assume there is an XML document that resides on a server. A user accesses and searches the data through a Web browser. Each keystroke that the user types invokes a query, which includes the current string the user has typed. The browser sends the query to the server, which computes and returns to the user the best answers ranked by their relevancy to the query.

Assume a user types in a query "`db mic`" letter by letter on the XML data in Figure 1. The string is tokenized to keywords using delimiters. The keywords are assumed as *partial keywords*, as the user may have not finished typing the complete keyword. For the partial keywords, we would like to know the possible words the user intends to type. We identify a set of words with this partial keyword as a prefix. This set of keywords are called the *predicted words*. For instance, for the partial keyword "`mic`," its predicted word could be "`mices`," "`mich`," etc. Then based on the predicted words, we identify the relevant subtrees in XML data that contain the predicted words. We call these relevant subtrees *predicted answers*. Apparently, INKS can significantly save users time and efforts, since they can find answers even if they have not finished typing all complete keywords.

2. LCA-BASED INTERACTIVE SEARCH

We propose a lowest common ancestor(LCA) based interactive-search method. We use the semantics of exclusive LCA (ELCA) [4] to identify relevant answers for predicted words. We use a trie to index the tokenized words in XML data.

For a query with a single keyword, we first find the corresponding trie node. Then we locate the leaf descendants of this node, and retrieve the corresponding predicted words and the predicted XML elements on their inverted lists. For a query with multiple keywords, we first tokenize the query string into keywords, $k_1, k_2, \ldots, k_\ell$. For each keyword k_i $(1 \leq i \leq \ell)$, there are multiple predicted words. Suppose there are q_i predicted words for k_i, and their corresponding XML element lists are $I_{i_1}, I_{i_2}, \ldots, I_{i_{q_i}}$. We fist com-

WWW 2009, April 20–24, 2009, Madrid, Spain.
ACM 978-1-60558-487-4/09/04.

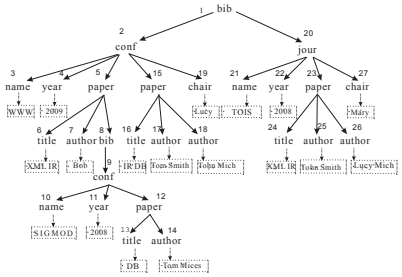

Figure 1: An XML document

pute the predicted XML element lists of the partial keyword, i.e., the union of these lists $U_i = \cup_{j=1}^{q_i} I_{i_j}$. Then, we compute the predicted answers, i.e., the subtrees of ELCAs on $U_1, U_2, \cdots, U_\ell$. We use the binary search based method to compute ELCAs and corresponding answers [8]. We use the tf*idf based ranking functions [4] to rank the answers.

Assume a user types in a query "db mic" letter by letter. For query "d," we locate the trie node for "d" and identify predicted word "db" and predicted XML elements 13 and 16 on its leaf descendants. For query "db m," we identify predicted words "mices" and "mich" for"m" and predicted elements 14, 18, and 26. Finally, we compute ELCAs on {13, 16} and {14, 18, 26} and get XML elements 12 and 15.

3. PROGRESSIVE SEARCH

Existing XML keyword-search algorithms [4] have two main limitations. First, they use the default "AND" semantics between input keywords. Second, they find candidate nodes first before ranking them, and this approach is not efficient for computing the best answers.

To address these limitations, we develop novel ranking techniques and efficient search algorithms. In our approach, each node on the XML tree could be potentially relevant to a keyword query. For each keyword in the tree, we index not only the *content nodes* containing the keyword, but also those *quasi content nodes* whose descendants contain the keyword. Given a keyword and node n, a *pivotal node* is a content node for the keyword, which has a minimal distance to n. The path from node n to this node is called the *pivotal path*. We introduce the notion of *minimal-cost tree* (MCT for short) to define the answer to the query for node n. The minimal-cost tree is the subtree rooted at n that includes all pivotal paths for input keywords and node n.

For example, for "DB," we index nodes 13, 16, 12, 15, 9, 2, 8, 1, and 5, sorted by relevance. For "Tom," we index nodes 14, 17, 12, 15, 9, 2, 8, 1, and 5. Node 13 is the pivotal node for node 12 and "DB," and its pivotal path is 12-13. Node 14 is the pivotal node for node 12 and "Tom," and its pivotal path is 12-14. For query "DB Tom," to identify the top-2 answers, we first find nodes 12 and 15 based on the index and then construct MCTs based on pivotal paths.

Now we discuss how to rank an MCT. Intuitively, we first evaluate the relevance between node n and each input keyword, and then combine these relevance scores as the overall score of the MCT. We can use the idea of tf*idf to score the relevance of the content nodes, but cannot rank a quasi content node. Given a quasi content node, we combine its pivotal node's tf*idf score and the distance between the quasi content node and its pivotal node for effective ranking.

We propose how to do progressive search in considering "OR" predicate. In the trie index, for leaf nodes, we keep content nodes and quasi content nodes, and corresponding scores and pivotal paths, sorted by their scores. For each internal node, we cache top-n relevant ones among (quasi) content nodes in its subtree. Given a query, for each keyword, we first locate the corresponding node on the trie. Then, we retrieve top-n (cached) relevant elements. We use the threshold-based algorithm [3] to identify the top-k answers. If we can guarantee that we have found the top-k answers using the cached elements, we can do early termination; otherwise, we retrieve the predicted XML elements and use the threshold-based method to find top-k answers.

4. EXPERIMENTAL STUDY

We have implemented our method on real dataset DBLP with the size of 470 MB. We set up a server using Apache and FastCgi. The server was running a program implemented in C++ using a GNU compiler. We conducted the evaluation on a PC running a Ubuntu Linux with an Intel(R) Xeon(R) CPU X5450@3.00 GHz CPU and 4 GB RAM. We selected ten queries on the dataset. We first evaluate result quality by human judgement. Answer relevance of the selected queries was judged from discussions of twenty randomly selected person. Figure 2 shows the top-10 precision. We observe that the progressive method achieves higher result quality than LCA based methods (We implemented XRank to generate answers for LCA based methods). This is attributed to our effective ranking functions. We then evaluate the server running time. Figure 3 gives the total server time for different queries. We observe that our progressive method achieves higher efficiency.

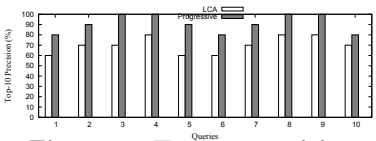

Figure 2: Top-10 precision

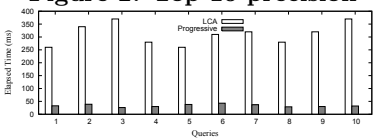

Figure 3: Elapsed server time

5. ACKNOWLEDGEMENT

This work is partly supported by the National High Technology Development 863 Program of China under Grant No.2007AA01Z152, the National Grand Fundamental Research 973 Program of China under Grant No.2006CB303103, and 2008 HP Labs Innovation Research Program.

6. REFERENCES

[1] H. Bast and I. Weber. Type less, find more: fast autocompletion search with a succinct index. In *SIGIR*, pages 364–371, 2006.

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

[3] R. Fagin. Fuzzy queries in multimedia database systems. In *PODS*, pages 1–10, 1998.

[4] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. Xrank: Ranked keyword search over xml documents. In *SIGMOD Conference*, pages 16–27, 2003.

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

[6] Z. Liu and Y. Chen. Identifying meaningful return information for xml keyword search. In *SIGMOD Conference*, pages 329–340, 2007.

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

[8] Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest lcas in xml databases. In *SIGMOD Conference*, pages 537–538, 2005.