XML Data Search with Query Assistance and Semantic Analysis

Selvi.V[*1],Rajkumar.S[*2]
[1]PG Scholar of Computer Science, K.S.Rangasamy College of Technology, Tiruchengode, India.
Email: selvipriya.6323@gmail.com, Mobile No: +91 8973015602.
[2]Assistant professor (Academic), K S Rangasamy College of Technology, Tiruchengode, India.

***Abstract*—XML documents are semi-structured databases that maintain the document and content description. XPath and XQuery query languages are used to query XML data. XQuery is fairly complicated to understand its structure. Query languages require the knowledge about the document schema. Keyword based search models does not requires the prior knowledge about the XML document structure. Searching XML documents is achieved in different ways. In keyword query model search query keyword is passed to the system to fetch the relevant documents. Fuzzy type-ahead search in XML data scheme is applied to search XML documents with query keyword. Auto-complete and auto-correction methods are used to submit query keywords. Index structures and searching algorithms are used to improve the quality and ranking process. Edit distance is used to quantify the similarity between two words. Minimal cost tree is constructed to index the keywords. Exact search and fuzzy search techniques are applied to fetch documents. The top-K results are fetched from top-K relevance technique. Semantic analysis based method is integrated with the fuzzy type-ahead search scheme to improve the query result accuracy. Index model is improved with keyword relevancy and weight values. The system is enhanced with search history based query assistance scheme. Weight threshold based retrieval is provided in the system.**

Index Terms—XML,Keyword search, Type-ahead search, Fuzzy search, Semantic Analysis

I.INTRODUCTION

Traditional methods use query languages such as XPath and XQuery to query XML data. These methods are powerful but unfriendly to nonexpert users. First, these query languages are hard to comprehend for non database users. For example, XQuery is fairly complicated to grasp. Second, these languages require the queries to be posed against the underlying, sometimes complex, database schemas. Fortunately, keyword search is proposed as an alternative means for querying XML data, which is simple and yet familiar to most Internet users as it only requires the input of keywords. Keyword search is a widely accepted search paradigm for querying document systems and the World Wide Web. Recently, the database research community has been studying challenges related to keyword search in XML data. One important advantage of keyword search is that it enables users to search information without knowing a complex query language such as XPath or XQuery, or having prior knowledge about the structure of the underlying data.

In this paper, we propose TASX (pronounced "task"), a fuzzy type-ahead search method in XML data [1]. TASX searches the XML data on the fly as users type in query keywords, even in the presence of minor errors of their keywords. TASX provides a friendly interface for users to explore XML data, and can significantly save users typing effort. In this paper, we study research challenges that arise naturally in this computing paradigm. The main challenge is search efficiency. Each query with multiple keywords needs to be answered efficiently. To make search really interactive, for each keystroke on the client browser, from the time the user presses the key to the time the results computed from the server are displayed on the browser, the delay should be as small as possible. An interactive speed requires this delay should be within milliseconds. Notice that this time includes the network transfer delay, execution time on the server, and the time for the browser to execute its Java-Script. This low-running time requirement is especially challenging when the backend repository has a large amount of data. To achieve our goal, we propose effective index structures and algorithms to answer keyword queries in XML data. We examine effective ranking functions and early termination techniques to progressively identify top-k answers. To the best of our knowledge, this is the first paper to study fuzzy type-ahead search in XML data.

II. RELATED WORK

Keyword search in XML data has attracted great attention recently. Xu and Papakonstantinou proposed smallest lowest common ancestor (SLCA) to improve search efficiency. Sun et al. studied multi-way SLCA-based keyword search to enhance search performance. Schema free XQuery employed the idea of meaningful LCA, and proposed a stack-based sort-merge algorithm by considering XML structures and

incorporating a new function mlcas into XQuery. XSEarch focuses on the semantics and the ranking of the results, and extends keyword search. It employs the semantics of meaningful relation between XML nodes to answer keyword queries, and two nodes are meaningfully related if they are in a same set, which can be given by administrators or users. Li et al. proposed valuable LCA (VLCA) to improve the meaningfulness and completeness of answers and devised a new efficient algorithm to identify the answers based on a stack-based algorithm. XKeyword is proposed to offer keyword proximity search over XML documents, which models XML documents as graphs by considering IDREFs between XML elements. Hristidis et al. proposed grouped distance minimum connecting tree (GDMCT) to answer keyword queries, which groups the relevant subtrees to answer keyword queries. It first identifies the minimum connected tree, which is a subtree with minimum number of edges, and then groups such trees to answer keyword queries. Shao et al. studied the problem of keyword search on XML views. XSeek studied how to infer the most relevant return nodes without elicitation of user preferences. Liu and Chen proposed to reason and identify the most relevant answers. Huang et al. discussed how to generate snippets of XML keyword queries. Bao et al. [5] proposed to address the ambiguous problem of XML keyword search through studying search for and search via nodes. Different from [6], we extended it to support fuzzy type-ahead search in XML data.

In addition, the database research community has recently studied the problem of keyword search in relational databases, graph databases and heterogenous data sources. DISCOVER-I, DISCOVER-II, BANKS-I, BANKS-II and DBXplorer are recent systems to answer keyword queries in relational databases. DISCOVER and DBXplorer return the trees of tuples connected by primary-foreign-key relationships that contain all query keywords. DISCOVER-II extended DISCOVER to support keyword proximity search in terms of disjunctive (OR) semantics, different from DISCOVER which only considers the conjunctive (AND) semantics. BANKS proposed to use Steiner trees to answer keyword queries. It first modeled relational data as a graph where nodes are tuples and edges are foreign keys, and then found Steiner trees in the graph as answers using an approximation to the Steiner tree problem, which is proven to be an NP-hard problem. BANKS-II improved BANKS-I by using bidirectional expansion on graphs to find answers. He et al. proposed a partition based method to efficiently find Steiner trees using the BLINKS index. Ding et al. proposed to use dynamic programming for identifying Steiner trees. Dalvi et al. studied disk-based algorithms for keyword search on large graphs, using a new concept of "superrnode graph."

Tao and Yu proposed to find co-occurring terms of query keywords in addition to the answers, in order to provide users relevant information to refine the answers. Koutrika et al. [3] proposed data clouds over structured data to summarize the results of keyword searches over structured data and use them to guide users to refine searches. Zhang et al. and Felipe et al. studied keyword search on spatial databases by combining inverted lists and R-tree indexes. Tran et al. [8] studied top-k keyword search on RDF data using summarized RDF graph. Qin et al. [7] studied three different semantics of m-keyword queries, namely, connect-tree semantics, distinct core semantics, and distinct root semantics, to answer keyword queries in relation databases. The search efficiency is achieved by new tuple reduction approaches that prune unnecessary tuples in relations effectively followed by processing the final results over the reduced relations. Chu et al. [10] proposed to combine forms and keyword search, and studied effective summary techniques to design forms. Yu et al. and Vu et al. studied keyword search over multiple databases in P2P environment. They emphasized on how to select relevant database sources in P2P environments. Chen et al. [9] gave an excellent tutorial of keyword search in XML data and relational databases..

Type-ahead search is a new topic to query relational databases. Li et al. studied type-ahead search in relational databases, which allows searching on the underlying relational databases on the fly as users type in query keywords. J_i et al. [2] studied fuzzy type-ahead search on a set of tuples/documents, which can on the fly find relevant answers by allowing minor errors between input keywords and the underlying data. A straightforward method for type ahead search in XML data is to first find all predicted words, and then use existing search semantics, e.g., LCA and ELCA, to compute relevant answers based on the predicted words. However, this method is very time consuming for finding top-k answers. To address this problem, we propose to progressively find the most relevant answers. For exact search, we propose to incrementally compute predicted words. For fuzzy search, we use existing techniques to compute predicted words of query keywords. We extend the ranking functions in [4] to support fuzzy search, and propose new index structures and efficient algorithms to progressively find the most relevant answers.This paper extended the poster paper [11] by adding efficient algorithms and ranking techniques to support fuzzy search.

III.FUZZY TYPE-AHEAD SEARCH IN XML DATA

In this section, we introduce the overview of fuzzy type-ahead search in XML data and formalize the problem. We first introduce how TASX works for queries with multiple keywords in XML data, by allowing minor errors of query keywords and inconsistencies in the data itself. Assume there is an underlying 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 in. 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.

The server first tokenizes the query string into several keywords using delimiters such as the space character. The keywords are assumed as partial keywords, as the user may have not finished typing the complete keywords. For the partial keywords, we would like to know the possible words the user intends to type. However, given the limited information, we can only identify a set of complete words in the data set which have similar prefixes with the partial keywords. This set of complete words are called the predicted words. We use edit distance to quantify the similarity between two words. The edit distance between two words s_1 and s_2, denoted by $ed(s_1; s_2)$, is the minimum number of edit operations of single characters needed to transform the first one to the second. For example, ed(mics, mices) = 1 and ed(mics, mich) = 1. For instance, given a partial keyword "mics," its predicted words could be "mices," "mich," "michal," etc.

Then, the server identifies the relevant subtrees in XML data that contain the predicted words for every input keyword. We can use any existing semantics to identify the answer based on the predicted words, such as ELCA. We call these relevant subtrees the predicted answers of the query. For example, consider the XML document in Fig. 1. Assume a user types in a keyword query "db mics." The predicted word of "db" is "db." The predicted words of "mics" are "mices" and "mich." The subtree rooted at node 12 is the predicted answer of "db mices." The subtree rooted at node 15 is the predicted answer of "db mich." Thus, TASX can save users time and efforts, since they can find the answers even if they have not finished typing all the complete keywords or typing keywords with minor errors.

IV.LCA-BASED FUZZY TYPE-AHEAD SEARCH

This section proposes an LCA-based fuzzy type-ahead search method. We use the semantics of ELCA to identify relevant answers on top of predicted words.

A. Index Structures

We use a trie structure to index the words in the underlying XML data. Each word w corresponds to a unique path from the root of the trie to a leaf node. Each node on the path has a label of a character in w. For each leaf node, we store an inverted list of IDs of XML elements that contain the word of the leaf node. For instance, consider the XML document in Fig. 1. The trie structure for the tokenized words "mich" has a node ID of 10. Its inverted list includes XML elements 18 and 26.

B.Answering Queries with a Single Keyword

We first study how to answer a query with a single keyword using the trie structure. Each keystroke that a user types invokes a query of the current string, and the client browser sends the query string to the server.

1). Exact Search

We first consider the case of exact search. One naive way to process such a query on the server is to answer the query from scratch as follows: we first find the trie node corresponding to this keyword by traversing the trie from the root. Then, we locate the leaf descendants of this node, and retrieve the corresponding predicted words and the predicted XML elements on the inverted lists.

2). Fuzzy Search

Obviously, for exact search, given a partial keyword, there exists at most one trie node for the keyword. We retrieve the leaf descendants of this trie node as the predicted words. However, for fuzzy search, there could be multiple trie nodes that are similar to the partial keyword within a given edit-distance threshold, called active nodes.

C.Answering Queries with Multiple Keywords

Now, we consider how to do fuzzy type-ahead search in the case of a query with multiple keywords. For a keystroke that invokes a query, we first tokenize the query string into keywords, $k_1, k_2, \ldots, k'$. For each keyword k_i ($1 \leq i \leq \ell$), we compute its corresponding active nodes, and for each such active node, we retrieve its leaf descendants and corresponding inverted lists. Then, we compute union list $\overline{U}_{ki}$ for every k_i as discussed in Section 4.2. Finally, we compute the predicted answers on top of lists $\overline{U}_{k1}, \overline{U}_{k2}, \ldots, \overline{U}_{k\ell}$ '.

V. PROGRESSIVE AND EFFECTIVE TOP-K FUZZY TYPE-AHEAD SEARCH

The LCA-based fuzzy type-ahead search algorithm in XML data has two main limitations. First, they use the "AND" semantics between input keywords of a query, and ignore the answers that contain some of the query keywords. For example, suppose a user types in a keyword query "DB IR Tom" on the XML document. The ELCAs to the

query are nodes 15 and 5. Although node 12 does not have leaf nodes corresponding to all the three keywords, it might still be more relevant than node 5 that contains many irrelevant papers. Second, in order to compute the best results to a query, existing methods need find candidates first before ranking them, and this approach is not efficient for computing the best answers. A more efficient algorithm might be able to find the best answers without generating all candidates.

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, and we use a ranking function to decide the best answers to the query. For each leaf node in the trie, we index not only the content nodes for the keyword of the leaf node, but also those quasi-content nodes whose descendants contain the keyword. For instance, consider the XML document in Fig. 1. For the keyword "DB," we index nodes 13, 16, 12, 15, 9, 2, 8, 1, and 5 for this keyword. For the keyword "IR," we index nodes 6, 16, 24, 5, 15, 23, 2, 20, and 1. For the keyword "Tom," we index nodes 14, 17, 12, 15, 9, 2, 8, 1, and 5. The nodes are sorted by their relevance to the keyword.

For instance, assume a user types in a keyword query "DB IR Tom." We use the extended trie structure to find nodes 15 and 12 as the top-2 relevant nodes. We propose minimal-cost trees (MCTs) to construct the answers rooted at nodes 15 and 12. We develop effective ranking techniques to rank XML elements on the inverted lists in the extended trie structure. We can employ threshold-based algorithms to progressively and efficiently identify the top-k relevant answers. Moreover, our approach automatically supports the "OR" semantics.

VI. SEMANTIC ANALYSIS FOR XML DATA SEARCH

The XML document search system is enhanced with a set of features. They are semantic analysis, indexing model, query assistance with historical data and weight threshold models. The semantic analysis is used to identify the semantic relations. The index model is used to improve the document indexing process. The history based query assistance is also provided in the system. The top-K query model and weight threshold models are used in the document search process.

A. Semantic Analysis

The semantic analysis model is used to assess term relationship. The Ontology is used for the semantic relationship analysis. The concept and term relationships are verified from the semantic analysis model. The fuzzy type-ahead search scheme is improved with semantic analysis method. XML data query is parsed and its concept is identified from the Ontology support. The concept and term details are also verified from the semantic analysis modules.

B. Indexing Model

The index model is used to rearrange the data values. The tree based index model is used in the system. The minimum cost based index structure is used in the system. Index model is improved with keyword relevancy and weight values. Tree based cost effective model is used for the data indexing and retrieval process.

C. Query Assistance Features

The query assistance is the main function of the system. The fuzzy type head model is used with the system. The system is enhanced with search history based query assistance scheme. Single word and multiple word based query assistance model is used in the system. User personal and global search history based models are used to improve the query assistance process.

D. Weight Threshold Model

The XML data search method supports top-K model for the data query submission under the query assistance environment. The documents are assigned with semantic weight values. The weight based ranking is used in the system. Weight threshold based retrieval is provided in the system. The weight threshold value is collected from the user to filter the documents with weight boundaries.

VII.CONCLUSION

XML documents are constructed to maintain and distribute data values. Fuzzy type-ahead search method in XML data (TASX) is applied to fetch XML documents using query keywords. TASX scheme is enhanced with semantic analysis and weight based index structure. Search history and weight threshold based models are used to improve retrieval quality. Effective index structures, efficient algorithms and novel optimization techniques are used to progressively and efficiently identify the top-k answers. A minimal-cost-tree-based search method is developed to efficiently and progressively identify the most relevant answers. The system also supports semantic analysis and search history based query assistance mechanism for the XML document search process. Retrieved documents are ranked with relevant levels. Ranking functions and early termination techniques are used to progressively identify top-k answers and weight threshold query results. The system reduces the user typing efforts on query keywords. User friendly interface supports query preparation process.

REFERENCES

[1] Jianhua Feng and Guoliang Li, "Efficient Fuzzy Type-Ahead Search in XML Data" IEEE Transactions On Knowledge And Data Engineering, Vol. 24, No. 5, May 2012

[2] S. Ji, G. Li, C. Li, and J. Feng, "Efficient Interactive Fuzzy Keyword Search," Proc. Int'l Conf. World Wide Web (WWW), pp. 371-380, 2009.
[3] G. Koutrika, Z.M. Zadeh, and H. Garcia-Molina, "Data Clouds: Summarizing Keyword Search Results over Structured Data," Proc. Int'l Conf. Extending Database Technology: Advances in Database Technology (EDBT), pp. 391-402, 2009.
[4] G. Li, S. Ji, C. Li, and J. Feng, "Efficient Type-Ahead Search on Relational Data: A Tastier Approach," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 695-706, 2009.
[5] Z. Bao, T.W. Ling, B. Chen, and J. Lu, "Effective XML Keyword Search with Relevance Oriented Ranking," Proc. Int'l Conf. Data Eng. (ICDE), 2009.
[6] G. Li, C. Li, J. Feng, and L. Zhou, "Sail: Structure-Aware Indexing for Effective and Progressive Top-k Keyword Search over XML Documents," Information Sciences, vol. 179, no. 21, pp. 3745-3762, 2009.
[7] L. Qin, J.X. Yu, and L. Chang, "Keyword Search in Databases: The Power of Rdbms," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 681-694, 2009.
[8] T. Tran, H. Wang, S. Rudolph, and P. Cimiano, "Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data," Proc. Int'l Conf. Data Eng. (ICDE), pp. 405-416, 2009.
[9] Y. Chen, W. Wang, Z. Liu, and X. Lin, "Keyword Search on Structured and Semi-Structured Data," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 1005-1010, 2009.
[10] E. Chu, A. Baid, X. Chai, A. Doan, and J.F. Naughton, "Combining Keyword Search and Forms for Ad Hoc Querying of Databases," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 349-360, 2009.
[11] G. Li, J. Feng, and L. Zhou, "Interactive Search in Xml Data," Proc. Int'l Conf. World Wide Web (WWW), pp. 1063-1064, 2009.