Intelligent Fuzzy Type-Ahead Search in XML Data

Supriya Chaudhari[1], Vaishali Deshmukh[2]

[1] SGBA University, Department of Computer Science & Engineering, PRMIT&R, Badnera, India

[2] Professor SGBA University, Department of Computer Science & Engineering, PRMIT&R, Badnera, India

Abstract: ***In a traditional keyword-search system over XML data, a user composes a keyword query, submits it to the system, and retrieves relevant answers. 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 fuzzy type-ahead search in XML data, a new information-access paradigm in which the system searches XML data on the fly as the user types in query keywords. It allows users to explore data as they type, even in the presence of minor errors of their keywords. Our proposed method has the following features: 1) Search as you type: It extends Auto complete by supporting queries with multiple keywords in XML data.2) Fuzzy: It can find high-quality answers that have keywords matching query keywords approximately. 3) Intelligent: Our effective index structures, searching algorithms and materialized views can achieve a very high interactive speed. Answering queries using materialized views has been well studied in the context of structured queries and has shown significant performance benefits.***

Keywords: XML, keyword search, type-ahead search, fuzzy search, Materialized views, Data warehouses, Clustering, Complex data.

1. Introduction

Traditional methods use query languages such as XPath and XQuery to query XML data. These methods are powerful but unfriendly to non expert 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. 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 a traditional keyword-search system over XML data, a user composes a query, submits it to the system, and retrieves relevant answers from XML data. This information- access paradigm requires the user to have certain knowledge about the structure and content of the underlying data repository. 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. He tries a few possible keywords, goes through the returned results, modifies the keywords, and reissues a new query. He needs to repeat this step multiple times to find the information, if lucky enough. This search interface is neither efficient nor user friendly. Many systems are introducing various features to solve this problem. One of the commonly used methods is Auto complete, which predicts a word or phrase that the user may type in based on the partial string the user has typed. More and more websites support this feature. One limitation of Auto complete is that the system treats a query with multiple keywords as a single string, thus, it does not allow these keywords to appear at different places.

Type-ahead search can provide users instant feedback as users type in keywords, and it does not require users to type in complete keywords. Type-ahead search can help users browse the data, save users typing effort, and efficiently find the information. They also studied type-ahead search in relational databases [1]. However, existing methods cannot search XML data in a type-ahead search manner, and it is not trivial to extend existing techniques to support fuzzy type-ahead search in XML data. This is because XML contains parent-child relationships, and we need to identify relevant XML subtrees that capture such structural relationships from XML data to answer keyword queries, instead of single documents.

By avoiding computing query results directly from the source data, exploiting materialized views has been proven crucial for performance optimization in evaluating SQL queries on databases and XPath/XQuery on XML. Caching query results as materialized views in web applications can also reduce the workload of servers and network traffic. Given the benefits of materialized views in structured query processing, it is a natural idea to leverage them to speed up XML keyword search.

2. Literature Review

Keyword search in XML data has attracted great attention recently. Xu and Papakonstantinou [2] proposed smallest lowest common ancestor (SLCA) to improve search efficiency. Sun et al. [6] studied multiway SLCA-based keyword search to enhance search performance. Schema free XQuery [4] 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 [3] 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. [7] 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 [8] 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. [9] 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. [10] studied the problem of keyword search on XML views.

Type-ahead search is a new topic to query relational databases. Li et al. [1] studied type-ahead search in relational databases, which allows searching on the underlying relational databases on the fly as users type in query keywords. Recently, there have been several papers on selection of materialized views in the OLAP/Data. The work by Baralis et al. [11] is also set in the context of OLAP/Data Cube and does not consider traditional indexes on base tables. For a given workload, they consider materialized views that exactly match queries in the workload, as well as a set of additional views that can leverage commonality among queries in the workload. In the context of SQL databases and workloads, the work by [12] picks materialized views by examining the plan information of queries.

2.1 Notations

An XML document can be modeled as a rooted and labeled tree. A node v in the tree corresponds to an element in the XML document and has a label. For two nodes u and v, we use "$u \prec v$" ("$u \succ v$." respectively) to denote that node u is an ancestor (descendant, respectively) of node v. We use "$u \preceq v$" to denote that $u \prec v$ or $u = v$. For example, consider the XML document in Fig. 1, we have paper (node 5) $\prec$ author (node 7) and paper (node 12) $\succ$ conf(node2).

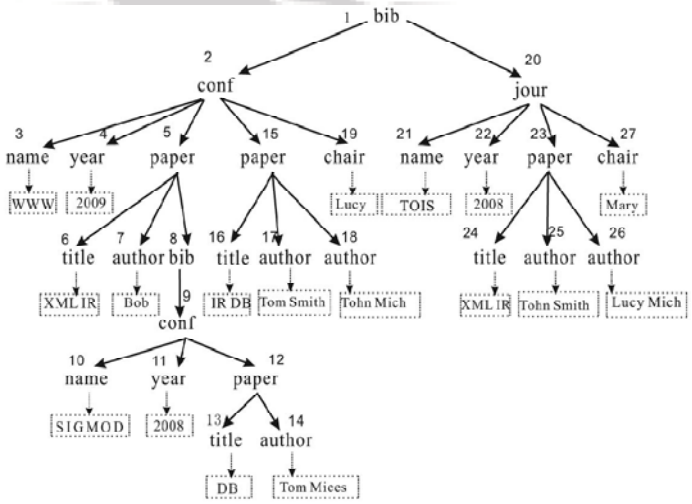


Figure 1: An XML document.

A keyword query consists of a set of keywords {k_1, k_2,…. , k_l}. For each keyword ki, we call the nodes in the tree that contain the keyword the content nodes for ki. The ancestor nodes1 of the content nodes are called the quasi-content nodes of the keyword. For example, consider the XML document in Fig.1, title (node 16) is a content node for keyword "DB," and conf (node 2) is a quasi-content node of keyword "DB."

3. Progressively Computing Top K Answers

3.1 Architecture

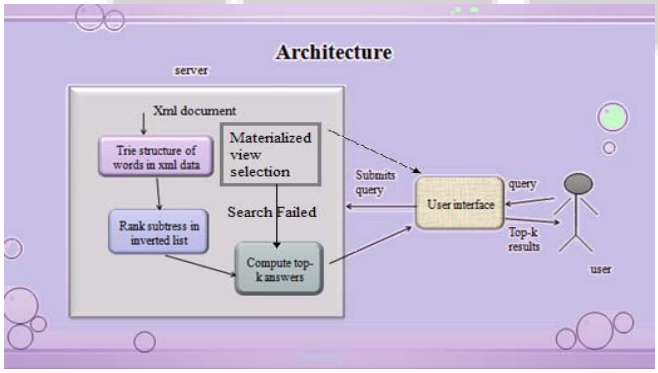


Figure 2: Architecture for compute top k answers

The user composes query and submits. The browser sends the query to the server where the xml document, its index structure and materialized views are lies in it. First the keyword search begin with materialized view if word found in view then server returns the results to user through browser and if not then the ranking module ranks the nodes that contain the keyword. The compute top k results uses effective algorithms to compute top k results and the server returns the results to user through browser.

Problem Formulation:

We formalize the problem of fuzzy type-ahead search in XML data as follows:

Definition 1: (FUZZY TYPE-AHEAD SEARCH IN XML DATA). Given an XML document D, a keyword query Q ={k_1,k_2,….k_l} and an edit-distance threshold $_T$. Let the predicted-word set be W_k={w|w is a tokenized word in D and there exists a prefix of w, k_i',ed(ki,k_i')<=$_T$.} Let the predicted answer set be R_Q={r|r is a keyword-search result of query {$w_1 \in W_{k1}$, w2 $\in W_{k2}$, . . . ,$W_l \in W_{kl}$. For the keystroke that invokes Q, we return the top-k answers in RQ for a given value k, ranked by their relevancy to Q.

Let treat the data and query string as lowercase strings. Now focus on how to efficiently find the predicted answers, among which we can find the best top-k relevant answers using a ranking function. There are two challenges to support fuzzy type-ahead search in XML data. The first one is how to interactively and efficiently identify the predicted words that have prefixes similar to the input partial keyword after each keystroke from the user. The second one is how to progressively and effectively compute the top-k predicted answers of a query with multiple keywords, especially when there are many predicted words.

3.2 Methods for Keyword Search Over Xml Data

3.2.1 LCA based method

The lowest common ancestor (LCA) is a concept in graph theory and computer science. Let *T* be a rooted tree with *n* nodes. The lowest common ancestor between two nodes *v* and *w* is defined as the lowest node in *T* that has both *v* and *w* as descendants.

The LCA of *v* and *w* in *T* is the shared ancestor of *v* and *w* that is located farthest from the root. There are different ways to answer the query on an xml document; one commonly used method is LCA based method [1]. Many algorithms that use query over xml uses this method. Content nodes are the parent node of the keyword. For example consider keyword db in Fig.1 then content node of db is node 13 and node16.The server contains index structure of xml document which each node is letter in keyword and leaf node contain all nodes that contain the keyword this leaf node is called inverted list. Procedure**:**

- For keyword query the LCA based method retrieves content nodes in xml that are in inverted lists.
- Identify the LCAs of content nodes in inverted list.
- Takes the sub tree rooted at LCAs as answer to the query

For example suppose the user typed the query “www db” then the content nodes of db are{13,16} and for www are3 , the LCAs of these content nodes are nodes ,12,15,2,1.here the nodes 3,13,12,15 are more relevant answers but nodes 2 and 1 are not relevant answers.

Limitation

- It gives irrelevant answers.
- The results are not of high quality.

3.2.2 ELCA based method

To address the limitation of LCA based method exclusive LCA (ELCA)[4] is proposed. It states that an LCA is ELCA if it is still an LCA after excluding its LCA descendents. for example suppose the user typed the query “db tom” then the content nodes of db are {13, 16} and for tom are {14.17},the LCAs of these content nodes are nodes2,12,15,1.here the ELCAs are 12,15.the subtree rooted with these nodes is displayed which are relevant answers Node 2 is not an ELCA as it is not an LCA after excluding nodes 12 and 15.

3.3 Ranking the Subtree

There are two ranking function to compute rank/score between node n and keyword ki[2] 1) The case that n contains ki. 2) The case that n does not contain ki but has a descendant containing ki.

Case 1: n contains keyword ki
The relevance/score of node n and keyword ki is computed by

$$\mathrm{SCORE}_1(n,k_i)=\frac{ln\left(1+tf(k_i,n)\right)*ln\left(idf(k_i)\right)}{(1-s)+s*ntl(n)}.$$

Where: tf(ki,n) - no:of occurences of ki in subtree rooted n, idf(ki) - ratio of no. of nodes in xml to no:of nodes that contain keyword ki. ntl(n) - length of n /nmax length, nmax=node with max terms

s - Constant set to 0.2 Assume users composed a query containing keyword “db”

Case 2: node n does not contain keyword ki but its descendant has ki. Second ranking function to compute the score between n and kj is

$$\mathrm{SCORE}_2(n,k_j)=\sum_{p\in P}\alpha^{\delta(n,p)}*\mathrm{SCORE}_1(p,k_j),$$

Where P - Set of pivotal nodes, α - constant set to 0.8, $\delta(n,p)$ - Distance between n and p Assume the user composed query “db”
Score2 (12, db) = (0.8)*score1 (13, db)
= 0.8 *1.52 =1.21

3.3.1 Ranking fuzzy search

Given a keyword query Q={k1,k2,…..kl} in terms of fuzzy search, a minimal-cost tree may not contain the exact input keywords, but contain predicted words for each keyword.Let predicted words be {w1,w2…..wl}the best similar prefix of wi could be considered to be most similar to ki.The function to quantify the similarity between ki and wi is

$$sim(k_i,w_i)=\gamma*\frac{1}{1+\mathrm{ed}(k_i,a_i)^2}+(1-\gamma)*\frac{|a_i|}{|w_i|},$$

where ed – edit distance, ai – prefix, wi – predicted word, γ – constant

3.4 Progressively Compute Top K Answers

The index structure is used to compute the answers. The leaf node inverted list contains the content nodes and quasi contend nodes, scores of the keyword. For computing top k results heap based method [3] is used which uses the partial virtual inverted lists which contain the higher score nodes so to avoid the union of lists which is expensive.

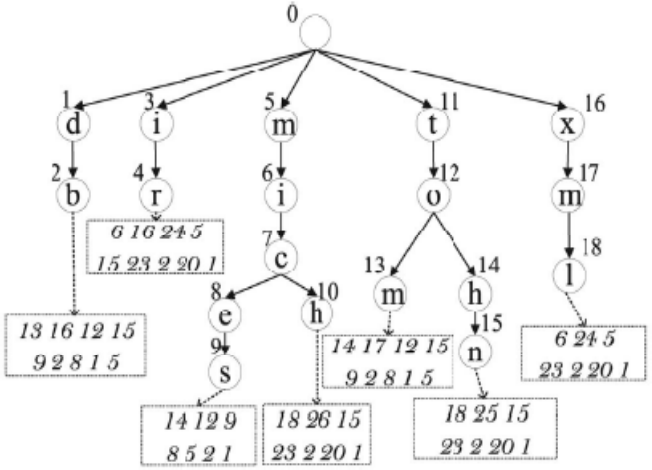


Figure 3: extended tree structure Procedure

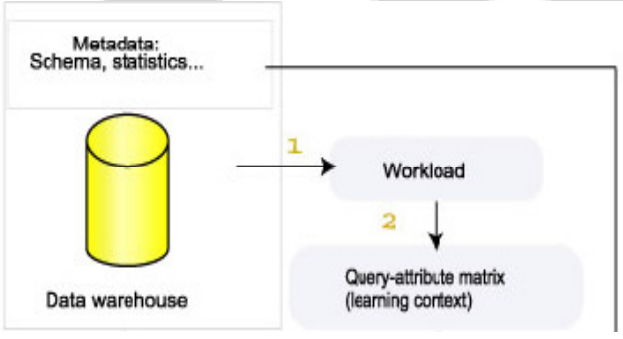
Sort the scores in the inverted lists;

1) If the inverted list is long the partial virtual inverted list
2) Construct max heap, such that each node contain <node, score>
3) The top element of max heap is highest score node and is deleted, max heap is adjusted
4) Deleted node with score<=T (threshold) are taken into result set and return the result set if the top –k answers are retrieved.

For example assume user composed the query "db". The inverted list of db contains the nodes 13,16,12,15,9, 2,8,1,5.The scores of these nodes computed by two ranking functions are 1.52,1. 52,1.21, 1.21,0. 9728,0, 495,0.77, 0,396,0.6225 respectively. These scores have to be sorted and max heap is constructed and a threshold is fixed be 10 so the top elements< (13, 1.52>, <16, 1.52>, <12, 1.21>, <15, 1.21> the top e results are retrieved. This technique is more efficient and effective.

3.5 Xml Materialized View Selection Strategy

The architecture of our materialized view selection strategy is depicted in Figure 4. We assume that we have a workload composed of representative queries for which we want to select a configuration of materialized views in order to reduce their execution time. The first step is to build, from the workload, a clustering context. Then we define similarity and dissimilarity measures that help clustering together similar queries.



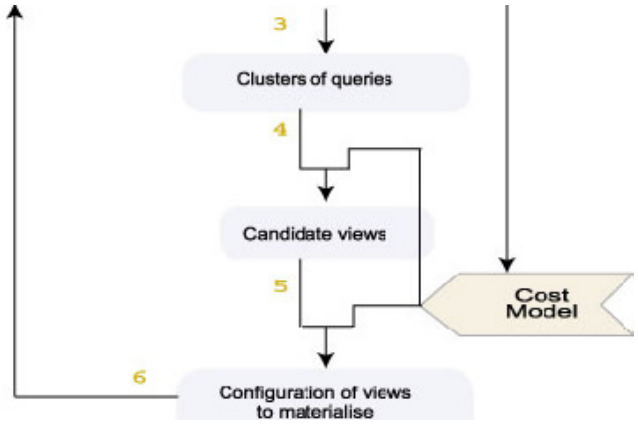


Figure 4: Materialized view selection strategy

Where:

1. Extraction of set of queries resolved by the system
2. Extraction of the representative attributes from the queries.
3. Application of a clustering algorithm to create clusters of queries
4. Generation the set of candidate views
5. Selection of the final view configuration
6. Materialization of final view

For each cluster, we build a set of candidate views. The last step exploits cost models that evaluate the cost of accessing data using views and the cost of their storage to build a final materialized view configuration.

3.5.1 Query workload analysis

The workload that consider is a set of *selection*, *join* and *aggregation* queries. The first step consists in extracting from the workload the representative attributes for each query. The representative attributes those are present in *Where* (selection predicate attributes) and *Group by* clauses. Let store the relationships between the query workload and the extracted attributes in a "*query-attribute*" matrix. The matrix lines are the queries and the columns are the extracted attributes. A query *i q* is then seen as a line in the matrix that is composed of cells corresponding to representative attributes. The general term *ij q* of this matrix is set to one if extracted attribute *i a* is present in query *i q,* and to zero otherwise. This matrix represents our clustering context.

3.5.2 Building the Candidate View Configuration

In practice, it is hard to search all the syntactically relevant views (candidate views) because the search space is very large. To reduce the size of this space, we propose to cluster the queries. Hence, we group in a same cluster all the queries that are similar. Similar queries are the one having a close binary representation in the query-attribute matrix. Two similar queries can be resolved by using only one materialized view. It can define similarity and dissimilarity measures that ensure that queries within a same cluster are strongly related to each other's whereas queries from different clusters are significantly different.

3.5.3 Cost Models

The number of candidate views is generally as high as the input workload is large. Thus, it is not feasible to materialize all the proposed views because of storage space constraints.

To circumvent this limitation, we propose to use cost models allowing keeping only the most pertinent views.

4. Conclusion

This paper presents the keyword search over the xml data which is user-friendly and there is no need for the user to study about the xml data .This paradigm gives the relevant results the user wants. Fuzzy search over xml data is studied which gives approximate results. Various methods for querying on xml data LCA based method, ELCA, heap based method are presented and of all these methods heap based method gives high quality results. To further improve the search performance, also addresses an open problem of answering XML keyword search using materialized views.

References

[1] M.D. Atkinson, J.-R. Sack, N. Santoro, and T. Strothotte, "Min-max Heaps and Generalized Priority Queues," Comm. ACM, vol. 29, no. 10, pp. 996-1000, 1986.
[2] 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.
[3] H. Bast and I. Weber, "Type Less, Find More: Fast Auto completion Search with a Succinct Index," Proc. Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR), pp. 364-371, 2006.
[4] H. Bast and I. Weber, "The Complete search Engine: Interactive, Efficient, and towards Ir&db Integration," Proc. Biennial Conf. Innovative Data Systems Research (CIDR), pp. 88-95, 2007.
[5] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword Searching and Browsing in Databases Using Banks," Proc. Int'l Conf. Data Eng. (ICDE), pp. 431-440, 2002.
[6] Y. Chen, W. Wang, Z. Liu, and X. Lin, "Keyword Search on Structured and Semi-Structured Data," Proc. ACM
[7] SIGMOD Int'l Conf. Management of Data, pp. 1005-1010, 2009.
[8] 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.
[9] B.B. Dalvi, M. Kshirsagar, and S. Sudarshan, "Keyword Search on External Memory Data Graphs," Proc. Int'l Conf. Very Large Data Bases (VLDB), pp. 1189-1204, 2008.
[10] B. Ding, J.X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding Top-k Min-Cost Connected Trees in Databases," Proc. Int'l Conf. Data Eng. (ICDE), pp. 836-845, 2007.
[11] R. Fagin, A. Lotem, and M. Naor, "Optimal Aggregation Algorithms for Middleware," Proc. ACM SIGMOD-SIGACTSIGART Symp. Principles of Database Systems (PODS), 2001.
[12] Baralis E., Paraboschi S., Teniente E., Materialized View Selection in a Multidimensional Database, VLDB 1997.
[13] Yang J., Karlapalem K., Li Q., Algorithms For Materialized View Design in Data Warehousing Environment. VLDB 1997.

Author Profile

Supriya Chaudhari completed B.E. Computer Science & Engineering from Jawaharlal Darda Institute of Engineering & Technology, Yavatmal, Maharashtra, in 2006. She is pursuing ME in Computer Science & Engineering from Sant Gadge Baba University Amravati, Maharashtra.