Volume 6, Issue 3, March 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Proximity Ranking Based Fast Fuzzy Search

Shraddha Khandade
Department of Computer Engineering,
DYPCOE Talegaon, Pune, Maharashtra, India

Prof. Sandeep U. Kadam
HOD, Department of Computer Engineering,
DYPCOE Talegaon, Pune, Maharashtra, India

Abstract— In traditional keyword based search system over Xml data, user takes a query, submit it to the system and gets relevant answer. When user has limited knowledge about the data when issuing queries, and has to use a try and see approach for finding information. This paper focus on the fuzzy type-ahead search in XML data using proximity rank method. It is a new information access paradigm in which the system searches XML data on the fly as user type in query keyword. XML model capture more semantic information and navigates into document and display more relevant information. The keyword search is alternative method to search in XML data, which is easy to use, user doesn't need to know about the XML data and query language. This paper focus on the techniques used to retrieve the top-k result from the XML document more efficiently. It is also needed good ranking functions that consider the proximity of keywords to compute relevance scores. This paper focus on study how to integrate proximity information into ranking in instant-fuzzy search while achieving efficient time and space complexities.

Keywords— XML data, Top-k answering, Fuzzy search, Instant search, Proximity ranking

I. INTRODUCTION

A. XML Document

Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost. XML stores data in plain text format. This provides a software and hardware-independent way of storing, transporting, and sharing data.

XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data. With XML, data can be available to all kinds of "reading machines" like people, computers, voice machines, news feeds, etc. Figure 1 shows structure of XML data.

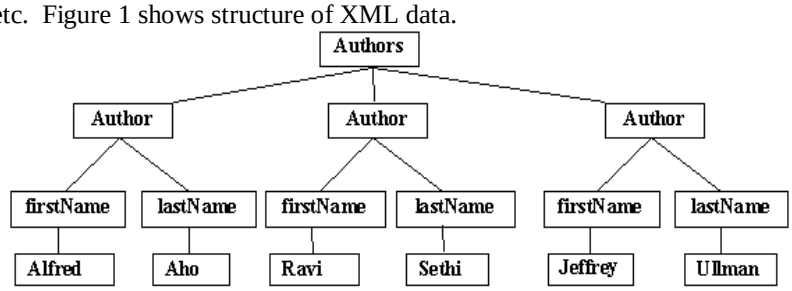

Fig. 1 XML data structure

B. Instant Search

As an emerging information-access paradigm, instant search returns the answers immediately based on a partial query a user has typed in. For example, the Internet Movie Database, IMDB, has a search interface that offers instant results to users while they are typing queries1. When a user types in "sere", the system returns answers such as "Serena", "Serenity", "Serendipity", and "Serena Williams". Many users prefer the experience of seeing the search results instantly and formulating their queries accordingly instead of being left in the dark until they hit the search button. The recent study showed that this new information-retrieval method helps users find their answers quickly with less effort.

C. Fuzzy Search

Users often make typographical mistakes in their search queries. Meanwhile, small keyboards on mobile devices, lack of caution, or limited knowledge about the data can also cause mistakes. In this case user cannot find relevant answers by finding records with keywords matching the query exactly. This problem can be solved by supporting fuzzy.

D. Proximity Ranking

Proximity measures the closeness or nearness of a keywords. The ranking to searched result is given depending on proximity. Proximity ranking will rank documents higher where the query words are found *close together .*

E. Top-k Query Answering

A positive integer k is given, system has to find the k most relevant answers to a query. One way to compute these results is to first find all the results matching the query conditions, and rank them based on their score.

To study fuzzy type-ahead search in XML data following contributions are made:

- Formalize the problem of fuzzy type-ahead search in XML data.
- Propose effective index structures and efficient algorithms to achieve a high interactive speed for fuzzy type-ahead search in XML data.
- Develop ranking functions and early termination techniques to progressively and efficiently identify top-k relevant answers.

II. EXISTING SYSTEM

In existing system the problem of fuzzy type-ahead search in XML data is studied. Two types of indexing are used as Forward index and Trie index. LCA-based method is used to interactively identify the predicted answers. As well as a minimal-cost-tree-based search method is developed to efficiently and progressively identify the most relevant answers. A forward-index structure is used to further improve search performance.

A. Minimal-Cost Tree

In this section, a framework is introduced to find relevant answers to a keyword query over an XML document. In the framework, each node on the XML tree is potentially relevant to the query with different scores. For each node, that system define its corresponding answer to the query as its subtree with paths to nodes that include the query keywords. This subtree is called the "minimal-cost tree" for this node. Different nodes correspond to different answers to the query, and it is studied how to quantify the relevance of each answer to the query for ranking.

III. KEYWORD SEARCH IN XML DATA

A. 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. Many algorithms that use query over xml uses this method. Content nodes are the parent node of the keyword. 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.

IV. INDEX STRUCTURES

An index structure called Inverted index is used for indexing. It is designed to allow very fast full-text searches. An inverted index consists of a list of all the unique words that appear in any document, and for each word, a list of the documents in which it appears.

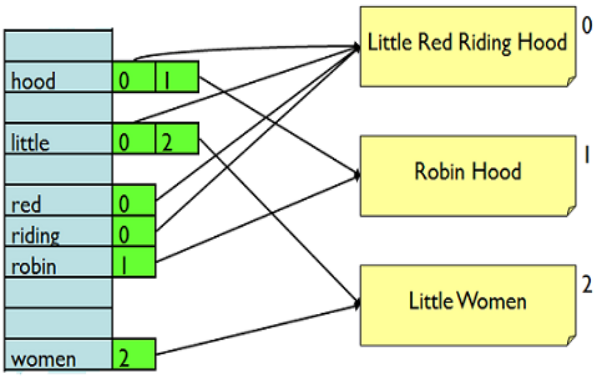

Fig 2: Structure of Inverted Index

Figure 2 shows structure of example of inverted index. The inverted index data structure is a central component of a typical search engine indexing algorithm. A goal of a search engine implementation is to optimize the speed of the query: find the documents where word X occurs. With the inverted index created, the query can now be resolved by jumping to the word id (via random access) in the inverted index.

V. RANKING

Modern web search engines have been exploiting more and more features to improve quality of search results. Document structure and term proximity are no doubt among the most important features. It is a common practice to

differentiate document fields to improve search results. Document fields which search engines considered most include title, URL, anchor text and body text. Term proximity means the distance relationship between query terms in a document. A document where query terms appear together is considered to be more relevant than another document in which query terms are far away from each other. The system takes a search query and returns a set of documents ranked by relevancy with documents most similar to the query having the highest score.

$score(q,d) = coord(q,d) \cdot queryNorm(q) \cdot \sum_{t \text{ in } q} (tf(t \text{ in } d) \cdot idf(t)^2 . t. getBoost() .norm(t, d))$

Where

score(q,d) is the relevance score of document d for query q

queryNorm = 1 / √sum Of Squared Weights

idf(t)= 1+log(numdocs/(docFreq+1))

if(t in d)=frequency $^{1/2}$

coord(q,d) is a score factor based on how many of the query terms are found in the specified document.

t.getBoost() is a search time boost of term t in the query q as specified in the query text

If keyword query is not found, Levenshtein distance is measured for fuzzy search.

VI. FUZZY SEARCH

FuzzyQuery matches terms "close" to a specified base term: specify an allowed maximum edit distance, and any terms within that edit distance from the base term (and, then, the docs containing those terms) are matched. The Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other.

Mathematically, the Levenshtein distance between two strings a,b (of length $|a|$ and $|b|$ respectively) is given by $lev_{a,b}(|a|, |b|)$ where

$Lev_{a,b}(i,j) = \max(i, j)$ ……. If $\min(i, j)=0$

$= \min \{ lev_{a,b}(i-1, j) + 1\}$

$= \min \{ lev_{a,b}(i, j-1) + 1\}$ otherwise

$= \min \{ lev_{a,b}(i-1, j-1) + 1_{(ai \neq bj}\}$

where $1_{(ai \neq bj}\}$ is the indicator function equal to 0 when $a_i = b_i$ and equal to 1 otherwise, and $Lev_{a,b}(i,j)$ is the distance between the first i characters of a and the first j characters of b. The first element in the minimum corresponds to deletion (from a to b), the second to insertion and the third to match or mismatch, depending on whether the respective symbols are the same.

For example, the Levenshtein distance between "kitten" and "sitting" is 3, since the following three edits change one into the other, and there is no way to do it with fewer than three edits:

1. kitten → sitten (substitution of "s" for "k")
2. sitten → sittin (substitution of "i" for "e")
3. sittin → sitting (insertion of "g" at the end).

Algorithm

This search system supports proximity search based on term proximity and supports exact phrase query search and proximity search with distances greater than 0(using the term slop to mean distance).

Algorithm for matching d (document) with q(query)

Input: a query $q=t_1...t_n$.

Output: Frequency of $t_1...t_n$ occurring in d.

for each $t_i \in q$ **do**

$p_d \leftarrow$ first position of $P_d(t_i)$; $\Delta_i \leftarrow p_d - pq_i$

end for

freq← 0; maxterm ← t_k where $\Delta_k = \max(\Delta_i)$

do

loop until $\min(\Delta_i) = \max(\Delta_i)$

do

minterm ← t_m where $\Delta_m = \min(\Delta_i)$

if next position of P_d(minterm) exists **then**

$p_d \leftarrow$ next position of Pd(minterm)

$\Delta_k \leftarrow p_d - pq_k$ /* update the value Δ of minterm */

else return freq **end if**

while $\Delta_k < \Delta_m$ /* Δ_m is the value Δ of maxterm */

end loop

freq←freq + 1

while

return freq

VII. EXPERIMENTAL RESULTS

Fig. 3 shows graph of search time in millisecond verses data size. The system searches the keyword within minimum time. It is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search.

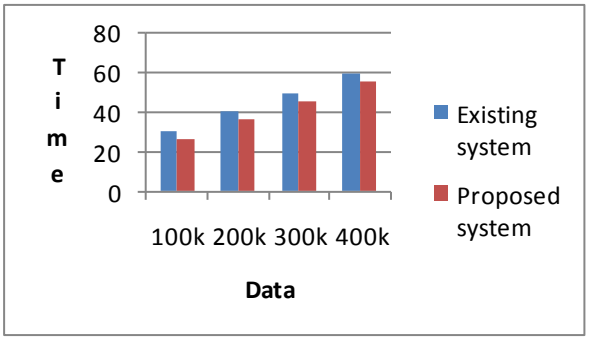

Fig. 3 Search time

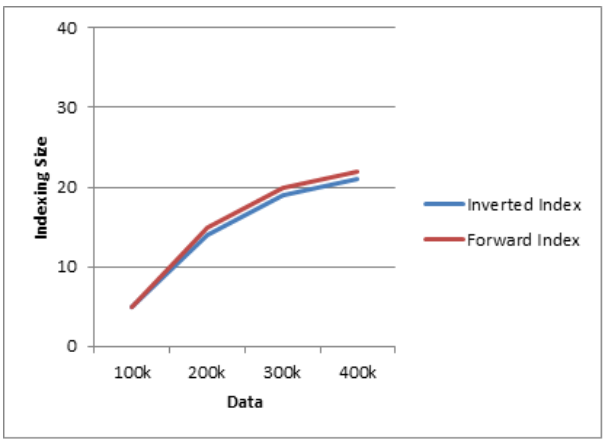

Fig. 4 Index size

The index stores statistics about terms in order to make term-based search more efficient. This system`s index falls into the family of indexes known as an *inverted index*. This is because it can list, for a term, the documents that contain it. This is the inverse of the natural relationship, in which documents list terms. Indexing means making data more suitable for fast search and making it small enough to search it fast.

VIII. CONCLUSION

The ranking of an instant-fuzzy search system is improved by considering proximity information when it is needed to compute top-k answers and to adapt existing solutions to solve this problem, including computing all answers, doing early termination, and indexing term pairs. A technique is proposed to index important phrases to avoid the large space overhead of indexing all word grams. An incremental computation algorithm is presented for finding the indexed phrases in a query efficiently, and studied how to compute and rank the segmentations consisting of the indexed phrases. The analysis is conducted by considering space, time, and relevancy tradeoffs of these approaches. In particular, the experiments on real data showed the efficiency of the proposed technique for keyword queries that are common in search applications. It is concluded that computing all the answers for the other queries would give the best performance and satisfy the high efficiency requirement of instant search.

REFERENCES

[1] Jianhua Feng and Guoliang Li," Efficient Fuzzy Type-Ahead Search in XML Data", Knowledge and Data Engineering, IEEE Transactions on (Volume: 24 , Issue: 5) 2012

[2] Hao Wu,: search as you type in forms "leveraging the usability and functionality of search paradigm in relational database", VLDB 2010, Ph D workshop, Singapore

[3] Inci, Jamshid,Tae woo Kimand Chen Li," Efficient Instant-Fuzzy Search with Proximity Ranking", Data Engineering (ICDE), 2014 IEEE 30th International conference.

[4] Mingjie Zhu, Shuming Shi, Mingjing Li, Ji-Rong Wen "Effective Top-K Computation in Retrieving Structured Documents with Term-Proximity Support", CIKM07,Copyright 2007 ACM

[5] S.Chaudhuri,V.Ganti, andR.Motwani,"Robust identication off uzzy duplicates", 2013 IEEE 29th International Conference on Data Engineering (ICDE)"

[6] G. Li, J. Wang, C. Li, and J. Feng, "Supporting efficient top-k queries in type-ahead search"' SIGIR12, August 1216, 2012, Portland, Oregon, USA.

[7] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram,"Xrank Ranked Keyword Search over Xml Documents", SIGMOD 2003 Copyright 2003 ACM

[8] S. Ji, G. Li, C. Li, and J. Feng, "Efficient interactive fuzzy keyword search", WWW 2009, April 2024, 2009, Madrid, Spain. ACM 978-1-60558-487-4/09/04.

[9] H. Bast and I. Weber, "Type Less, Find More: Fast Auto-completion Search with a Succinct Index", SIGIR06, August 611, 2006, Seattle, Washington, USA.

[10] Tao Tao Microsoft Corporation Redmond, Cheng Xiang Zha,,"An Exploration of Proximity Measures in Information Retrieval" SIGIR'07, July23–27,2007, Amsterdam, The Netherlands.

[11] Tao Tao Microsoft Corporation Redmond, Cheng Xiang Zha,,"An Exploration of Proximity Measures in Information Retrieval" SIGIR'07, July23–27,2007, Amsterdam, The Netherlands.

