Interactive Fuzzy based Search over XML Data for Optimized Performance

[1] Sushma. J. Basanagoudar, [2] Dr. B. G. Prasad

[1] PG Student, M. Tech (CSE), B.N.M Institute of Technology
Bangalore, Karnataka, India

[2] Computer Science & Engineering, B.N.M Institute of Technology
Bangalore, Karnataka, 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, TASX - 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. TASX provides friendly interface for users to explore XML data and can save users typing effort.

***Keywords* - XML, Keyword Search, Type-ahead Search**

1. Introduction

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. As XML is about to become the standard format for structured documents, there is an increasing need for appropriate information retrieval (IR) methods. XML provides a standard method to access information, making it easier for applications and devices of all kinds to use, store, transmit, and display data. XML has grown from a markup language for special purpose documents to a standard for interchange of heterogeneous data over Web, a common language for distributed computation, and universal data format to provide users with different views of data. All of these increase the volume of data encoded in XML, consequently increasing the need for database management support for XML documents. An essential concern is how to store and query potentially huge amounts of XML data efficiently. Traditional information systems allow users to compose and submit a query to retrieve relevant answers. This information-access paradigm requires the user to have certain knowledge about the structure and content of the underlying data repository. With limited knowledge about the data, a user often feels "left in the dark" when issuing queries, and the user has to use a try-and-see approach for finding information. For instance, Fig.1 shows a traditional interface to search on the book information. To find a book, a user needs to fill in the form by providing information for multiple attributes, such as book title, author, ISBN, publisher. If the user has limited information about the book she is looking for, such as the exact spelling of the book title, the user needs to try a few possible keywords, go through the returned results, modify the keywords, and reissue a new query. She needs to repeat this step multiple times to find the book, if lucky enough. This search interface is neither efficient nor user friendly.

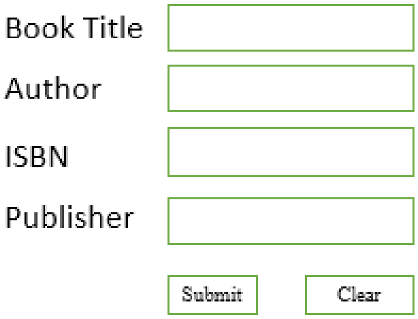

Fig.1 Traditional Book Search Form

Many systems are introducing various features to solve this problem. One of the commonly used method is Autocomplete, 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 multiple 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. Type-ahead search in relational databases is studied. 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. The proposed method TASX (pronounced “task”), a fuzzy Type-Ahead Search in XML data is used to search 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.

1.1 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 <= v” to denote that node u is an ancestor of node v. For example, consider the XML document in Fig. 2, we have paper (node5) <= author (node 7) which indicates node 5 is ancestor of node 7. A keyword query consists of a set of keywords {k1, k2,…., kl}. For each keyword k_i, we call the nodes in the tree that contain the keyword the content nodes for k_i. The ancestor nodes of the content nodes are called the quasi-content nodes of the keyword. For example, consider the XML document in Fig.2, title (node 16) is a content node for keyword “DB,” and conf (node 2) is a quasi-content node of keyword “DB.”

1.2 Problem Formulation

Given an XML document D, a keyword query Q={k1,k2,….kl} and an edit-distance threshold τ. TASK - a Fuzzy Type-Ahead Search method in XML data works for queries with multiple keywords by allowing mismatch of query keywords. It efficiently identifies the predicted words that have prefixes similar to input partial keywords after each keystroke from the user. TASX provides a friendly interface for users to explore XML data and can significantly save user’s typing effort.

2. Literature Review

Keyword search in XML data has attracted great attention recently. Xu and Papakonstantinou [1] proposed smallest lowest common ancestor (SLCA) to improve search efficiency. Sun et al. [2] studied multiway SLCA-based keyword search to enhance search performance. 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. Type-ahead search is a new topic to query relational databases. Li et al. [4] studied type-ahead search in relational databases, which allows searching on the underlying relational databases on the fly as users’ type in query keywords. Ji et al. [5] studied fuzzy type-ahead search on a set of documents, which can on the fly find relevant answers by allowing minor errors between input keywords and the underlying data. Koutrika et al. [6] 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. Chen et al. [7] gave an excellent tutorial of keyword search in XML data and relational databases.

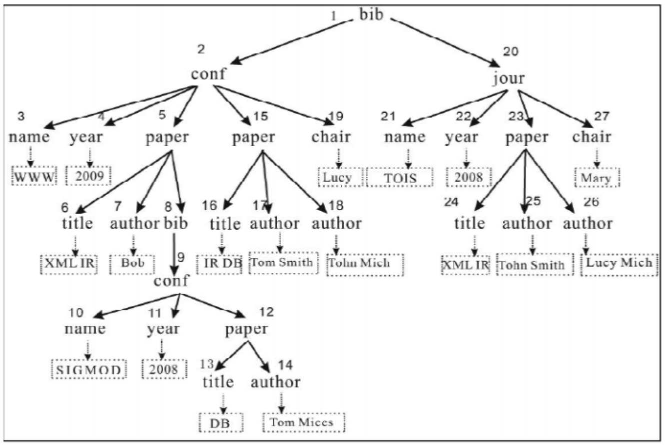

Fig. 2 An XML Document

3. System Analysis

3.1 Existing System

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 nondatabase 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. 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. User tries a few possible keywords, goes through the returned results, modifies the keywords, and reissues a new query. This search interface is neither efficient nor user friendly.

Disadvantages:

1. XQuery hard to understand for non-database users.
2. XQuery languages requires queries to be posed against underlying data scheme.
3. Keyword search does not support approximate search.
4. Keyword search interface is not efficient and user friendly.

3.2 Proposed System

The proposed method is TASX (pronounced “task”), a fuzzy Type-Ahead Search in XML data. TASX searches the XML data on the fly as user’s 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. 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. Interactive speed requires this delay to be within milliseconds. In proposed system, users explore data as they type, even in the presence of minor errors of their input keywords. 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. Existing methods cannot search XML data in a type-ahead search manner.

Proposed method has following features:

- ➢ Search as you type: It extends Autocomplete by supporting queries with multiple keywords in XML data.
- ➢ Fuzzy: It can find high quality answers that have keywords matching query approximately.
- ➢ Effective MCT (Minimal Cost Tree) method is implemented for retrieving answers from the XML document.

3.3 Proposed System Architecture

Fig. 3 shows the proposed architecture of the system. TASX works for multiple keyword queries in XML data, by allowing small errors of query keywords and inconsistencies in the data itself. Suppose there is an original XML document that resides on a server. A user accesses and searches the data through a web browser. Each keystroke that the user types invoke a query consists existing string. The browser sends the query to the server, which computes query answer. The underlying data residing is XML data. The client has a browser, using which a user can send query requests to the server to retrieve results. The client side contains HTML contents with JavaScript code executed in the browser. When the user types in a query, the JavaScript code issues an AJAX query to the server. Each keystroke of the user could invoke a query, which includes the current string the user has typed in. The browser sends the query to the server.

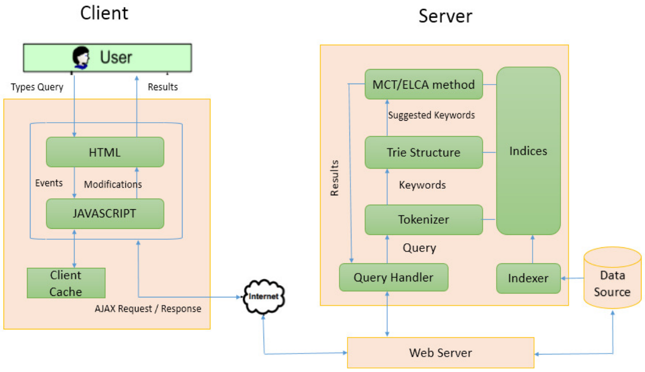

Fig. 3 Fuzzy Search System Architecture

The server tokenizes the query string, computes and returns to the user the best query answers. For each query, the server treats the last keyword as a partial keyword the user is completing, and other earlier keywords as complete keywords. For a keyword query, find the records that contain every complete keyword in the query and a keyword with the partial keyword as a prefix.Trie structure is used 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, store an inverted list of IDs of XML elements that contain the word of the leaf node. Supporting fuzzy search is very important especially when users do not remember the exact spelling of the right keywords. To support this feature, for each complete keyword in a query, identify the keywords in the data that are similar to the keyword. For the partial keyword, identify its similar keywords with a prefix similar to the partial keyword. Edit distance is used to quantify the similarity between keywords.

Two keywords are similar if their edit distance is within a given threshold τ. Next, server identifies the related subtrees in XML data for every input keyword that contain the predicted words. By using ELCA (Exclusive Lowest Common Ancestor) / MCT (Minimum Cost Tree) semantics, users recognize the answer based on the predicted words. These related subtrees are the predicted answers for the query.

4. Implementation

The implementation involves mainly 3 modules and they are:

1. User Interface.
2. Document Upload.
3. Fuzzy Search.

4.1 User Interface

User page is created using GUI (Graphical User Interface), which will be the media to connect user with the server and through which client is able to request the server and server can respond to the client. Through this module, communication is established between client and server. Before client creation, user credential is verified by login page through user name and password entered by user. An interface is provided for new user creation through user registration page by taking all the important details like username, password, email id, location from the user. JSP (Java Server Pages) is used to create the GUI in the form of webpages.

4.2 Document Upload

Document uploading is the second module of the project. This module performs uploading the user data in the document format. Here authenticated user wants to upload his/her document he can upload that document. This particular document is saved in the form of XML data. For uploading user selects one document and browse the document in the webpage of user. Then start uploading selected document using uploading option in the webpage. Uploaded document is saved in user directory.

4.3 Fuzzy Search

TASX – Fuzzy type-ahead search in XML data works for queries with multiple keywords in XML data, by allowing minor errors of query keywords and inconsistencies in the data itself. A user accesses XML data 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.

Fuzzy search consists of four steps:

1. Parsing the XML document.
2. Indexing the XML document using trie data structure.
3. Finding predicted word using edit distance method. (Fuzzy).
4. Computing answers based on predicted words using MCT (Minimal Cost Tree) based method.

Parsing XML Document

Parsing the XML document means “reading” the XML file and getting its content according to the structure. DOM (Document Object Model) is used to parse XML document. It provides a structured representation of the document (a tree) and it defines a way that the structure can be accessed from programs so that they can change the document structure, style and content.

Trie Index

Trie index structure is used to index the words in the underlying XML data. Trie is an ordered multi-way tree data structure that is used to store strings over an alphabet. Each node in a tree contains an array of pointers, one pointer for each character in the alphabet and all the descendants of a node have a common prefix of the string associated with that node. The root is associated with the empty string. The term trie comes from "retrieval”. Due to this etymology it is pronounced [tri] ("tree").

For instance, consider the XML document in Fig. 2. The tries structure for the tokenized words is shown in Fig. 4. The word “mich” has a node ID of 10. Its inverted list includes XML elements 18 and 26.

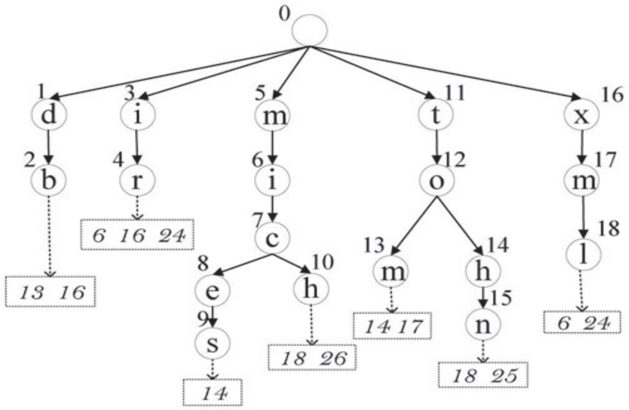

Fig. 4 The trie on top of words in Fig 2 (a part of words)

Edit distance method

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 is given by

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Where $1(a_i \neq b_j)$ is the indicator function equal to 0 when $a_i = b_j$ and equal to 1 otherwise.

Computing answers for multiple keywords

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. For example consider keyword db in Fig.2 then content node of db is node 13 and node16. 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 “DB Tom” on XML document in Fig.2. The content nodes of “DB” are {13,16} and for “Tom” are {14,17}. The LCAs of these content nodes are nodes 2, 12 and 15. Here the node 2 is less relevant to query than nodes 12 and 15 as nodes 13 and 17 corresponds to a values of different papers.

Limitation

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

ELCA based method:

To address the limitation of LCA based method exclusive LCA (ELCA) is proposed. It states that an LCA is ELCA if it is still an LCA after excluding its LCA descendants. For example suppose the user typed the query “DB Tom” on the XML document in Fig.2, then the content nodes of “DB” are {13, 16} and for “Tom” are {14,17}, the LCAs of these content nodes are nodes 2, 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.

Limitation

Use of “AND” semantics between input keywords of query and ignore answers that contain some of query keywords. For example, suppose a user types in a keyword query “DB IR Tom” on the XML document in Fig. 2. 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.

MCT (Minimal Cost Tree) method:

MCT method is used to find relevant answers to a keyword query over an XML document. In the MCT framework, each node on the XML tree is potentially relevant to the query. For each node corresponding answer to a query is defined as its subtree with the paths to the nodes that include the query keywords. This subtree is called the “minimal cost tree” for this node. For each leaf node in the trie, indexing is done both for content and quasi-content nodes whose descendants contain the keyword. For instance, consider the XML document in Fig. 2. For the keyword “DB,” index nodes are 13, 16, 12, 15, 9, 2, 8, 1, and 5 as shown in Fig. 5 which shows the extended trie structure. For the keyword “IR,” index nodes are 6, 16, 24, 5, 15, 23, 2, 20, and 1. For the keyword “Tom,” index nodes are 14, 17, 12, 15, 9, 2, 8, 1, and 5.

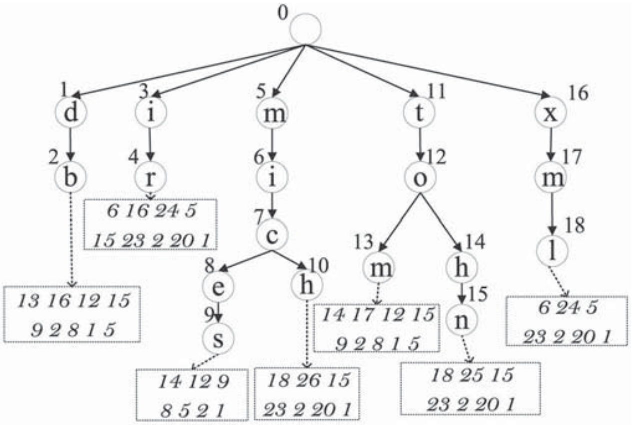

Fig. 5 The Extended trie on top of words in Fig. 2(a part of words)

The main advantage of MCT is that, even if a node does not have descendant nodes that include all the keywords in the query, this node could still be considered as a potential answer.

Consider the XML document in Fig.2 and given a keyword query Q = {DB, Tom, WWW}. Nodes 3, 13, 14, 16, and 17 are content nodes of the three keywords. Nodes 1, 2, 5, 8, 9, 12, and 15 are their quasi-content nodes. Node 3 is the pivotal node for node 2 and “WWW”. Node 16 is the pivotal node for node 2 and “DB”. Node 17 is the pivotal node for node 2 and “Tom”. The MCT of node 2 is the subtree rooted at node 2, which contains the paths:
$n_2 \rightarrow n_3$, $n_2 \rightarrow n_{15} \rightarrow n_{16}$, and $n_2 \rightarrow n_{15} \rightarrow n_{17}$.

5. Results

We have implemented our method using propose techniques on sample data set. We set up a server using Apache Tomcat server. The server was running program implemented in Java programming language. We used Ajax and JavaScript to allow client browser to interact with the server and display the result. We conducted an evaluation on a PC running Windows 7 operating system. Table 1, Table 2, Table 3 shows the exact query and its typed query with Edit Distance 1, 2, 3 respectively.

Table 1 Selected Queries on Books.xml file with Edit distance = 1

Exact Query	Typed Query
Randall Carla	ranall crla
Khnorr Crawlies	Korr crawhies
Kress Peter	Kiss pete
Microsoft Programming	Microsof programing
Comprehensive interfaces	Comprehensive interhaces

Table 2 Selected Queries on Books.xml file with Edit distance = 2

Exact Query	Typed Query
Randall Carla	raiall crl
Khnorr Crawlies	Kor crakhies
Kress Peter	Kss pehe
Microsoft Programming	Microof prograzing
Comprehensive interfaces	Comrehensive inerhaces

Table 3 Selected Queries on Books.xml file with Edit distance = 3

Exact Query	Typed Query
Randall Carla	riall crl
Khnorr Crawlies	Khn crakhie
Kress Peter	Krhhs pehes
Microsoft Programming	Micros prograzing
Comprehensive interfaces	Comrehensiv inerhacek

Fig.6 shows the comparison graph for ELCA and MCT method. The comparison is made based on the number of results found for a given query keywords with the edit distance = 3. The graph predicts that MCT method result count is comparatively more than ELCA method.

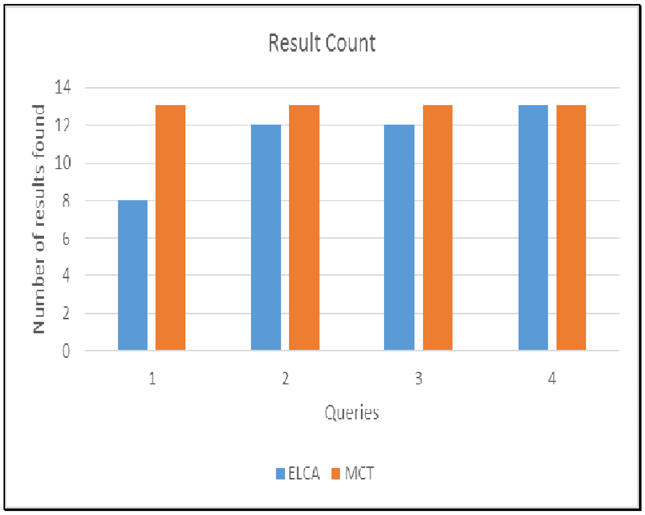

Fig. 6 Comparision of result count with Edit distance = 3

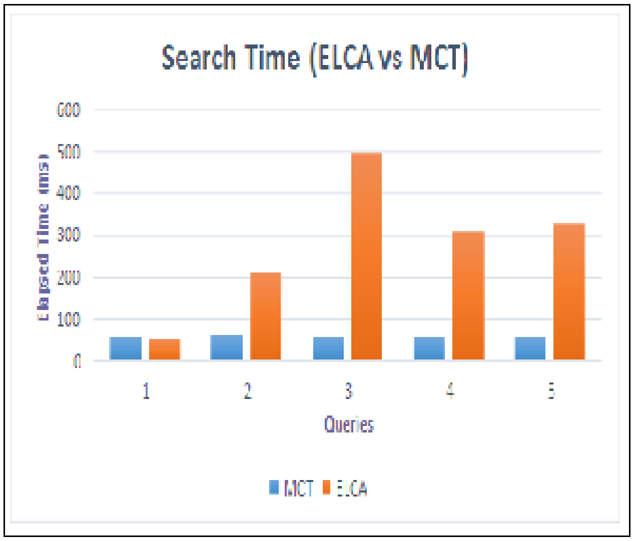

Fig. 7 Search Time (ELCA versus MCT)

Fig.7 shows the comparison graph for ELCA and MCT method. The comparison is made based on the search time. We used the queries on Books.xml file. The graph predicts that the MCT method takes less time compared to ELCA method for fuzzy search.

6. Conclusion and Future Enhancement

The problem of fuzzy type-ahead search in XML data, a new information-access paradigm in which system searches XML data on the fly as the user types in query keywords is implemented. This method provides approximate keyword search for users. In fuzzy type-ahead search method the predicted keywords are calculated using edit distance method and relevant answers among predicted keywords are retrieved using MCT (Minimal Cost Tree) method. MCT-based search method effectively and progressively identifies the most relevant answers.

The fuzzy type ahead search method provides a friendly interface for users to explore XML data and saves users typing effort. The comparison is done between the ELCA and MCT based method on search efficiency and result count of a query answer. The comparison based on search time predicts that MCT based search method is better than ELCA based method and achieves higher performance in terms of fuzzy search. The comparison based on result count shows that MCT based method is more effective than ELCA based method.

Following are some of the enhancements that can be made for improving our work:

- Progressively finding Top-k results.
- Improvement using Forward Index.

References

[1] Y. Xu and Y. Papakonstantinou, "Efficient Keyword Search for Smallest Lcas in XML Databases," Proc. ACM SIGMOD Int'l Conf.Management of Data, pp. 537-538, 2005.

[2] C. Sun, C.Y. Chan, and A.K. Goenka, "Multiway Slca-Based Keyword Search in Xml Data," Proc. Int'l Conf. World Wide Web (WWW), pp. 1043-1052, 2007.

[3] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv, "Xsearch" A Semantic Search Engine for Xml," Proc. Int'l Conf. Very Large Data Bases (VLDB), pp. 45-56, 2003.

[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] 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.

[6] 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.

[7] 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.

Sushma. J. Basanagoudar completed B.E in Computer Science & Engineering from S.J.C.I.T college in the year 2011, M. Tech (pursuing) Computer Science & Engineering in B.N.M Institute of Technology in the year 2014.

Dr. B. G. Prasad currently working as Professor & Head of Department in CSE at B.N.M Institute of Technology. He has over 25 years of academic experience in Computer Science. Has published 24 journal/conference papers. His areas of interest encompass Computer Networks, Image Processing, CBIR, Operating System and Computer Vision.