USING CTREE INDEX FOR PROXIMITY SEARCH IN XML DOCUMENTS

Swapna JawariKapisha[1] , G. Vijaya Lakshmi[2]
[1]Vikrama Simhapuri University, Nellore, Andhra Pradesh.

Abstract : Proximity Keyword Search is especially useful when searching on the web and in long unstructured documents such as XML. This system is designed to handle novel features of Proximity Keyword Search in XML documents. It concentrates mainly on producing ranked results efficiently for keyword search queries over XML documents. The proposed system is first of its kind in which the keyword string is pre processed before searching the XML document. This system eliminates the stop words and spaces entered by the user before locating the elements which contain the keywords. The search is case insensitive. In particular, this system is implemented in two stages. In pre processing stage, a set of keyword indices are built using CTREE concept for a set of XML documents. In the searching phase, the keywords entered by the user are analyzed and searched. Lowest common ancestor of the given keywords is computed and the results are ranked based upon the distance between the keywords located.

***IndexTerms* - CTREE, Indexing, Keyword Proximity Search, Minimal Connecting Trees, XML.**

I. INTRODUCTION

Semantic and keyword web based technique is becoming a generic issue in an application of Information Retrieval (IR). Most of the researchers used different web techniques for finding relevant information and find the keyword based search, which are not able to fetch the relevant search result because they do not know the actual meaning of the term or expression and relationship between them in the web search. HTML, Hypertext Markup Language (HTML) [1] is the standard markup language for creating web pages and web applications. Most documents on the web are currently stored and transmitted in HTML. One of the strengths of HTML is its simplicity, allowing it to be used by a wide variety of users. However, its simplicity is arguably is one of its weaknesses, with the growing need of users who want to create their tags to simplify their own tasks. In an attempt to satisfy this demand, W3C has produced a standard called the eXtensible Markup Language (XML), which could preserve the general application independence that makes HTML portable and powerful and adds many more new features.

XML[2] is a restricted version of SGML (Standard Generalized Markup Language), designed especially for Web documents. For example, XML supports links that point to multiple documents, as opposed to an HTML link that can reference just one destination document. XML is a format for representing semi structured data, since it allows more flexibility by not constraining to single structure. XML is designed to describe data on the web, basically Internet. XML allows us to define our own tags. XML used DTD (Document Type Definition) or XML Schema to describe the structure of the data. XML with a DTD or XML schema is self descriptive. XML is a W3C recommendation. XML is not a replacement for HTML (Hyper Text MarkUp Language). HTML is designed to describe the presentation of the content, while XML is designed to describe the content. As said before, XML allows the user to define his own document structure. Every starting tag needs an ending tag. Hence XML is strictly tag matching, unlike HTML.

The difference between text database and XML database results in three new challenges: 1) Identify the user search intention, i.e., identify the XML node types that user wants to search for and search via. 2) Resolve keyword ambiguity problems: a keyword can appear as both a tag name and a text value of some node; a keyword can appear as the text values of different XML node types and carry different meanings; a keyword can appear as the tag name of different XML node types with different meanings. 3) As the search results are subtrees of the XML document, new scoring function is needed to estimate its relevance to a given query. However, existing methods cannot resolve these challenges, thus return low result quality in term of query relevance.

Keyword search [3] is gaining popularity for querying XML data now days as it relieves user from understanding the complex schemas of XML document and query languages such as XQuery and XPath. Various query processing techniques and efficient algorithms have been proposed in recent days to address the keyword search over XML data.

Finding several terms that are close to one another is a way to make the search results more relevant, i.e. make the search more semantic. This feature is called Proximity Search ; it's especially useful when searching on the web and in long, unstructured documents A familiar example is to search for the word manage close to the word people, to find boss of those who have managed people, vs. profiles that just have both words somewhere in the text. Another example would be to look for a school name close to the year of graduation. Applications of proximity search are multiple. Standard full-text search with TF/IDF treats documents, or at least each field within a document, as a big bag of words. The match query can tell us whether that bag contains our search terms, but that is only part of the story. It can't tell us anything about the relationship between words.
Consider the difference between these sentences:
"Sue ate the alligator."
"The alligator ate Sue."
"Sue never goes anywhere without her alligator-skin purse."
A match query for sue alligator would match all three documents, but it doesn't tell us whether the two words form part of the same idea, or even the same paragraph. Understanding how words relate to each other is a complicated problem, and we can't solve it by just using another type of query, but we can at least find words that appear to be related because they appear near each other or even right next to each other. Each document may be much longer than the examples we have presented: Sue and

alligator may be separated by paragraphs of other text. Perhaps we still want to return these documents in which the words are widely separated, but we want to give documents in which the words are close together a higher relevance score. This relevance we can term as proximity of search terms.

In this paper, we propose a system that transforms XML documents of any organization into Ctree[4]. With the help of Ctree an index is built on all words present in the documents. It provides an interface which assists user of this system to search keywords in the XML documents. The keywords submitted by the user are analyzed by filtering out the spaces, tabs, stop words and further the keywords are converted into lower case. The algorithm locates the elements which contains the keywords from the Ctree Index table. After locating the elements, with the help of other entries of the index table such s groups, parent elements, lowest common ancestor of the keywords is located. Edge Distance is measured from the lowest common ancestor to elements which contain the keywords is computed. Score is assigned to each XML document based upon the number of keywords matched in the document. Finally based on the score and edge distance, the lowest common ancestor of the keywords with edge distance is displayed. The remaining of the paper is arranged as follows: Section 2 present a literature review of similar work conducted by different scholars; Section 3 present in details the concept of CTree and its usage for searching the keywords. ; In section 4, a proposed framework is described and how it will operate. Section 5 finalize with conclusion and then give the direction of the future work of the study.

II. LITERATURE REVIEW

In study number [5], the authors presents a novel method to find top-k answers in a node proximity search based on the well-known measure, Personalized PageRank (PPR). First, they deduct a distribution state transition graph (DSTG) to depict iterative steps for solving the PPR equation. Second, they proposed a weight distribution model of a DSTG to capture the states of intermediate PPR scores and their distribution. Using a DSTG, they selectively followed and compared multiple random paths with different lengths to find the most promising nodes. The limitation of this work is that it cant be applied directly to XML documents.

A multilevel and closed-form computational framework for keyword optimization (MKOF) to support various keyword decisions was propsed in [6]. Based on this framework, they developed corresponding optimization strategies for keyword targeting, keyword assignment, and keyword grouping at different levels (e.g., market, campaign, and adgroup).

Furthermore, Justin et al in their study [7], describe a method for ranking over the graph structures. Ziyang Liu et al [8] present a study that composes atomic and intact query results driven by users' search targets. They addressed the problems to identify user search intention, making ranking schemes ineffective. Roko et al [9] proposes an entity based query segmentation (EBQS) method which interprets a user query as a list of keywords and/or named entities to resolve ambiguity between subtrees. Then, segment terms proximity scorer (STPS) that assigns relevance scores to XML fragments that contains query keywords is proposed. Fragments containing the keywords as interpreted by EBQS are assigned higher scores. Finally, an effective predicate identification algorithm (EPIA) which uses EBQS and STPS to return relevant predicates is introduced.

Yushan et al [10] presented a XML semantic weight-value structural model which can speculate the relationship of the keywords and the SLCA (Smallest Lowest Common Ancestor) nodes based on the characteristic of XML documents and the structural relationship of SLCA. This model calculates the discriminating degree and the describing degree of the nodes to the documents by structural analyzing of the SLCA nodes and the keywords, and it designs a keyword-based query result ranking under the SLCA structure.

III. MOTIVATION

The User is always interested in finding how closely the keywords are associated instead of where that keywords appeared in a list of XML documents. Though Vagelis at al.[11] proposed an idea which finds how closely the keywords are associated, it is a bit complicated. It doesn't display the resulting XML sub tress rank wise. Our idea uses efficient indexing which helps in computing the LCA with less complexity. It also displays the XML subtrees by ranking them based on edge distance. It processes the keywords entered by the user before searching. Fig 3.1 shows an Example XML subtree.

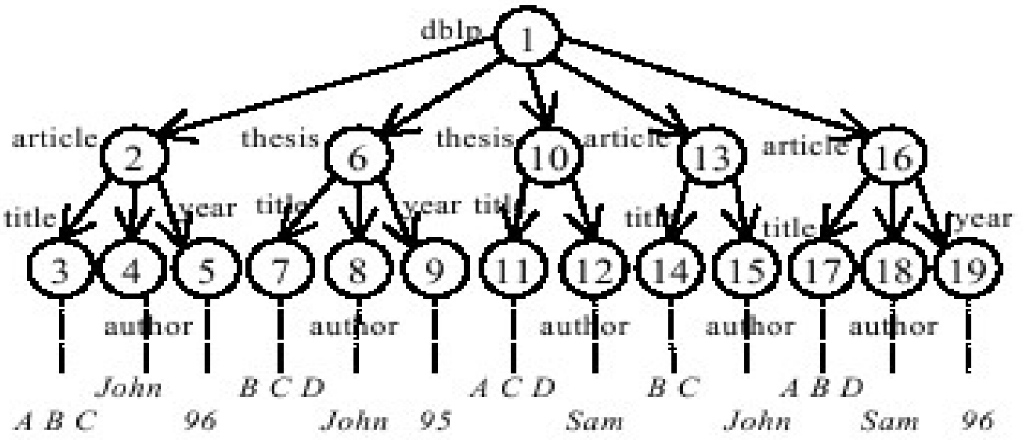


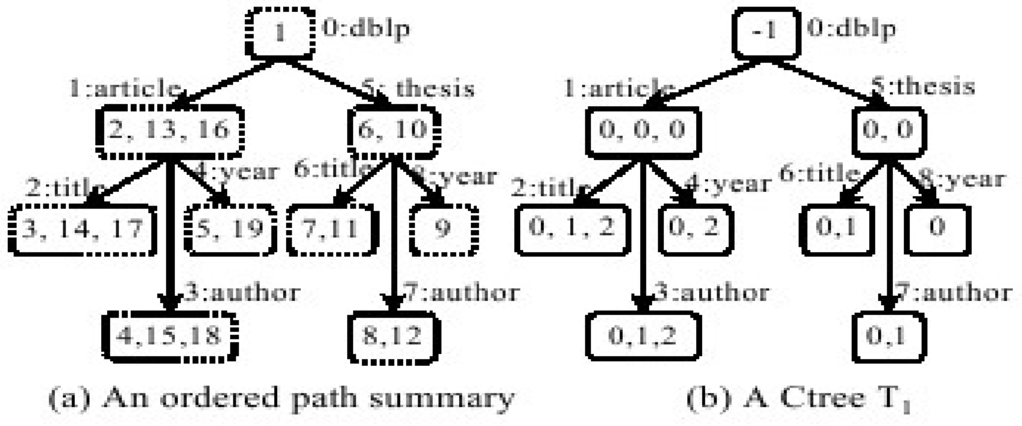
Fig 3.1 : Example XML Tree

3. 2 Ctree Based Indexing :

Ctree[8] is a two-level tree which provides a concise structure summary at its group level and detailed child-parent links at its element level which can provide fast access to element's parents. At the group level, Ctree provides a summarized view of hierarchical structures. At the element level, Ctree preserves detailed child-parent links. Each group in Ctree has an array mapping elements to their parents. We now define label path, equivalent nodes, Path Summary which helps in describing the Ctree.

LABEL PATH :A label path for a node v in an XML data tree D, denoted by L(v), is a sequence of dot separated labels of the nodes on the path from the root node to v. For example, node 8 in Figure 3.2 can be reached from the root node 1 through the path: 1-6-8. So label path for node 8 is dblp.thesis.author

EQUIVALENT NODES: Nodes in an XML data tree D are equivalent if they have the same label path. For example, nodes 8 and 12 in Figure 3.2 are equivalent since their label paths are the same dblp.thesis.author.



(a) An ordered path summary (b) A Ctree T_1

Figure 3.2 : Path Summary and its equivalent Ctree

Path Summary: Path Summary is a tree on which each node is called a group and corresponds to exactly one label path l in D. Path summary is called an ordered path summary if the equivalent nodes in every group are sorted by their pre-order identifiers. An ordered path summary for the XML data tree is shown in Figure 3.2. Each dotted box represents a group and the numbers in the box are the identifiers of equivalent data nodes. Each group has a label and an identifier listed above the group. For example, data nodes2,13,16 of XML document in Fig 3.3 are in group 1 since their label path are the same: dblp.article. Every data tree has a unique path summary. We now define Ctree as we undersood Path Summary.

Definition : A ctree is a rooted tree where each node g, called a group, contains an array of elements denoted as g.pid[] such that:

- Each group g is associated with an identifier and a name, denoted by g.id and g,name respectively.
- Edge directions are from root to the leaves. If there is an edge from g_1 to g_2 , then g_1 is called the parent of g_2 and g_2 is called a child of g_1. If there is a path from $g_1$to g_3, then g1 is called an ancestor of g_3 and g_3 is called a descendant of $g_{1.}$
- An array index k of g.pid[] reperesents an element in g, denoted by g:k. The value of g.pid[k] points to an element in g's parent g_p; and g_p:g.pid[k] is called the parent element of g:k
- For any two elements g:k_1 and g:k_2, if $k_1 < k_2$, then g.pid[k_1] <= g.pid[k_2].
- For example , Fig 3.2 (b) is sample Ctree. There is an array in each group. The array values are shown in the box separated by a comma. The array indexes are the positions of the values numbered starting from 0. The two elements in group 4(year) are referred by 4:0(first child of article element) and 4:1(second child of article element), whose values are 0 and 2 which are relative references.

3.3 Searching Keywords:

The Ctree index supports a search(word) operation. The search operation returns a list of absolute elements (when gid is not specified) or relative element (when the gid is specified). Since the inverted index is clustered by (wid,gid,eid), the operation serach (wid,gid) can be computed very efficiently once the value is mapped to a wid. Once we know the element id's and group id's where the keywords have occurred, we can use our LCA algorithm to find the Lowest Common Ancestor which connects the keywords.

The algorithm is as follows:

1. Find the group id's and element id's of the given keywords from the index table and store it in two lists.
2. If the group id's of all the keywords are same, check their element id's are equal.

(a) If they are equal – Display the element id along with the given keywords.

(b) If they are not equal – Compute the LCA of the keywords by retrieving their parent element ids and group ids.

Else

(a) Retrieve the depth of each keyword. Let p and q be the keywords which are at maximum depth and minimum depth respectively.

(b) Recursively reach to the ancestor of every keyword which is at level(q) from the keywords which have depth <= p.

3. Compute the LCA of the ancestors.

4. Rank the results based upon the distance between the keywords.

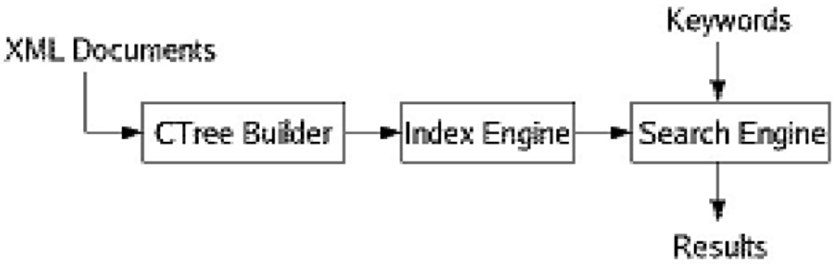


Figure 3.3 : Components of Proposed System

3.4 Proposed System :

The design of the proposed system is divided into three steps as shown in figure 3.3.

- Using Ctree based indexing to index the XML documents. This requires XML documents to be parsed and stored them in relational database in the form of tables. And building an index on this tabular data.
- The second major step is to efficiently use the Ctree index to compute the XML subtrees which contain all the keywords entered by the user.
- The final step is displaying the XML subtrees by ranking them based on edge distance from the Lowest Common Ancestor of the elements which contain the keywords.

3.5 Score of a XML Document:

In addition to distance between the keywords, a metric known as score is also computed for every XML document. Lets assume the user has submitted n keywords. If a XML document contains all n keywords, its score is defined as 100. With n keywords we can find n! Combinations. If a XML document contains less than n number of keywords say p, its score is defines as 100 - ((p/n!) * 100). For example, with 3 keywords, there are 6 possible combinations. Score of a XML document which contains all 3 keywords is 100 percent. Score for an XML document which contains 2 keywords is 100 -((2/6)*100).

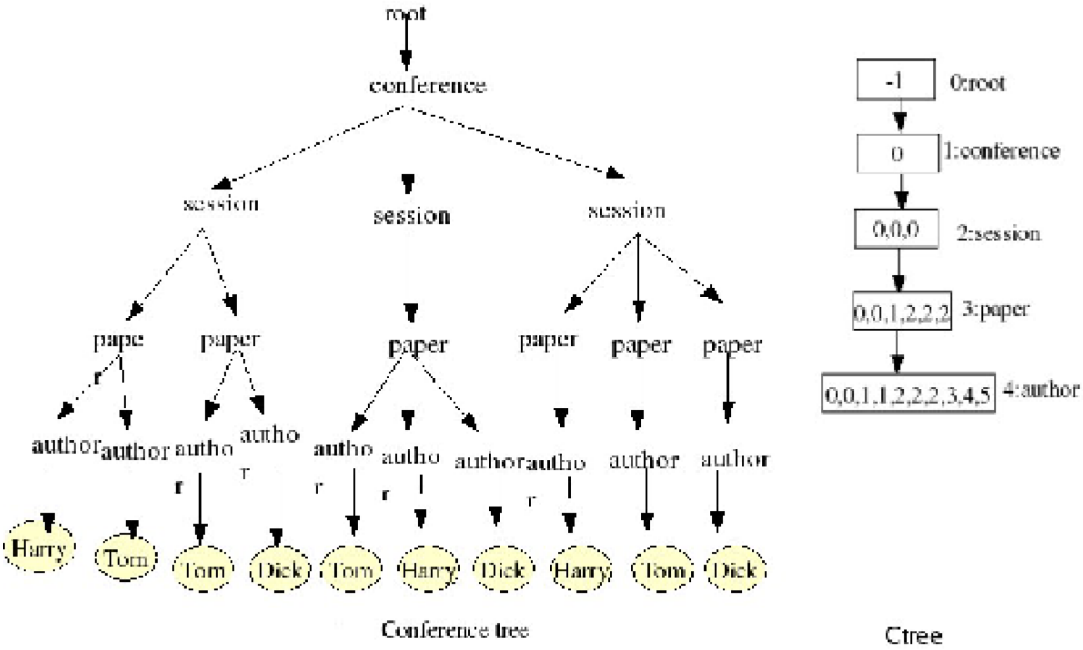


Figure 3.4 (a) Example Xml Document (b) Corresponding Ctree

3.6 Displaying the Results:

The LCA's which are computed for the given set of keywords are stored with the distance between the keywords from the LCA. Every subtree with LCA computed is stored. These subtrees are ranked and displayed. According to the typical assumption of keyword proximity systems smaller MCT's are considered better solutions since they provide a closer connection between the keywords. For example if the user submits the keywords Tom, Dick, Harry against the XML document of Figure 3.4, Figure 3.5 shows the possible minimum connecting trees.

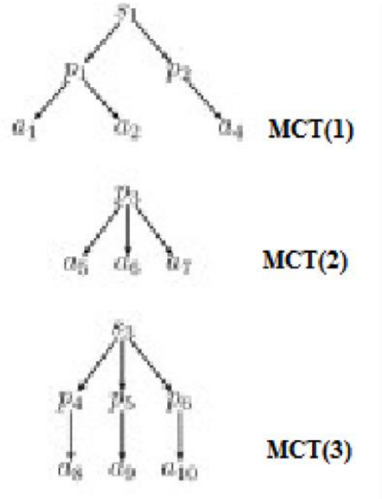


Figure 3.5 : Minimum Connecting Trees of Keywords Tom, Dick and Harry

IV. SYSTEM IMPLEMENTATION

The system is implemented in Java on a linux machine. SAX parser is used for parsing the XML document. JAVA API is used to process the XML documents and build the Ctree. Oracle Database is used to store the data in the tables. JavaScript is used to display the results graphically to the user. The entire system is implemented in four modules.

1. **dbase**: deals with establishing a connection with the database. 2

2. **indexing:** It contains the following three components.

- Analyzer : Helps in analyzing the given keywords by filtering out the white spaces, converting all upper case letters to lowercase letters, tokenizing the keyword strings and deleting the stop words.
- CTree: This deals with creation of necessary tables to build the database for the given XML documents. It creates the necessary tables such as Elements, groups, FileDetails, ElementPositions etc.
- Parser: This component parses the given XML documents and builds an index based on the content present in XML tags.

3. **init:** It configures the JBOSS with our application files

4. Search Engine: It takes input from the user, starts searching the keywords, ranks the distance between the keywords and displays the results.

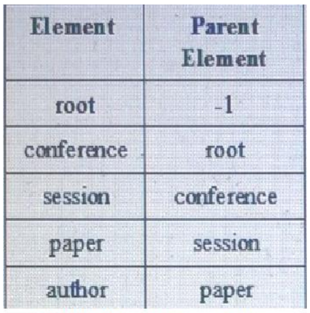
4. 1 Implementing Ctree:

Ctree index is mapped into four tables:

- Elements : It stores the mappings from elements to their parents
- Groups: It stores the group level tree by gid, subnum (the number of descendant groups) , level (the depth of the group), and pgid(parent group). It also stores the group name, and label path.
- The CtreeDB table contains one row for each Ctree including the Ctree name, the file group, the number of groups and elements.
- The ElmPosLen table records the position and length of each element, which is useful for retrieving the element.
- The invert table uses the table Words to map a word to an identifier (wid) which minimizes storage overhead by eliminating expensive string comparisons. The table Hits stores the occurrences and positions (pos) of words (wid) in XML elements (gid:eid). The XML files stores all the XML documents of the Ctree which are required if a user wants to look up the source of an element.

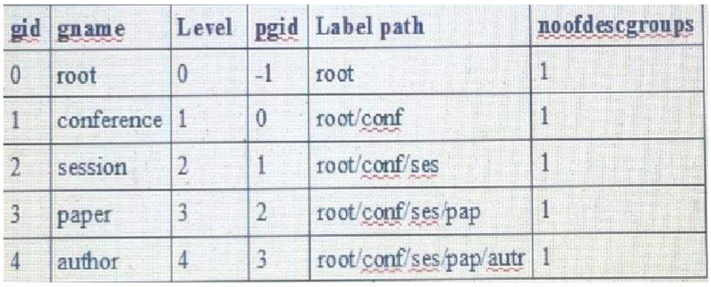
Tables 4.1, 4.2, 4.3, 4.4 shows values populated in Elements, Groups, ElmPosLen, Words tables when the example XML document Fig 3.4 is converted into Ctree. An inverted index is built on the words table based on keywords present in the XML data. This index returns a list of absolute elements and the group ids which contain keyword k_i. Since the Invert value index is clustered by (wid, gid, eid0, the operation search(wid,gid) can be computed very efficiently once the value is mapped to a wid.

4.1.2 Searching the keywords:



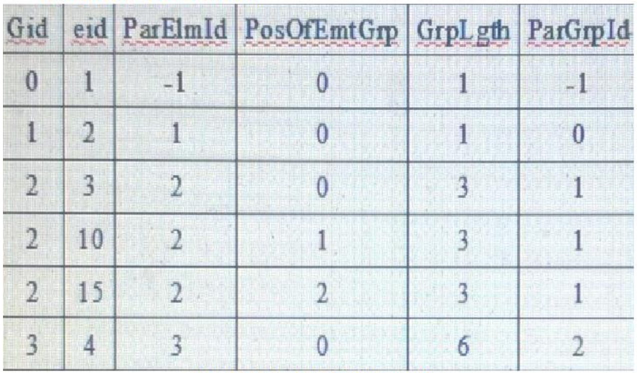
Element	Parent Element
root	-1
conference	root
session	conference
paper	session
author	paper

Table 4.1: Values in Elements Table



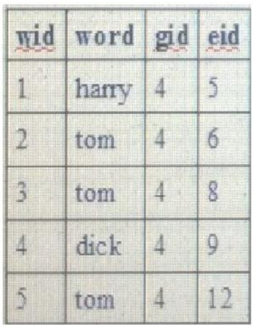
gid	gname	Level	pgid	Label path	noofdescgroups
0	root	0	-1	root	1
1	conference	1	0	root/conf	1
2	session	2	1	root/conf/ses	1
3	paper	3	2	root/conf/ses/pap	1
4	author	4	3	root/conf/ses/pap/autr	1

Table 4.2: Values in Groups Table



Gid	eid	ParElmId	PosOfEmtGrp	GrpLgth	ParGrpId
0	1	-1	0	1	-1
1	2	1	0	1	0
2	3	2	0	3	1
2	10	2	1	3	1
2	15	2	2	3	1
3	4	3	0	6	2

Table 4.3 : Snapshot of ElmPosLen Table



wid	word	gid	eid
1	harry	4	5
2	tom	4	6
3	tom	4	8
4	dick	4	9
5	tom	4	12

Table 4.4 : Snapshot of Values in Words Table

Suppose the user enters the keywords k1 and k2 in the search interface. From the index table retrieve the wid's where the keywords are occurred. From the list of wid's, retrieve gid and eid from the words table, from this list, retrieve the ParElmId and level from ElmPosLen table and groups table respectively. Now compare the ParElmId's of two keywords. If they are equal, then the element with the ParElmId is the LCA of the keywords. The distance from the LCA to these keywords is two. If the ParElmId's of the elements which contain the keywords are not equal, then check whether their levels are equal. If they are equal, retrieve the ParElmId's of the parents of the elements which contain the keyword. If they are equal, then we found the LCA with edge distance 4. If the levels are not equal, then recursively find out the ParElmId's until the level of the parElmId's become equivalent. Update the edge distance as we iterate to find out the LCA.

KEYWORD LIST

"**Tom**" Occurrences		
widgi	gid	eid
2	4	6
3	4	8
5	4	12
9	4	19

"**Harry**" Occurrences		
wid	gid	eid
1	4	5
6	4	13
8	4	17

Keyword Tom has occurred in group 4 four times with wid's 2,3,5,9. Keyword Harry has occurred in group 4 three times with wid's 1,6,8. Lets compute the LCA for word id's 2 and 1. wid 2 belongs to group 4 and is contained in element with eid is 6. wid 1 belongs to group 4 and is contained in element with eid is 5. ParElmId of element with eid 6 is 4. ParElmId of element with eid 5 is 4. Since both elements ParElmId's are equal, this is the LCA of keywords Tom and harry with edge distance is 2. Lets compute LCA for the id's 8 and 9. wid's 8 and 9 are occurred in elements with eid's 17 and 19 respectively. Their parElmId's are 16 and 18 respectively. Since they are not equal, retrieve at which level they have occurred and update the edge distance s 2. Both the elements are at same level. Now find out the parents of elements with eid's 17 and 19. ParElmId of 17 and 19 is 4. So add two to edge distance value. Element with eid 4 is the LCA of the keywords with edge distance 4. Keyword Tom has occurred 4 times while Harry has occurred 3 times in the document. So there are 12 possible LCA's. LCA's of all the possible combinations are calculated with edge distance. The LCA with least distance is displayed first.

4.1.3 Analyzing the keywords : When the user submits the keywords, all the white spaces between them are removed, and the keywords are checked with stopwords list and are removed. Besides this, all the symbols such as +, -, /, * are also filtered out.

4.1.4 Displaying the results: Results are displayed to the user graphically. Details such as field, fileName, group name, combination of search keywords, time taken to search are displayed to user. The user is also provided with the option of a link that will display how those keywords are related.

V. CONCLUSION AND FUTURE WORK

Unlike previous works, this work provides the distance analysis of the keywords. The entire XML document is stored in in-memory as the trees are stored in the form of Ctree. The Ctree index helps in efficiently computing LCA which is different than [9]. There is no need to maintain separate index files unlike previous approaches. In future work, we expect to compute lowest common ancestors for all the given keywords. It can be extended to compute the LCA of any number of keywords by sorting the parent element ids which contains keywords. Index updation must be taken care. It can be extended to implement grouping similar minimum connecting tress such as isomorphic trees, filtering out redundant trees.

REFERENCES

[1] Available at http://www.w3schoold.com/html

[2] Available at http://www.w3.org/Consortium/XML

[3] Zhifeng Bao , Jiaheng Lu , Tok Wang Ling , Bo Chen, ***"Towards an Effective XML Keyword Search"***, IEEE Transactions on Knowledge and Data Engineering (Volume: 22 , Issue: 8 , Aug. 2010),Page(s): 1077 - 1092

[4] Qinghua Zou, Shaorong Liu, Welsley W.Chu, “Ctree: A Compact Tree for Indexing XML Data”, in WIDM 2004.

[5] [Yanwu Yang , Bernard J. Jansen , Yinghui Yang , Xunhua Guo , Daniel Zeng, ***"Keyword Optimization in Sponsored Search Advertising: A Multilevel Computational Framework***", IEEE Intelligent Systems (Volume: 34 , Issue: 1 , Jan.-Feb. 1 2019)

[6] [6] Vajenti Mala , D. K. Lobiyal , ***"Semantic and keyword based web techniques in information retrieval"***, 2016 International Conference on Computing, Communication and Automation (ICCCA), 16 January 2017

[7] Justin J. Song , Inkyo Kang , Wookey Lee , Jinho Kim , Joo-Yeon Lee, ***"Discussions on Subgraph Ranking for Keyworded Search"***, 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)

[8] Ziyang Liu, Yichuang Cai,Yi Shan and Yi Chen, ***"Ranking Friendly Result Composition for XML Keyword Search"*** in Springer International Publishing Switzerland 2015.

[9] Roko Abubakar , Shyamala Doraisamy , Bello Nakone, ***"Effective Predicate Identification Algorithm for XML Retrieval"***, 2018 Fourth International Conference on Information Retrieval and Knowledge Management (CAMP)

[10] Yushan Ye , Kai Xie , Tong Li , Nannan He, ***"Result ranking of XML keyword query over XML document"*** , 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)

[11] Vagelis Hristidis, Yannis Papakostantinou, Andrey Balmin, ***“Xkeyword: Keyword Proximity Search on XML Graphs”***, in 11th International Conference on Data Engineering, 2002.