ISSN: 2278-4632
Vol-11 Issue-02 2021

DATA MINING TECHNOQUES AND ITS APPLICATIONS A SURVEY

Durga Shankar Baggam[1], Dr.Prakash Chandra Jena[2]
[1]Computer Science & Engineering,Gandhi Engineering College,Bhubaneswar, India
[2]Computer Science & Engineering,Gandhi Engineering College,Bhubaneswar, India

Abstract
Proximity Keyword Search is the best and de-facto mechanism that is utmost useful in searching the web and particularly in long unstructured XML documents. This system is designed to convert XML documents to relational databases by using a very scarcely used Ctree Concept. The Ctree concept helps us to match the XML very fastly to schema-less relational databases. Then the index is built on the database and helps in faster retrieval of Proximity Keyword Search in XML documents. It provides an efficient mechanism of generating ranked results for queries related to keyword search over XML documents. The proposed system is the first of its kind in which the keyword string is pre processed before searching the XML document. In particular, this system is implemented in two stages. In the first 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 generated ranks are directly dependent on the located keyword distance.

Keywords— XML, Relational Databases, Indexing, CTREE, Proximity Search, LCA

I. INTRODUCTION

XML, whose full-form is Extensible Markup Language is a widely used markup language for Structured Information documents. When we call it Structured Information, it represents words, pictures, etc. and even purpose of that content (for instance, footnote content means a lot more different than the section heading content which inturn is a lot more different than figure caption content or a database table content). Every document possesses a structure. With the rapid digital evolution, XML has been used in diverse fields such as data mining, intelligent retrieval, artificial intelligence, bio technology, medical science etc. Hence it has become popular to use XML to publish data on the internet and searching for useful information from XML documents has gained wide publicity. A markup language is a way to mark structures in a document. One standard way to markup the documents is XML. Let's discuss some differences between searching XML and HTML documents

Hypertext Markup Language simply known as HTML[1] is the conventional and easy-to-use markup language for generating both web pages and applications. It's core virtue is simplicity which allows a wide array of users to be benefitted. The same asset of simplicity can sometimes turn into a liability with ever- growing users needs to create their custom tags for simplifying their tasks. In an attempt to satisfy this demand, Extensible Markup Language has been developed that facilitates general application dependence which in turn further complements the HTML to be portable and powerful.

A great improvement of XML[2] above HTML is XML allows the linking support to multiple documents whereas HTML link can only reference a single 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 the Internet. XML allows us to define our own tags. XML Schema or DTD(Document Type Definition)is used by XML to describe the data structure. XML is designed to describe the presentation of the content, while XML is to describe the content itself. 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 main differences between using Hyper Text Markup Language and Extensible Markup Languages are: 1) XML identifies the user search intention, i.e., it identifies the XML node types which the user wants to search for and search via any other term. 2) XML helps is resolving keyword ambiguity problems i.e in a document a keyword can appear in a node as a tag as well as node value; and a keyword can appear in the tag name in different XML node types and each would have different meanings. 3) XML usually gives the final search results in the form of sub trees a part of the XML document. These sub trees are used to compute a score which would determine how the result is relevant to a given query. Existing methods developed so far has not been successful enough to return the query results which were relevant to the search.

Keyword Search [3] is on a steady-rise now-a-days when it comes to querying XML data as it substitutes the user from understanding the complex schemas of XML document and query languages such as XQuery and XPath. There are a lot of advanced algorithms and query processing techniques

ISSN: 2278-4632
Vol-11 Issue-02 2021

proposed to address the keyword search over XML data.

Proximity refers to nearness or closeness. Identifying the terms that are identical to one another is a proposition to make the search more semantic which is known as Proximity Search. In the same way Proximity Keyword Search is defined to be searching multiple keywords in a list of documents and retrieving the documents which match all the keywords with a condition that all keywords must be found with a certain number of words in between them. This certainty usually refers to the distance between the keywords and it can range from anywhere from 1 to Max (value depends on search engine). In general proximity keyword search could be termed as adding a constraint of proximity to simple keyword matching in a set of documents. This characteristic helps us on web searching which usually consists of vast amounts of unstructured data.

For example, a query could be made for " Blue Gate House" and it could match the documents " Mr Blue has built a beautiful house on the banks of River Gate", " The Gate was painted blue and the house in Red", "The house was fixed with a gate painted in blue". Our search intention was not to find out in which documents it appeared, but how the words have been associated with each other, and the semantics of the document in which the keywords match. It might not be possible to understand the true meaning of how the query terms are related to each other, but definitely we can return the query results based on how the keywords are closely associated in context of relevance score.

Considering the above, our work transforms XML documents of any organization into Ctree[5] which is a set of relational database tables. With the help of Ctree an index is built on all the words present in the documents. It provides an interface which assists users (who don't know any query language) of this system to search the 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 contain the keywords from the Ctree Index table. After locating the elements, with the help of other entries of the index table , the lowest common ancestor (LCA) of the keywords are 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.

II. RELATED WORK

In study number [6], The authors produced a page rank algorithm to find the most relevant answers to a proximity search query. They used a graph which used the page ranking algorithm to recursively find the nodes which matched the given query and found an equation to calculate the relevance scores. They summarized all the selected nodes and compared with the paths visited in the graphs to find the most relevant answers. This algorithm wouldn't solve our problem as it cant be used on XML documents.

Vajenti et al developed a strategy where all the keywords queried are grouped based on the levels of the XML tree derived from XML document and grouped based on the node names. The algorithm optimized the search of keywords in a framework which was not using any kind of indices.Justin et al in their study[8] devised a ranking algorithm based on the structure of a graph.s. Their work helps to retrieve the results which are related to a part of sub graph and ranking the results based on that particular part of the sub graph. It's a time consuming process to find out how the results are connected.

Ziyang Liu et al [9] proposed an algorithm called Target Search to find out query results based on user submitted targets. These needs targets to be defined by users. Roko et al developed an algorithm in which the user has to enter an entity to resolve ambiguity between the keyword search results and then it calculates the score of the results using fragments of the sub trees matching the keywords asked in the query. They developed some scoring algorithms for the ranking.

Yushan et al [11] devised an algorithm where the relationship between the keywords in the query are determined by the content of the XML documents. From the structure of the XML trees, they find the lowest common ancestor and infer the relationship between the leaf nodes. After evaluating the relationship they describe the degree of the proximity and score the results. This algorithm falls under keyword search result ranking schemes.

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.[12] 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. So, we tried to use efficient indexing which helps in computing the LCA with less complexity. And we focussed on displaying the XML subtrees by ranking them based on edge distance. It pre processes the keywords entered by the user before searching.

III. PROPOSED SYSTEM

The main idea of this paper is to model the XML documents into a Compact tree known as Ctree. This Ctree is lesser known and used in XML document conversion. But it's a very convenient way to represent the XML document into a set of relational database tables and could also be used as an index for retrieving the keywords efficiently from the database.

The design of the proposed system is divided into three steps as shown in Fig 1. (a) This requires XML documents to be parsed and stored in relational database in the form of tables. And building an index on this tabular data. (b) 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. (c)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.

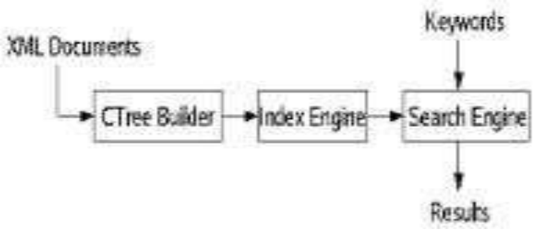


Fig. 1 Flowchart of Proposed System.

Before we describe how an XML Document is converted to relational tables, lets understand the Ctree data structure and its related terms: Ctree[8] is a similar form of binary tree with two levels which is used to summarize the entire XML document in a compact manner. Each node in CTree has two pointers, where the group pointer points to the nodes of similar child nodes having the same parent node and the element pointer stores data about its children nodes as well as respective parent nodes. Building of group pointer is usually done by identifying all the nodes with similar hierarchical sub structures who have a common parent node and storing them as a group. In the next level, each node holds the pointers sequentially to its list of parents and children.

Let's understand the terminology which helps in defining the CTree: *label path, equivalent nodes, Path Summary*. Figure 2 shows an example XML tree and all the definitions are defined referring to it.

Label Path for an XML Tree T is usually defined as the sequence of labels of nodes visited from the root to the leaf node (N) separated by a special character dot (.) and represented as *LP(N)*. Let us take an example of XML Tree shown in Fig 2. Node 14 could be visited through nodes 1-13-14, hence its label path is expressed as *dblp.article.title*.

Equivalent Nodes for an XML Tree T are usually the list of nodes which have a similar label path. In Fig 2 , nodes 4, 15, 18 are termed as equivalent as they share the same label path *dblp.article.author*.

Now that we know the terms Label Path and Equivalent Nodes, we use them to define Path Summary for an XML Tree. Path Summary is usually represented as a Tree where each node is termed as a Group which matches to exactly one label path LP(N) in a XML Data Tree (T). This Group Node usually has a set of equivalent nodes. If this set is sorted out based on their parent links, it is known as Ordered Path Summary. Fig3 shows an example of Path Summary for the XML Tree shown in Fig 2. Each node is dotted in this tree which identifies its to be a group. The values in the dotted box are the list of equivalent nodes identified. Every Group has a label associated with it and a pointer to its parent node. Take the instance of nodes 3,14,17 of the XML Tree shown in Fig (2). Their Label Path is *dblp.article.title* and are labelled as Group 2 with name title in Fig 3(a). If you keenly observe every data tree for a corresponding XML tree has a unique Path Summary.

Lets define Ctree on the terms of Path Summary. It is a tree with a unique root and set of child nodes. Each child node is termed as Group and denoted by g and contains a set of elements. Each element in the group g is denoted by g.pid and satisfies :

- Each Group g is defined in two parts : group id (g.id) and group name (g.name).
- Traversal is from top to bottom from groups. It means we can visit the nodes from root to leaves and hence the edge directions.
- An edge from g1 to g2 represents that g2 is the child node of g1 and g1 is the parent node of g2. In general if a path exists from g1 to g3, then g3 would be descendant of g1 and g1 the grandparent/ancestor of g3.
- Element k in a group g is denoted by g:k and usually termed as an array index. g.pid[k] points to

ISSN: 2278-4632
Vol-11 Issue-02 2021

elements in g's parent group gp. gp.g.pid(k) is known as the parent element of g:k. If g.k1 and g.k2 are two elements in a group, and if k1 is less than k2, then g.pid[k1] <= g.pid[k2].

Let us understand the above terminology with the help of an example XML Tree shown in figure 2. Fig 3(a) shows the corresponding label path summary and Fig 3 (b) shows the corresponding Ctree after according to the groups definition. The Ctree has a set of nodes with edges towards its child nodes , each node has a label and a set of values separated by commas.. The set of values represent the positions of the elements beginning from 0. The three elements in the group 3:author are pointed as 3:0 (first child of article element) , 3:1 (second child of article element), 3:2 (third child of article element and their values being 0,1,2 with relative reference of nodes in the sub tree.

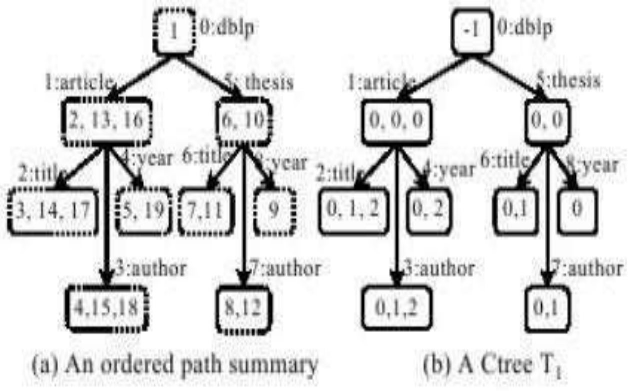


Fig. 3 Path Summary and its equivalent Ctree for given XML TREE

Searching Keywords: The Ctree index supports a search(word) operation. It usually returns the nodes where the word matches in the form of a list of group id's (if we specify the group id in the search) or a list of parent element ids (if we don't specify the the group id). The index built on Ctree is clustered and inverted. This inverted index is usually built on three elements namely word id, group id and element id. In a successful match , we get the word ids. Once we know the element id's and group id's where the keywords have occurred, we can use our LCA algorithm to find Lowest Common Ancestor which connects the keywords.

- The algorithm is as follows:
- Locate the group id's and element id's of the given keywords from the index table and store it in two lists.
- If the group id's of all the keywords are same, check their element id's are equal.

❖ If they share the same group id – Display their element id's along with the given keywords.

❖ If they are different– find a minimum connecting tree with the lowest common ancestor of the keywords by retrieving their parent element ids and group ids.

- Else

- Retrieve the depth of each keyword. Let p and q be the keywords which are at maximum depth and minimum depth respectively.
- Recursively reach to the ancestor of every keyword which is at level(q) from the keywords which have depth <= p.
- Compute the LCA of the ancestors.
- Rank the results based upon the distance between the keywords.

Score of a XML Document*:* In addition to distance between the keywords, a metric known as score is also computed for every XML document. Let's 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 defined 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).

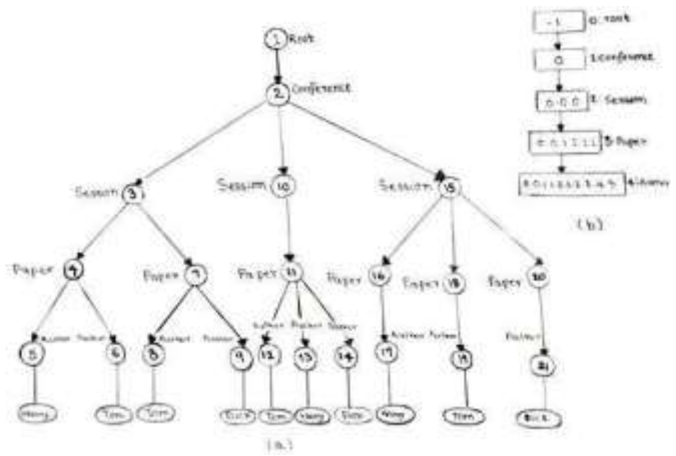


Fig. 4 XML Tree and its corresponding Ctree (Label Path Summary)

Displaying the Results: The user is always interested to know the results which are closely associated , so we calculate the lowest common ancestor (LCA) for the keywords searched. Sometimes these LCA's are also termed as Minimum Connecting Trees (MCT's). The LCA's which are computed for a 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. For example if the user submits the keywords Tom, Dick, Harry against the XML document of 4(a), Fig 4(b) shows the corresponding Ctree and Figure 5 on the left hand side shows the possible minimum connecting trees.

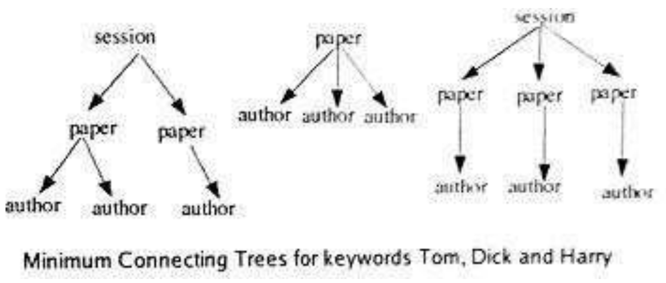


Fig. 5 Minimum Connecting Trees for keywords

IV. SYSTEM IMPLEMENTATION

System is implemented in three stages.

Ctree Builder : SAX parser is used for parsing the XML document. JAVA API is used to process the XML documents and build the Ctree.

Index Engine : It contains the following three components. *Analyzer* : Helps in analyzing the given keywords by filtering out the white spaces, converting all uppercase letters to lowercase letters, tokenizing the keyword strings and deleting the stop words.

Parser: This component parses the given XML documents and builds an index based on the content present in XML tags.

Ctree: This deals with the 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.

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

Implementing CTREE

Ctree index is mapped into four relational tables

Elements : This table has two columns and it stores the elements and their respective parent groups. The Groups table has four columns and usually stores the group id, group name, the level in which it appears in the C Tree, the label path and number of descendants.

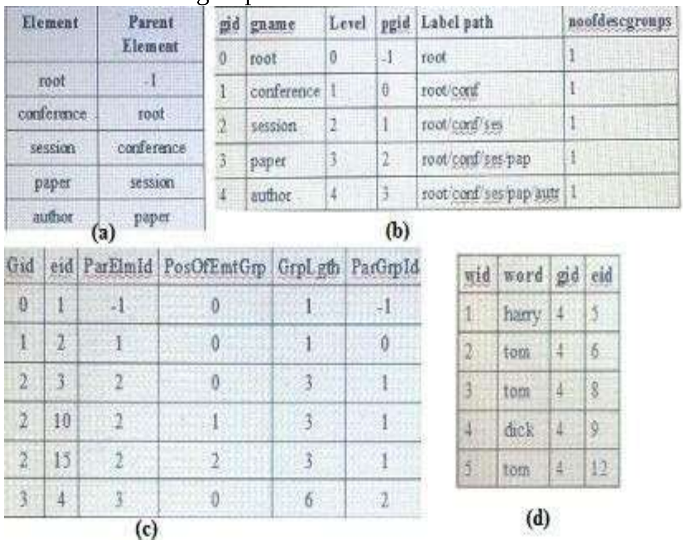
The CtreeDB table has four columns (Ctree name, the file group, total number of group elements, total number of elements).

The ElmPosLen table has two columns used to store the position of each element in the group and its length(label path).

The invert table is an index which uses the the table Words. This table decreases the storage cost as it effectively maps the keyword to an identifier in the words table and eliminates redundant comparison of strings. The identifiers matched from the Words table helps to find out the words and its positions from the Hits table

ISSN: 2278-4632
Vol-11 Issue-02 2021

which stores the group id's and element id's.



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

(a)

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

(b)

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

(c)

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

(d)

Fig. 6 Snapshot of Tables populated with values

Tables a, b, c, d shown in Fig (6). shows a snapshot of values populated in Elements, Groups, ElmPosLen, Words tables when the example XML document Fig 4 is converted into Ctree. An inverted index is built on the words table based on keywords present in the XML data. This index matches the words which contain the keywords i.e returns an array of elements (group id's). Once we know the group id's we can compute the sub trees by tracing the parent id's, element id's and word id's from the inverted index table.

Searching the Keywords

Let's take an example when two keywords k1 and k2are given as input to the search interface. The algorithm locates the word id’s where the keywords are present from the index table. 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 present in the same levels, then retrieve the ParElmId's of the parents of the elements which contain the keyword. If they are different, 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. Keep adding the edge distance as we iterate to find out the LCA.

Table 1 List of Keywords matched

“Tom” Occurrences			**“Harry”** Occurrences		

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.Group 4 contains the word with wid 2 and is present within element with eid is 6. Group4 contains the word with wid 1 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 have 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

ISSN: 2278-4632
Vol-11 Issue-02 2021

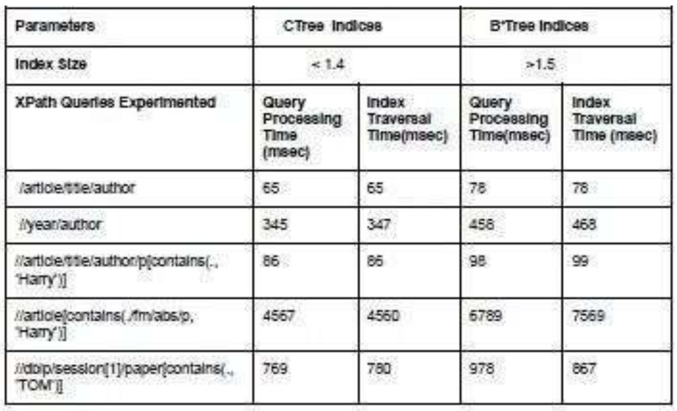
edge distance s 2. Both the elements are at the same level. Now find out the parents of elements with eid's 17 and 19. ParElmId of 17 and 19 is 4. So add two to the edge distance value. Element with id:4 is the LCA of the keywords with edge distance 4. Keyword Tom has occurred 4 times while Harry has 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 the least distance is displayed first. Table 1 shows the list of values populated for given keywords.

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.
Displaying the Results :
Results are displayed to the user graphically. Details such as field, , 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. COMPARISON OF RESULTS

Differences between CTree Indexing and B$^+$Tree
Indexing
We implemented CTree and examined its effectiveness with respect to building index with B$^+$ Tree[13]. We used the Generalized Search Tree for the implementation of B$^+$ Tree
[14] Indices. We used the DBLP Data set [15]. We compared CTree Index with the method proposed in [14], which is compatible with keyword searches using inverted list on keywords. We experimented with the B$^+$ Tree Indexes built with an inverted list consisting of keywords.
[15]

VI. RESULTS :



Parameters	CTree Indices		B$^+$Tree Indices	
Index Size	< 1.4		>1.5	
XPath Queries Experimented	Query Processing Time (msec)	Index Traversal Time(msec)	Query Processing Time(msec)	Index Traversal Time (msec)
/article/title/author	65	65	78	78
//year/author	345	347	458	468
//article/title/author/p[contains(., 'Harry')]	86	86	98	99
//article[contains(./fm/abs/p, 'Harry')]	4567	4560	6789	7569
//dblp/session[1]/paper[contains(., 'TOM')]	769	780	978	867

VII. CONCLUSIONS

Unlike previous approaches, this work provides the distance analysis of the keywords. The entire XML document is stored in the in-memory as the trees are stored in the form of Ctree which are tables. 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 trees such as isomorphic trees, filtering out redundant trees.

REFERENCES

[1] 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
[2] Gelin Deng et al , “Approximate Searching XML Elements Based on Semantic Restrictions”, 2008 International Conference on Management of e- Commerce and e-Government , 17-19 Oct.2008
[3] Qinghua Zou, Shaorong Liu, Welsley W.Chu, “Ctree: A Compact Tree for Indexing XML Data”, in WIDM 2004.
[4] 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)
[5] 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
[6] 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)

ISSN: 2278-4632
Vol-11 Issue-02 2021

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

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

[9] Yushan Ye , Kai Xie , Tong Li , Nannan He, "Result ranking of XML keyword query over XML document"

[10] , 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.

[12] Toshiyuki Shimizu1, Masatoshi Yoshikawa2 ,"Full- Text and Structural XML Indexing on B+-Tree" Part of the Lecture Notes in Computer Science book series (LNCS, volume 3588)

[13] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer, "Generalized Search Trees for Database Systems," In VLDB, pp.562–573, September 1995.

[14] https://dblp.uni-trier.de/xml/