A Comparative Study of Keyword Based Searching on XML Trees Using Compact Tree Indexing Algorithm and Entity Sub Tree Algorithm

Swapna JawariKapisha[1] and G. Vijaya Lakshmi[2]
[1]*Vikrama Simhapuri University, Nellore, Andhra Pradesh.*
swapnajk2001@gmail.com, vijaya_suma17@yahoo.co.in

Abstract

Keyword Searching on XML documents have been in use as it is used to store both structured and Semi Structured data. The beauty of XML is that it can store multi data types in a single document and allows you to link the documents. To be user friendly in querying XML documents without being aware of the format or tags or structure of XML documents, has been in demand with the advent of the digital revolution. There has been extensive research done retrieving efficient results for keyword search in XML documents. For efficient search results, primarily a good index structure is needed which is the basis for index traversal time and query processing time. In this paper, we would like to highlight the usage of compact tree index structures which helps in decreasing the query processing time to retrieve the keywords and also helps in maintaining a low index file in terms of storage. Also, it was proved experimentally that the time taken by our approach is lesser than the Entity Subtree(ES) Tree Index Structures.

Keywords: XML, Keyword Searching, Compact Tree, XML Region, Indices

1. Introduction

The digital revolution and the growing use of the internet with huge amounts of data on the world wide web has made us to explore and invent new ways of storing and sharing data. eXtensible MarkUp Language (XML) is used to represent data from a wide variety of web sources. XML is the first language that makes the documents both human readable and computer manipulatable. It is the language of the intelligent document, a step ahead of conventional methods of document representation that rely on format rather than structure. Data independence, the separation of the content and its presentation, is the essential characteristics of XML. Because an XML document describes data, it can conceivably be processed by any application. The absence of formatting instructions makes it easy to parse. This makes XML an ideal framework for data exchange on the web. XML has become prominent especially after the realization of the inadequacy of HTML to represent

information on the web. In this typical environment, there is a need to develop keyword search engines exclusively for XML documents. In Contrast to searching keywords on static documents the results retrieved by XML Keyword Search must be fine grained and they should be related and satisfy the user's query. As all the documents are represented in trees and subtrees in the XML database, identifying the results with finer granularity is of utmost importance.

In general XML data is stored in a database management system which could be traditional , structured , semi-structured . Then to retrieve this data, firstly the user should give keyword queries in a language which is understood by the database engine. Usually the user should know the semantics of Xpath which is a query language to express XML queries. There are many other XML specific languages which have rules and are complex to learn by the user. Secondly, how to index the XML data to increase the performance of the query engine for frequent keyword queries. There should be an efficient index, to facilitate searching frequent queries, taking less time to update the index , and access the relevant data. Indexing plays an important role in any kind of search engine. Without an efficient index structure, no search engine will perform to its best.

There have been many traditional indexing techniques which used to consider an entire document as a set of words and used to find the keywords belonging to the document or not. These conventional indexing won't work for databases which are semi structured and where data is related with more that one document. Researchers have found many techniques to address this but they are not yet proved to be used effectively on the heterogenous public data systems. In contrast to traditional indexes, XML indexes are different in representation, building and updation as the XML documents are in tree structures. This has given rise to new problems and challenges for tree based indexes. There have been many previous studies [1] on the challenges for XML indexing whose main aim was to reduce the XML data to be searched and speed up the query processing times.

That's why our main focus in this paper is to explore different index structures for XML Keyword search queries which is faster in query processing and optimize the index traversal time. Section 2 discusses the existing approaches which were introduced to address the indexing problems. Section 3 lists various kinds of indexing techniques based on databases used to store data. Section 4 discusses using Compact Ttree (CTree) [2] indexing structure which addresses the existing challenges. Section 5 describes the Entity Subtree[3] Indexing which also uses a similar kind of indexing approach. Section 6 presents the comparison results of the proposed system with Entity Subtree Indexing [3] and conclusion remarks.

2. LITERATURE REVIEW

Any indexing structure to be most efficient must be able to retrieve the sub trees of XML documents which represent related keywords. In one way XML index must be able to process the trees top-down or bottom-up by traversing the tree nodes. The results could be simple paths or complex paths which are known as twigs. Labeling XML document is a method to assign an unique identifier to each node in the XML tree which holds information about its

position in the tree and its relationship with other nodes. The relationship says that each node can be a parent, ancestor, child, descendant or sibling to another node in the XML tree. These relationships play a very important role in accelerating query processing.

Let's take an example query article[author="Jane"] and see how it is evaluated in Path traversal systems. In top-down processing, the query is processed by looking at all downward paths starting from any article element which has an immediate author element. It then traverses downward to find the article authored by "Jane". Next, it looks for all the article elements in the document to determine all possible paths. To process the keyword search query consisting of "article" and "issn_num", it needs to traverse all the paths starting from root node article to leaf node which consists "issn_num".

In general there would be more than one match, so it needs to backtrack again to its previous node which contains an article to search for the next child node which contains the "issn_num" element.

This is a time taking, in-efficient method to search the matching keywords in the XML trees. So, we move on efficient indexing techniques which overcome exhaustive path traversals and are also effective in processing parent-child related queries.An index technique would be proved as efficient if it helps in deducing the relationship between the nodes present in different levels which are encoded in the index files [4]. It should also help in knowing the attributes of the document such as number of levels, depth of the tree, number of children and number of similar nodes. The index file must be able to manage the load of the hierarchical XML document.

Gou et al [5] had categorized two types of indices based on the traversals. a) XML Path Indices are used for processing the simple path queries. During path query evaluation, it considers the whole path instead of each node in the path separately. b) XML Twig indices were used for processing twig queries. In the processing of twig query, a subsequent joining of simple paths is required. In comparison with the node indexing scheme, the path indexing scheme requires less number of joins in query processing, thereby improving the performance.

In contrast to above, Vakali et al[1, 6] summarized different indices based on the content and structure of the document instead of path and twig queries. They defined a structural summary index structure which preserves all the paths from root nodes to leaf nodes. This eliminated book keeping information about the hierarchical structure and still maintained the ancestor-descendant and Parent-Child relationships. The main advantage of these indices are they are very effective in processing path queries, but not suitable for twig queries. They differentiated the structural summary into path based, node based and sequence based indices. Similar to Gou et al[5], path indices were used to store the label paths whereas Node indices were defined to store node names and joins were needed to reconstruct the structure. The latter most sequence based indices stored data related to both the documents and queries in a sequential manner to match the keyword queries by sequence matching.

Path Index Structures mainly focus on root to leaf paths and don't bother about the content. They used to take the help of a supplementary value index for the content. Because of this,

the query processing cost involves not only joins but also recursive look ups in the index to match the keywords. An index structure which stores both the path and content was devised by Cooper et al. [7]. However, it cannot support ancestor-dependent queries efficiently. They cannot efficiently process the partial match or content-based queries.

Node Index Structures [8, 9] are based on the labeling schemes which can be containment based or prefix-based. It holds the value that depicts the node's position in the XML document. These indices support both parent-child and ancestor-dependent relationships and it needs only two comparisons to infer. But a challenging task is to update the index When a node is inserted or deleted from the XML data tree, it may have to relabel some or entire nodes. For a keyword query with p number of nodes, n-1 joins are needed to evaluate. This is the main drawback of node indices because of the huge number of structural joins to evaluate a query.

In Sequence Structural Index structures both the XML document and the keyword query are converted into a predefined sequence. It uses a subsequence matching algorithm to evaluate the query results. Structural-encoding sequences are defined by Haixun et al [10] to store the XML data and the search query. The SES consists of two components. The first component is the element tag and the second is the path of its parent starting from the root node. The encoding starts by scanning the data tree in a top-down approach. For the deep and long data tree, the size of the index may become a problem, as it does not scale well with an increase in data size and the top elements have to be included in the root path as a second component. Hence, the length of the sequence will increase as the path of the XML data tree gets longer. Label length increases as the tree depth increases. To evaluate a branch query with multiple identical child nodes, it disassembled the query tree at the branch into multiple trees. Further, it is processing using the join operations and combines their result. Hence, the solution is expensive as it requires additional join operations.

Praveen et.al [11] overcomes the limitation of Haixun [12] which requires a large number of nodes (paths) to be examined during commonly occurring non contiguous tags names. It represents the entire XMl Tree in a bottom up style using prufer sequences which plays a vital role in reducing the query processing time. The main limitations of Sequence based indices are that they support only ordered branch queries, and usually involve repeatedly visiting the nodes unnecessarily which increases the high I/O cost.

Hence there is a dire need to devise an index structure which can help in overcoming the limitations.

1. It should be able to preserve both content and structural properties on XML documents.

2. Should be able to encode all kinds of relationships between the documents.

3.Index file size should be less and reduced index file updates

4. Speed up query processing

In the proposed study we use an indexing structure which addresses all the above issues and have the advantages of Path based and Node based indexing. Compact Tree Index (Ctree) [2, 13] is used to index the XMl documents which stores not only the path summaries which

preserve the Parent-Child relationships but also detailed summary of ancestor to dependent relationships at element level. It also speeds up the query processing as it prunes out a large number of irrelevant nodes and matches the context using an inverted index.

3. Existing techniques of Proximity Keyword Based Search on XML documents

The notion of proximity among keywords is more complex for XML. In HTML, proximity among keywords translates directly to the distance between keywords in a document. However, for XML, the distance between keywords is just one measure of proximity; the other measure of proximity is the distance between keywords and the resultant XML element. we need to consider a two-dimensional proximity metric involving both the keyword distance (i.e., width in the XML tree) and ancestor distance (i.e., height in the XML tree).

1. XRANK: XRANK proposed by Lin Guo et al.[14] generalizes HTML search engine. It evaluates keyword search queries over hyperlinked XML documents, as opposed to at HTML documents and also ranks XML results by taking into account the granularity of an XML element instead of granularity of a document. It is the first system to take into account (a) the hyperlinked structure of XML documents and (b) a two-dimensional notion of keyword proximity, when computing the ranks for XML keyword search queries. XRANK proposed specialized index structures such as Ranked Dewey Inverted List(RDIL) and Hybrid Dewey Inverted List(HDIL) and query evaluation techniques that provide significant space savings and performance gains.

2. XSEARCH : XSEarch proposed by Cohen et al.[15] provides a semantic search engine for XML which has a simple query language suitable for naive user. It returns semantically related document fragments that satisfy the user's query. Query answers are ranked using extended information retrieval techniques and are generated in an order similar to ranking. Advanced indexing techniques such as interconnection index and path index are proposed which help in efficient searching and ranking. The interconnection index allows for rapid checking of the interconnection relationship. Path index allows us to create initial answers with high estimated ranking.

3. On User-Centric XML Keyword Search [16] : They devised algorithms which used human collective intelligence to improve data processing tasks by providing a rich user interface to the user. They discussed different aspects of processing tasks over XML Data, search and retrieval and personalization but an efficient XML index was not used.

4. EBQS and STPS : Roko at al [17] address query results which were returning irrelevant predicate nodes from XML data, they devised entity based query segmentation (EBQS) method that return correct user query interpretation and also proposed segment terms proximity scorer (STPS) which helps to resolve query keyword ambiguity. The main limitation was a lot of bookkeeping information was needed to build the index for a given XML Document.

5. Layered Intersection Scan Algorithm : Yushan et al [18] proposed a XML semantic weight-value structural model which can speculate the relationship of the keywords based on the characteristic of XML documents by Smallest Lowest Common Ancestor. This algorithm

calculates the SLCA by using Dewey Coded and a lot of preprocessing on the xml sub structures is made which is time consuming.

6 Keyword Proximity Search On XML Trees: Proposed by Vagelis et al. [19]. In the case of XML trees, the problem of keyword proximity search reduces to the problem of finding the subtrees rooted at the lowest common ancestors (LCAs) [20] of the XML nodes that contain the keywords. They found solutions for two main problems: 1)identifying and presenting in a compact manner all MCTs which explain how the keywords are connected and 2) identifying only MCTs whose root is not an ancestor of the root of another MCT. They designed algorithms to compute MCTs in 2 cases: 1) when the XML data has been preprocessed and relevant indices have been constructed and 2) when the XML data has not been pre processed.

In this proposed study we show how compact tree index structures can be used for effective proximity keyword based searching on huge XMl documents.

.

4. Proposed Method

4.1 : Compact Tree : Ctree[2] 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 the element's parents. Thus Ctree is an efficient index for processing the structure constraints of XML queries.

A compact tree is defined to contain a group level and an element level. 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 dene label path, equivalent nodes, Path Summary which helps in describing 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 1 can be reached from the root node 1 through the path: 1-6-8. Therefore the 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 1 are equivalent since their label paths are the same dblp.thesis.author.

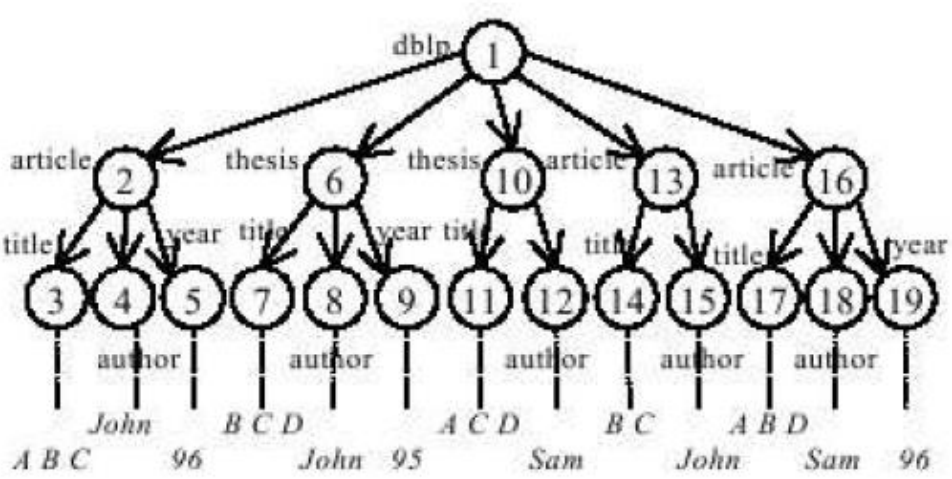


Figure 1 : Example XML tree from DBLP Data set

Path Summary : For a data tree D, a path summary as in [21][22] is a tree on which each node is called a group and corresponds to exactly one label path l in D. The group contains all the equivalent nodes in D sharing the label path l. We call a path summary an ordered path summary if the equivalent nodes in every group are sorted by their pre-order identifiers. For example, an ordered path summary for the XML data tree in Figure 1 is shown in Figure 2a. 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 nodes 2, 13, 16 are in group 1 since their label paths are the same: dblp.article. Every data tree has a unique path summary [21].

4. 2 Definition of Compact Tree :

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

1. Each group g is associated with an identifier and a name, denoted by g.id and g.name respectively.
2. Edge directions are from the root to the leaves. If there is an edge from g1 to g2, then g1 is called the parent of g2 and g2 is called a child of g1. If there is a path from g1 to g3, then g1 is called an ancestor of g3 and g3 is called a descendant of g1.
3. An array index k of g.pid[ ] represents an element in g, denoted by g:k. The value of g.pid[k] points to an element in g's parent gp; and gp:g.pid[k] is called the parent element of g:k.
4. For any two elements g:k1 and g:k2, if k1 < k2, then g.pid[k1] g.pid[k2].

For referring to an element k in group g, g:k is called an absolute reference and k is called a relative reference. For example, Figure 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 to by 4:0(rst child of article element) and 4:1(second child of article element), whose

values are 0 and 2 which are relative references for elements 1:0(rst child of dblp element) and 1:2(third child of dblp element).

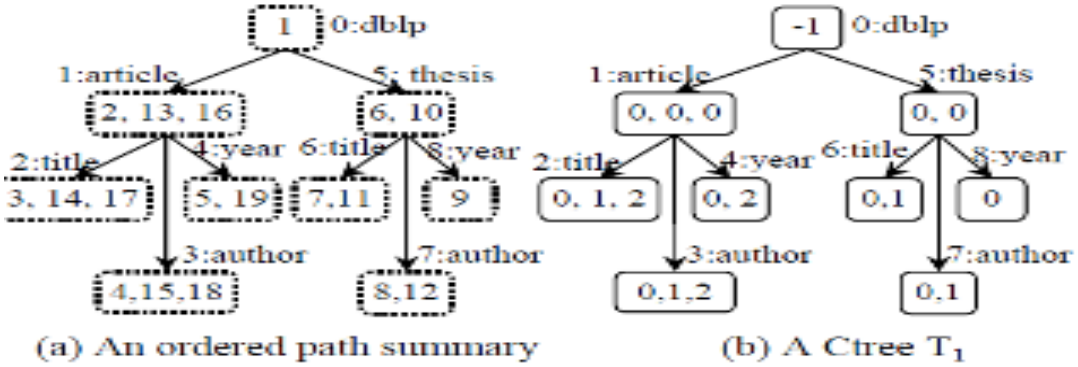


Figure 2 (a) Path Summary for the XML tree Figure 2(b) Ctree for XML Tree

4. 3 Properties of Ctree :

1. Every data tree D has a unique Ctree TD.
Proof: We can construct a TD in the following three steps:
1) Create a path summary S for D.
2) Replace the collection C of equivalent nodes in every group g of S with a array g.pid[] of the same size as C and build the mapping M: d□g:k where d is a data node and k is its position in C.
3) Build element level pointers. Given d is the parent of d' in D, if d->g1:k1 and d'->g2:k2, then let g2:k2 point to g1:k1, i.e.,g2.pid[k2]= k1.
Using the above steps, only one TD can be constructed

With the Ctree in Figure 2(b), we can answer not only single-path queries but also branching queries. For example, for the query /dblp/article[title and year], elements 1:0 and 1:2 are the answers since the boxes in groups 2 and 4 contain values 0 and 2. A Ctree
has parent-child edges at the group level and provides child parent links at its element level. Such a bidirectional tree makes Ctree better than other indexing methods.

We know that the elements in a group are arranged in consistent with the order of their parents. In other words, for two elements i and j in a group g, if i precedes j, then i's children precedes j's in every child group of g.
2. Monotonic property: The values of a group's array are arranged in increasing order. That is, if i<j, then g.pid[i] ≤ g.pid[j].
This property enables us to use a binary search to locate the child elements of a given element. It also results in the contiguous property.

4.4 Advantages of Ctree : The proposed Ctree-based query processing algorithm has the following advantages:

1. Locating frames at the group level prunes a large number of irrelevant groups at an early stage. If there is no grouplevel match, it returns an empty answer in step 1. For example, for a query //article[author =”John”]/address on DBLP, it will return no matches at the first step since the path //article/address does not exist in the Ctree of the DBLP [23] dataset. Many previous approaches, however, require a set of expensive join operations to return the no answer.
2. Evaluating a value predicate based on a group significantly reduces the possible matches and improves the efficiency of combining the matches for value predicates and structure constraints.
3. Using an array for fast mapping elements to their parents facilitates the evaluation of element level structure constraints. Such fast access to elements’ parents is essential for efficient XML query processing since the tree nature of XML data and queries is rooted in sharing common parents.

4. 5 Building CTree Index : Ctree index is built in three steps.

1. The Scan module collects the structure and value characteristics from an XML dataset and extracts schema information if it exists. The Scan also proposes indexing options for each group of equivalent data nodes based on data features and schema.

2. A user reviews the proposed indexing options and makes proper adjustments to finalize the index configurations.

3. Based on the index configurations, the Index Builder constructs a Ctree and builds a value index for the XML dataset. The Invert uses the table Words to map a word to an identifier (wid) which minimizes storage overhead by eliminating replicated strings and computational overhead by eliminating expensive string comparisons. The table Hits stores the occurrences and positions (pos) of words (wid) in XML elements (gid:eid).

4.6 Searching Keywords : The Ctree index supports a search(word) operation. The search operation returns a list of absolute elements (when the gid is not specified) or relative element (when the gid is specified). Since the inverted index is clustered by (wid, gid, eid), the operation search(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 ids 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 the 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 the ancestor of every keyword which is at level(q) from the keywords which have depth less than equal to p.

(c) Compute the LCA of the ancestors.

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

4.7 Score of the 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 dened as 100. With n keywords we can nd n! combinations. If a XML document contains less than n number of keywords say p, its score is dened 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).

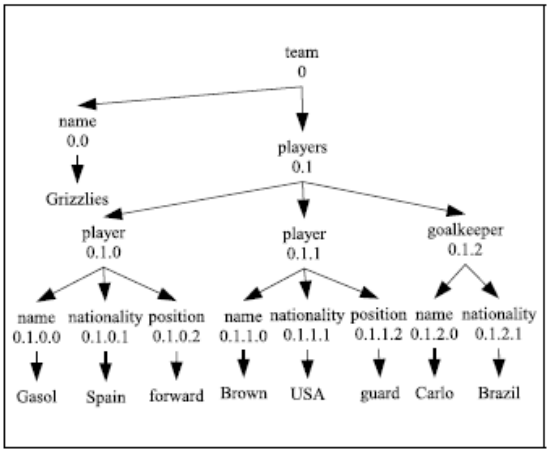
5. Entity Subtree

Entity Subtree[3] are similar to XML trees where the tree is constructed with relevance nodes where the semantic relevance of all the interconnected nodes are exploited. XUdong et al[3] proposed a new query language similar to regular expressions known as Grouping and Categorization Keyword Expression and the core query algorithm, finding entity subtrees (FEST) is proposed to return high quality results by fully using the keyword semantic meanings exposed by GCKE. As discussed previously, existing approaches use node labels to identify the relationship between nodes. In contrast to previous approaches where they try to find the relationship between the nodes using node labels, they used a different approach of prefixing the path from root to child. The prefix path of a node is the path from the root to its parent. For instance, the prefix path of node [0.1.0.0] in figure 3(a) is [team/players/player] shown in 3(b).

5.1 Entity Subtree Structure

1. A node represents an entity if it corresponds to a *-node in the DTD.
2. A node denotes an attribute if it does not correspond to a *-node, and only has one child, which is a value.
3. A node is a connection node if it represents neither an entity nor an attribute. A connection node can have a child that is an entity, an attribute or another connection node.

They define that two nodes a and b are meaningfully related if a and b are of same type and there do not exists two other nodes with the equivalent type of shortest path from a to b, except a to b.



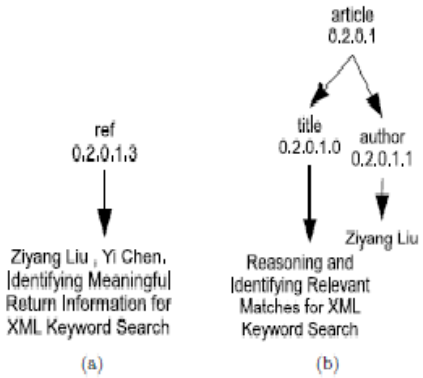


Figure 3 (a) Example XML Data Tree **Figure 3 (b) Entity Subtree for th elements**

6. Comparison of Keyword Search using Entity Subtree and Ctree

To validate the efficiency of Ctree in XML indexing, we tested DBLP [26] data set for keyword searching. Ctree uses edge scoring to determine the relevance of the keyword queries whereas the Entity sub tree needs intervention from the user in prior hand to define the prefix of the path. The latter can calculate the relevance only by determining the node meaningfulness by using the prefix paths. The finding entity subtrees (FEST) is proposed to return high quality results by fully using the keyword semantic meanings exposed by Grouping and Categorization Keyword Expression (GCKE). Though they return the relevant results faster, the limitation of learning the GCKE language is a must for the end user which is not needed while searching using Ctrees.

Table 1 showing processing Times of Ctree and Entity Subtree Keyword Search

Parameters	CTree Indices		Entity Sub Trees	
XPath Queries Experimented	**QPT**	**ITT**	**QPT**	**ITT**
/article/title/author	65	65	63	72
//year/author	345	347	418	492
//article/title/author/p[contains(., 'Harry')]	86	86	102	105
//article[contains(./fm/abs/p, 'Harry')]	4567	4560	5768	5998
//dblp/session[1]/paper[contains(., 'TOM')]	769	780	798	845

7. Conclusion

In this paper we tried to compare two keyword based search algorithms using two different XML index structures Ctree and Entity Subtree. Both are good in preserving relationships between different nodes. Entity subtrees can be effectively used for personalization search results and where the user can spend some time to learn the query language whereas Ctree could be used efficiently for searching huge heterogeneous databases without any extra cost of learning the structure of the XML document.

References

[1] *Barbara Catania, Anna Maddalena, Athena Vakali " XML Document Indexes", Proc. IEEE Internet Computing, SEPTEMBER - OCTOBER 2005, pp 64 -71.*

[2] *Qinghua Zou, Shaorong Liu, Welsley W.Chu, "Ctree: A Compact Tree for Indexing XML Data", in WIDM 2004.*

[3] *Lin, Xudong & Wang, Ning & Xu, De & Zeng, Xiaoning. (2010). A novel XML keyword query approach using entity subtree. Journal of Systems and Software. 83. 990-1003. 10.1016/j.jss.2009.12.024.*

[4] *I. Tatarinov et al., "Storing and Querying Ordered XML Using a Relational Database System," Proc. Int'l Conf. Management of Data (ACM Sigmod), ACM Press, 2002, pp. 204–215.*

[5] *Q. Li and B. Moon, "Indexing and Querying XML Data for Regular Path Expressions," Proc. Int'l Conf. Very Large Databases (VLDB 01), Morgan Kaufmann, 2001, pp. 361–370*

[6] *YP.E. O'Neil et al., "Ordpaths: Insert-Friendly XML Node Labels," Proc. Int'l Conf. Management of Data (ACM Sigmod), ACM Press, 2004, pp. 903–908.*

[7] *R. Goldman and J. Widom, "DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases," Proc. Int'l Conf. Very Large Databases (VLDB 01), Morgan Kaufmann, 1997, pp. 436–445.*

[8] *W. Wang et al., "Efficient Processing of XML Path Queries Using the Disk-Based F&B Index," to appear, Proc. Int'l Conf. Very Large Databases (VLDB), Morgan Kaufmann, 2005.*

[9] *T. Milo and D. Suciu, "Index Structures for Path Expressions," Proc. Int'l Conf. Database Theory (ICDT 99), LNCS 1540, Springer-Verlag, 1999, pp. 277–295.*

[10] *C.W. Chung et al., "APEX: An Adaptive Path Index for XML Data," Proc. Int'l Conf. Management of Data (ACM Sigmod), ACM Press, 2002, pp. 121–132.*

[11] *Praveen rao, Bongki Moon, "PRIX: Indexing and querying XML using prufer sequences", Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA.*

[12] *Haixun Wang, Sanghyun Park, Wei Fan, PhilipS.Yu, "ViST: A DynamicIndex MethodforQueryingXMLDataby TreeStructures" SIGMOD 2003,June9-12,2003,SanDiego,CA.Copyright2003ACM1-58113-634-X/03/06.*

[13] *S. Al-Khalifa et al., "Structural Joins: A Primitive for Efficient XML Query Pattern Matching," Proc. Int'l Conf. Data Eng. (ICDE 02), IEEE CS Press, 2002, pp. 141–152.*

[14] *Lin Guo, Feng Shao, Chavdar Botev, Jayavel Shanmugasundaram, "XRank: Ranked keyword Search over XML Documents," in SIGMOD, 2003.*

[15] *Sara Cohen, Jonathan Mamou, Yaron Kanza, Yehoshua Sagiv, "XSEarch: A Semantic Search Engine for XML," in Proceedings of the 29th VLDB Conference, 2003.*

[16] *L. M. Amini and M. Keyvanpour, "On user-centric XML keyword search," 2018 4th International Conference on Web Research (ICWR), Tehran, 2018, pp. 51-57, doi: 10.1109/ICWR.2018.8387237.*

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

[18] *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)*

[19] *T. Chen, J. Lu, and T.W. Ling, "On Boosting Holism in XML Twig Pattern Matching Using Structural Indexing Techniques," Proc. Int'l Conf. Management of Data (ACM Sigmod), ACM Press, 2005, pp. 455–466.*

[20] *H. Jiang et al., "XR-Tree: Indexing XML Data for Efficient Structural Joins," Proc. Int'l Conf. Data Eng. (ICDE 02), IEEE CS Press, 2002, pp. 253–263.*

[21] *A. Silberstein et al., "BOXes: Efficient Maintenance of Order-Based Labeling for Dynamic XML Data," Proc. Int'l Conf. Data Eng. (ICDE), IEEE CS Press, 2005, pp. 285–296.*

[22] *B. Catania et al., "Lazy XML Updates: Laziness as a Virtue of Update and Structural Join Efficiency," Proc. Int'l Conf. Management of Data (ACM Sigmod), ACM Press, 2005, pp. 515–526.*

[23] *H. Wang et al., "ViST: A Dynamic Index Method for Quering XML Data by Tree Structures," Proc. Int'l Conf. Management of Data (ACM Sigmod), ACM Press, 2003, pp. 110–121.*

[24] *P.R. Raw and B. Moon, "PRIX: Indexing and Querying XML Using Prüfer Sequences," Proc. Int'l Conf. Data Eng. (ICDE), IEEE CS Press, 2004, pp. 288–300.*

[25] *H. Wang and X. Meng, "On the Sequencing of Tree Structures for XML Indexing," Proc. Int'l Conf. Data Eng. (ICDE), IEEE CS Press, 2005, pp. 372–383.*

[26] *Michael Ley. DBLP database web site. http://www.informatik.uni-trier.de/ley/db.*