ISSN: 2321-7782 (Online)

Volume 2, Issue 3, March 2014

International Journal of Advance Research in Computer Science and Management Studies

Research Article / Paper / Case Study

Available online at: www.ijarcsms.com

Efficient Extraction of Frequent Elements from XML Document Using XML Tree Pattern Matching

M. Rekha[1]
Computer Science and Engineering
SVU College of Engineering
Tirupati – India

N. Usha Rani[2]
Computer Science and Engineering
SVU College of Engineering
Tirupati – India

Abstract: Now a days eXtensible Markup Language (XML) documents play vital role in storing the database in different organizations. This is because of their rich and flexibility in retrieving the required information. As the requirement of XML data is ever-increasing, there is an increasing need for efficient processing of queries on XML data to extract the occurrences of a tree pattern query. To query XML document there is a need to use any one of the XML query languages, this requires a basic knowledge about the language such as syntax to write a query. This paper provides user interface to query the XML document. Present paper focus on building new hoard (temporary storage) for storing the repeated queries. There is a chance of querying about the same element again while processing queries on XML data. In this context, there is a need for efficient extraction of frequently asked elements. The present work focus to extract results for frequently asked query or elements efficiently by storing the results of every query (query may consists of elements or nodes) in hoard, with this time complexity will be reduced. This paper come across a large set of XML tree pattern, called extended XML tree pattern, which may include parent-child (P-C), ancestor-descendant (A-D) relationships, negation functions(¬), wildcards(*) and order restriction(<).

Keywords: Tree pattern, XML, Query processing, Hoard, XPath.

I. INTRODUCTION

XML is the universal format for structured documents data on web. XML documents are used to transfer data from one place to another often over the Internet. XML is a markup language much like Hyper Text Markup Language (HTML). XML is not a replacement for HTML. XML is more flexible and adaptable than HTML. XML was designed to carry data not to display data. XML and HTML were designed with different goals: XML was designed to store, transport and display required data. HTML was designed to display data, with focus on how data looks HTML is about displaying information, while XML is about carrying information.

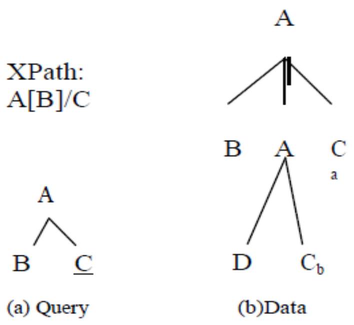

Fig. 1. Example XML tree query and document.

In HTML tags are predefined but in XML tags are not predefined. XML Document Type Definition (DTD) and XML Schema define rules to describe data. XML provides a robust, non-proprietary, persistent, and verifiable file format for the storage and transmission of text and data.XML query pattern is tree like structure, commonly represented as rooted, labeled tree.

Let A, B, C are the elements in the XML document, fig. 1(a) shows an XPath query: A [B]/C and the corresponding XML document which is represented as XML tree pattern fig. 1(b). This query finds all 'C' nodes that has sibling 'B' and has parent 'A'.

In fig. 1(a) "_" denotes return node in the query. Answer for the query from fig. 1(b) is 'C_a' only, not 'C_b' why because 'C_a' has sibling 'B' and parent 'A'. But 'C_b' has parent 'A' but there is no sibling 'B', it has sibling 'D'. Here Parent-Child (P-C) relationship is denoted with "/" and Ancestor- Descendant (A-D) relationship is denoted with "//". An XML tree pattern with negation function, order restriction and wildcards is called as extended XML tree pattern [1]. Consider the following three extended XML tree patterns.

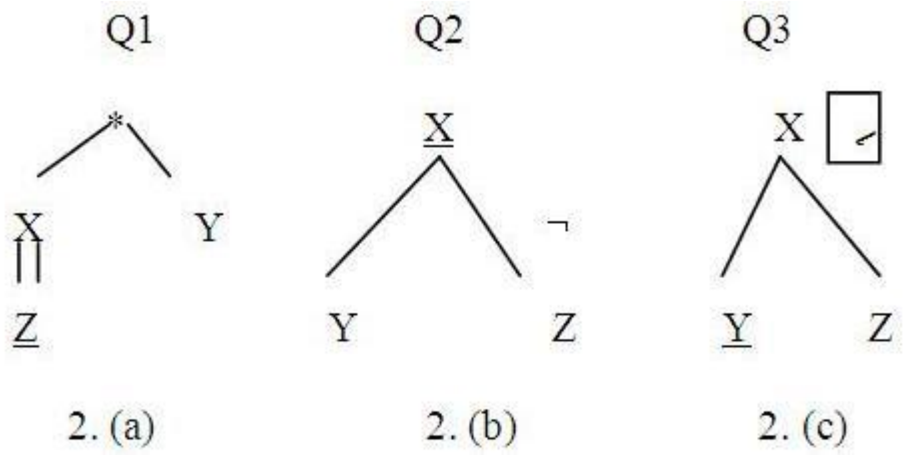

Fig. 2. Extended XML tree pattern queries

Query 1 in 2.(a) include a wildcard node "*" which can match any single node in an XML document. Query 2 in 2.(b) includes a negative edge which is denoted by "¬" , this query finds X that has a child Y, but has no child Z. In Xpath language negative edge can be represented with Boolean function "not". Query 3 in 2.(c) has the order restriction, '<' in a box shows that all children under X are ordered. In XPath order restriction are denoted by following, following-sibling.

XPath is a language used to selecting and retrieving the required parts of an XML document. Path to the required element is defined in the same way as OS defines paths to files. Present work use XPath language to query XML document, a query may include /,//,*,<,¬. Following are the some of the basic terms used in this paper:

Query class: Query class is a set of XPath queries Eg: query class: //*[A]/B//C, //A[B][not (C)].

Query nodes: This paper refers that each node in the query class as query node. For example in query class //*[A]/B//C query nodes are A,B and C.

Return nodes: Return nodes are also called as output nodes. All nodes in query might not be return nodes, only part of query nodes belong to return nodes. Consider an example XPath A[B]/C, from this XPath expression return nodes are only C elements and its subtrees. Previous methods find the query answer with the combination of all query nodes, and then perform an appropriate projection on those return nodes. This is a post-processing approach, obviously this type of approaches has disadvantage because it returns many elements which are not relevant for final result. This paper used a new encoding method [1], to record mapping relationships and avoids non-return nodes in the output.

This paper maintains a special hoard to store query and its corresponding results. By listing this, it is easy to answer the query which is previously asked. If the same query is repeated then the result will be retrieved from the hoard and displayed to the user. If it is not available in the hoard, then the query is to be evaluated. After evaluating query, query and corresponding result will be stored in hoard for future reference. If hoard is not maintaining and same query is asked frequently, then the query is to be evaluate every time, and it is a time taking process. To overcome that limitation, this paper storing each result along with the query. Therefore if the same query occurs again, then it is easy to retrieve from the hoard and display the same as result. Hence the time to retrieve the result is reduced.

II. RELATED WORK

Structural indexes [2][3] is build over XML documents to avoid unnecessary scanning of source XML documents. Structural join is essential to XML query processing because XML queries usually have certain structural relationships such as Parent-Child (P-C) and Ancestor-Descendant (A-D) relationships [4]. Structural indexing of XML document is a technique to partition XML document and this technique is called as streaming scheme [5] [6]. Zhang et al [7] proposed a Multi-Predicate MerGe JoiN (MPMGJN) algorithm for binary structural join, based on region labeling of XML elements. The later work by Al-Khalifa et al [8] gave a stack-based binary structural join algorithm, called Stack-Tree-Desc/Anc which is optimal for an P-C and A-D binary relationship. Wu et al [9] studied the problem of binary join order selection for complex queries. N. Bruno et al [10] proposed a holistic twig join algorithm, namely TwigStack, to avoid producing a large intermediate result. TwigStack is I/O optimal for queries with only ancestor-descendant edge. Therefore, TwigStack is optimal for queries with only A-D edges. TwigStack may output many useless intermediate results for queries with parent-child relationship. TwigStack cannot process XML twig queries with ordered predicates, like “Preceding”, “Following” in Xpath. TwigStack cannot answer queries with wildcards in branching nodes.

Lu et al.[11] proposed TwigStackList, which identifies a larger optimal query class than TwigStack. TwigStackList enlarges the optimal query class of TwigStack by including P- C relationships in non-branching edges. Lu et al.[12] also researched how to answer an *ordered* twig pattern based on region encoding. Choi et al [13] demonstrated that no matter how elements in a Tag Streaming stream are ordered, it is impossible to achieve optimal holistic matching. Lu et al [11] proposed a look-ahead method to reduce the number of redundant intermediate paths. Jiang et al [14] used an algorithm based on indexes built on element containment labels. Finding all the occurrences of a twig pattern in an XML database is a core operation for efficient evaluation of XML queries. To process a twig query based on region encoding labeling scheme number of algorithms have been proposed. A new labeling scheme called Extended Dewey was proposed in [15].this scheme is a powerful labeling scheme, because only with the label of an element, it is possible to derive all the elements names along the path from the root to the element. Based on this scheme a new twig join algorithm called TJFast was designed. In TJFast, to answer a query it only needs to access the labels of the leaf query nodes.

Dewey ID labeling scheme comes from the work of Tatarinov et al.[16] to represent XML order in the relational data model, and to show how this labeling scheme can be used to preserve document order during XML query processing. O'Neil et al.[17] introduced a variation of prefix labeling scheme called ORDPATH. Unlike our *extended Dewey*, the main goal of ORDPATH is to gracefully handle insertion of XML nodes in the database. TwigStack and TwigStackList cannot handle order-based twig query. XPath and XQuery includes ordered axes such as following, preceding, following-sibling and preceding-sibling. OrderedTJ algorithm was proposed[12] for evaluating ordered twig query pattern. OrderedTJ additionally checks the order conditions of elements before outputting intermediate paths.

Previous twig pattern matching algorithms is an efficient technique to answer an XML tree pattern with parent-child (P-C) and ancestor-descendant (A-D) relationships, as it can effectively control the size of intermediate results during query processing. However, XML query languages (e.g. XPath, XQuery) define more axes and functions such as negation function, order-based axis and wildcards. In [1] TreeMatch algorithm was proposed to achieve a large optimal query class for three categories of queries such as negation function, order-based axis and wildcards.

III. TREEMATCH ALGORITHM WITH HOARD

In prior work if user asked same query frequently the system has to execute the query repeatedly. Since it is a time taking process present work extends the work done in [1] to efficiently extract the frequent elements from XML documents by maintaining hoard which reduces the time complexity. Xpath query language was used to query XML document. This paper proposed an efficient technique to answer an XML tree pattern with parent-child (P-C), ancestor-descendant (A-D), negation functions, wildcards and order restriction. All these types of queries can be answered by using TreeMatch algorithm [1]. Same TreeMatch algorithm is used in this work but it is extended by storing the results acquired from this method along with query in hoard. Hence it is called as modified TreeMatch algorithm. This will increase the performance of the existing method. This will work as follows: if any user asked the same query about any element in an XML document then the result will retrieved from hoard instead of searching for the element in XML document.

TreeMatch algorithm [1] for query class Q $^{/,//,*}$ works as: In first step it locates the first elements whose paths match the individual root-leaf path pattern. In each iteration, a leaf node f_{act} is selected by *getNext* function. Next insert the potential matching elements to *outputlist*. Next step advances the list Tf_{act} and update the set encoding. Then locate the next matching element to the individual path. Finally, when all data have been processed, we need to empty all sets in Procedure *EmptyAllSets* to guarantee the completeness of output solutions.

In Procedure *addToOutputList*(*q; eqi*), the potential query answer e_{qi} is added to the set of S_{eq} , where *q* is the nearest ancestor branching node of *qi* (i.e. *NAB*(*qi*) = *q*). Procedure *updateSet* accomplishes three tasks. First, clean the sets according to the current scanned elements. Second, add *e* into set and calculate the proper *bitVector*. Finally, we need recursively update the ancestor set of *e*. Because of the insertion of *e*, the *bitVector* values of ancestors of *q* need update.

The purpose of the extention of TreeMatch algorithm for query class $Q^{/,//,*,<}$ is to maintain and check the order relationship among the matching elements of query sibling nodes. In Procedure updateSet, we need to set the proper *minChild* and *maxChild* according to the current elements. In Function satisfyTreePattern, we also need to check the order restriction according to *minChild* and *maxChild*. TreeMatch algorithm is further extended to support negative edges. *negBitVector* is added to record the matching information about negative child edge. In Procedure updateSet, we need to set the proper *negBitVector* according to the current elements.

IV. EXPERIMENTAL RESULTS

An XML document itself is considered as a database for this project. Here XML document consisting of book details of departmental library in a college was considered as a database to perform experiments. Book details such as book name, author, price, publisher, ISBN, years are considered as elements in that XML document. TreeMatch algorithm is implemented on book database which is in XML format. Number of queries has been devoted to modify TreeMatch algorithm for book database which enclose approximately 700 books and observed that time to execute and retrieve result is reduced. This is shown in the following Table1.

Table1: Elapsed time for different queries

Type of Queries	**Time to execute** (in milli seconds)	
	TreeMatch	TreeMatch with hoard
Q1: Display all book details from bookshelf	25	10
Q2: List all book details whose subcategory is language	43	30
Q3: show all book details whose subcategory is not multimedia	39	23
Q4: show all book whose name is CAD/CAM	42	31
Q5: Display all book details from bookshelf	25	5

Table1 consists of 5 queries and corresponding time to execute in both TreeMatch algorithm and Modified TreeMatch with Hoard. From this table it is observed that execution time is reduced on an average of 10milli seconds for considered book database. As shown in the above table new method will answer efficiently for same kind of queries. Consider Q1 and Q5, both queries are same. Time to execute Q1 (first time) by using TreeMatch algorithm is 25ms where as by using TreeMatch with hoard is 10ms. Executing query Q5 (i.e, executing Q1 second time) by using TreeMatch algorithm will take same time as it is executing first time but by using TreeMatch with hoard the execution time will reduce to 5ms.

Pictorially the comparison of execution time is as shown below:

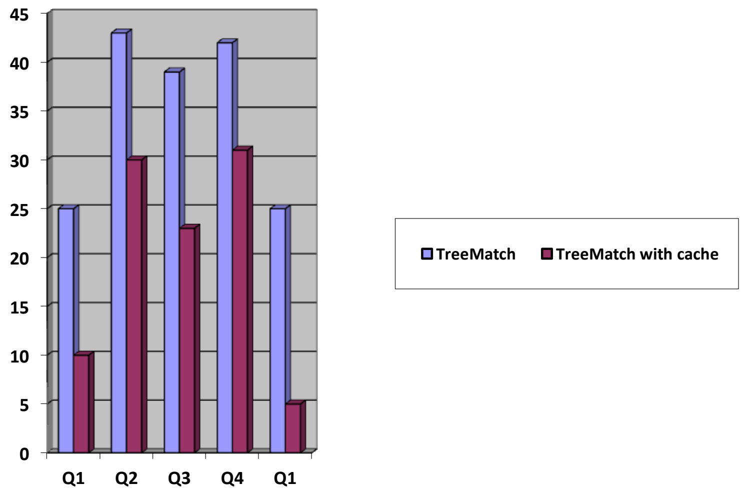

Fig 3: comparison of execution time for existing and proposed method

V. Conclusion

This paper introduced an efficient approach for XML query processing. Here TreeMatch algorithm is modified in such a way that hoard is incorporated to reduce the time to retrieve the required data. The hoard consists of frequently asked queries and its relevant elements. Through this approach it is possible to answer extended XML tree pattern queries which may include parent-child, ancestor-descendant relationships, negation function, wildcards and order restriction. This approach reduces time consumption. This method will be well suited for same kind of queries. This work can be extended by applying semantic analysis for the queries.

References

1. J. Lu, T. W. Ling, Z. Bao, and C. Wang. Extended xml tree pattern matching: theories and algorithms. In Technical Report, 2010.
2. Q. Chen, A. Lim, and K. W. Ong. D(k)-index: An adaptive structural summary for graph-structured data. In Proceedings of SIGMOD 2003, pages 134–144, 2003.
3. H. He and J. Yang. Multiresolution indexing of XML for frequent queries. In Proceedings of ICDE 2004, pages 683–694, 2004.
4. T. Chen, J. Lu, and T. W. Ling. On boosting holism in xml twig pattern matching using structural indexing techniques. In SIGMOD, pages 455– 466, 2005.
5. T. Milo and D. Suciu. Index structures for path expressions. In Proceedings of ICDT 99, pages 277–295, 1999.
6. R. Kaushik, P. Sheony, P. Bohannon, and E. Gudes. Exploiting local similarity for efficient indexing of paths in graph structured data. In Proceedings of ICDE 2002, pages 129–140, 2002.
7. C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, and G. M. Lohman. On supporting containment queries in relational database management systems. In Proceedings of SIGMOD 2001, pages 425–436, 2001.
8. S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava, and Y. Wu. Structural joins: A primitive for efficient XML query pattern matching. In Proceedings of ICDE 2002, pages 141–152, 2002.
9. Y. Wu, J. M. Patel, and H. V. Jagadish. Structural join order selection for XML query optimization. In Proceedings of ICDE 2003, pages 443–454, 2003.
10. N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal XML pattern matching. In Proceedings of SIGMOD 2002, pages 310–321, 2002.
11. J. Lu, T. Chen, and T. W. Ling. Efficient processing of XML twig patterns with parent child edges: a look- ahead approach. In CIKM, pages 533-542, 2004.
12. J. Lu, T. Ling, T. Yu, C. Li, and W. Ni. Efficient processing of ordered XML twig pattern matching. In DEXA To appear, 2005.
13. B. Choi, M. Mahoui, and D. Wood. On the optimality of the holistic twig join algorithms. In In Proceeding of DEXA 2003, pages 28–37, 2003.
14. H. Jiang, W. Wang, H. Lu, and J. X. Yu. Holistic twig joins on indexed XML documents. In In Proceeding of VLDB 2003, pages 273–284, 2003.
15. J. Lu, T. W. Ling, C. Chan, and T. Chen. From region encoding to extended dewey: On efficient processing of xml twig pattern matching. In VLDB, pages 193–204, 2005.
16. I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang:. Storing and querying ordered XML using a relational database system. In Proc. of SIGMOD, pages 204{215, 2002.
17. P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-friendly XML node labels. In SIGMOD, pages 903-908, 2004.

AUTHOR(S) PROFILE

M. Rekha received B.Tech degree in Computer Science and Engineering from Sri Venkatesa Perumal College of Engineering and Technology, JNTUA University, Anantapuram, A.P, India in 2009 and currently pursuing M.Tech, Computer Science and Engineering, final semester, from Sri Venkateswara University College of Engineering, TIRUPATI, A.P, India. Her interested areas are Data Mining, Artificial Neural Networks, and Software Architecture. She attended Two National Conferences during 2012 and 2014.

N. Usha Rani N. Usha Rani is working as Assistant professor in the department of Computer Science & Engineering, SVU College of Engineering, Sri Venkateswara University, Tirupati. She did B.Tech in CSE, JNTU, Hyderabad. She did M.Tech in Artificial Intelligence, University of Hyderabad. She is pursuing Phd in the university of Hyderabad. She published papers in International journals. She presented papers in National and International conferences. Her research areas are Speech processing, Data Mining, Artificial Neural Networks, Fuzzy logic, Genetic Algorithms & Machine learning.