Asian Journal of Computer Science and Technology
ISSN: 2249-0701 (P) Vol.3 No.1, 2014, pp.1-5

DOI: https://doi.org/10.51983/ajcst-2014.3.1.1733

An Implementation of Tree Pattern Matching Algorithms for Enhancement of Query Processing Operations in Large XML Trees

N. Murugesan[1] and R.Santhosh[2]

[1]PG Scholar, [2]Assistant Professor, Department of Computer Science and Engineering,

Karpagam University, Coimbatore, Tamil Nadu, India

E-mail: nmuruhesen@gmail.com

(Received on 25 December 2013 and accepted on 05 March 2014)

Abstract – **Now-a-days XML has become a defacto standard for storing, sharing and exchanging the information across the various domains. Interoperability is achieved using XML. Due to the increasing popularity of XML enterprises are generating and exchanging the data across the various domains. This paper presents a wide analysis to identify the efficiency of XML Tree pattern matching algorithms. Previous years many methods have been proposed to match XML Tree queries efficiently. In particularly TwigStack , OrderedTJ, TJFast and TreeMatch algorithms. All algorithms to achieve something through these own ways like structural relationship including Parent-Child (P-C) relationship (denoted as '/') and Ancestor-Descendant (A-D) relationships (denoted as '//') and more. Finally, we report our results to show that which algorithm is superior to previous approaches in terms of the performance.**

Keywords: **XML, TreeMatch, TwigStack, XQuery, XPath**

I. Introduction

Data mining is the process of analyzing data from different perspectives and summarizing it into useful information. In this project mining is applied to gain knowledge for large amount of XML Datasets. XML has become ubiquitous language sharing, storing and exchanging information across various platforms. XML documents can be represented as a Tree structure using DOM Parser. DOM Parser is mainly used to store, access or manipulate the XML Tree. XQuery (XML Query Language) and XPath (XML Path Language) are traditional XML query languages to query the XML Data. Our existing system provides answers to the queries using these query languages. These query languages requires some complex notations to perform query processing. XQuery and XPath are powerful but unfriendly to non-expert users. Existing system uses TwigStack Algorithm to perform query answering. But, TwigStack algorithm provides answers to the queries containing P-C (Parent-Child) and A-D (Ancestor-Descendant) relationships. This causes sub-optimality problem in proposed system we are using keyword query to perform query answering. A Holistic XML Tree Pattern Matching Algorithm called TreeMatch is used to overcome the sub-optimality problem faced by the existing system. This algorithm is based on the concept of extended Dewey Labeling. According to the Labeling Scheme the root node, children, grand children are associated with the number or label. For instance 0 is assigned to the root node. The children of the root gets labeling such as 0.0, 0.1. The grand children of the first parent node start with 0.0.0 and continue like 0.0.1 etc.

II. Related work

Several research papers have studied in the area of XML Tree Pattern Matching and the surveys of those papers have been presented here.

J.T.Yao, M.Zhang (2013) have proposed a holistic algorithms for XML Query Processing. The novel holistic XML twig pattern matching method called TwigStack which avoids storing intermediate results unless they contribute final results. The major advantage of this method is that it avoids computation of large redundant intermediate results. But main limitation of TwigStack is that it may produce large set of "useless" intermediate results when queries contain

parent child relationship. TwigStack has been proved optimal only for queries with A-D edges and it still cannot control the size of intermediate results for queries with P-D edges. TwigStack operates in two steps.

1. A list of intermediate path solutions is output as intermediate results and
2. The intermediate path solutions in first step are merge-joined to produce the final solutions.

Xiaoying Wu, Stefanos Souldatos (2011) have proposed MPMGJN (multi-predicate Merge-Join) algorithm and typically this algorithm consists of decomposition-matching and merging process:

- Decompose the tree pattern into linear patterns which might be binary (parent-child or ancestor–descendant) relationships between pairs of nodes or root-to-leaf paths.
- Find all matching's of each linear pattern
- Merge-join them to produce results.
- MPMGJN varies from TwigStack merge join algorithm is that it requires multiple scans of input list.

Li *et al.* and Chien *et al.* (2011) have proposed Stack-Tree Algorithm which mainly used to overcome the drawbacks of MPMGJN algorithm. The major drawback of MPMGJN algorithm is that is requires multiple scan of input list whereas Stack-Tree algorithm needs only one scan of the input lists. Stack Tree algorithm uses stacks to maintain the ancestor or parent nodes. Stack Tree Algorithm works for both P-D and A-D edges.

Jaihaeng Lu (2010) has proposed OrderedTJ Algorithm which is mainly used to overcome the drawbacks of decomposition-matching-merging algorithms. In OrderedTJ algorithm an element contributes to final results only if the order of its children accords with the order of corresponding query nodes. If we call edges between branching nodes and their children as branching edges then denote the branching edge connecting to the nth child as the nth branching edge. OrderedTJ is I/O optimal among all sequential algorithms that read entire input. In other words, the optimality of OrderedTJ allows the existence of parent-child edges in non-branching edges and the first branching edge. OrderedTJ algorithm output much less intermediate results, OrderedTJ increases linearly with the size of the database; OrderedTJ is not optimal and outputting less intermediate results.

Al-Khalifa *et al*. (2007) have proposed TJFast algorithm to overcome the drawbacks of containment labeling scheme. While containment labeling scheme preserves the positional information within the hierarchy of an XML Document but some limitations of containment labeling scheme are

- The information contained by a single containment label is very limited. For example, we cannot get path information from any single containment label.
- Wildcard are widely used in XPath and it cannot be supported by the containment label scheme.

The containment label scheme is difficult to answer queries with wildcards in branching nodes. TJFast does not produce the individual solution for each node when there are multiple return nodes for the query. TJFast cannot work with ordered restriction and negation function.

Wen-Chiao Hsu (2007) has proposed CSI-X technique to speed up the query evaluation in XML documents. CIS-X mainly used to overcome the drawbacks of decomposition-matching-merging algorithms to process XML Path expressions. According to decomposition-matching-merging algorithms a query is decomposed into several sub-queries, each of which is separately executed and its intermediate results stored for further processing. However these methods still have drawbacks of producing large intermediate results and time-consuming merging processing. So in this paper CIS-X technique has been proposed which support for complex XQueries. But the drawback with the CIS-X Technique is that it takes more time for index construction.

K. Kubota, Y. Kanemasa (2006) have proposed a new algorithm called Twig Square Stack which mainly used to eliminate the merging costs in second phase. Twig Square Stack is a one phase algorithm which can process path matching efficiently and avoids the high cost of merging phase. The overall solutions are stored in hierarchical stacks

and the final solutions can be output by applying a simple enumeration function. However the data structures are too complex and expensive to maintain.

X. Wu, D. Theodoratos (2005) have proposed an algorithm TwigList which is a refined version of Twig Square Stack, utilizing a much simpler data structure, a set of lists to store solutions. TwigList has advantages over Twig Square Stack but has same shortcomings. One drawback is that all the potential nodes related to QP (Query Processing) will be pushed into and popped from the temporary stack, even though some of them are not part of the solution. Another drawback is they have less ability to efficiently discard useless nodes.

S. Al-Khalifa, H.V. Jagadish (2009) have proposed Structural Join methods to process twig pattern matching. In the first phase, a twig query is decomposed into several binary P–C or A–D relationships. Each binary sub-query is separately evaluated and its intermediate result is produced. The final result is formed by merging these intermediate results in the second phase. This method generates a huge number of intermediate results that may not be part of the final results. In addition, the phase of merging is expensive.

Jihaeng Lu *et al.* (2011) have proposed TreeMatch algorithm which is mainly used to solve the problems of existing Algorithms for Twig Pattern Matching. TreeMatch provides optimal results for queries by eliminating useless intermediate results. The sub-optimality problem faced by the existing system is solved by using Tree Match Algorithm. This algorithm is based on the concept of extended Dewey labeling. As per the labeling scheme the root node, children, grand children are associated with number or label. For instance label 0 is assigned to the root node, children of root gets labeling such as 0.0, 0.1 and the grand children of the first parent node start with 0.0.0 and continue like 0.0.1.TreeMatch Algorithm mainly used for searching large XML Tree Patterns. The efficiency of the tree match is better when compared with other algorithms.

III. Existing System

Existing System uses Traditional XML Query languages like XQuery and XPath to perform query processing in an XML File. Existing System used TwigStack Algorithm to make pattern matching. TwigStack Algorithm provides answers to queries containing P-C and A-D relationships. P-C edges are denoted by (/) and A-D edges are denoted by (//). The TwigStack Algorithm is a decomposition-matching and merging algorithm. According to this algorithm a query is decomposed into several sub-queries. Each sub-query is executed separately and intermediate results are stored for further processing. The final result is obtained by merging these intermediate results. TwigStack Algorithm provides useless intermediate results for queries containing P-C relationships and it controls the size of intermediate result for queries containing A-D relationship. The TwigStack algorithm is described by the following:

```
    // Phase 1
1:  while notEnd (q)
2:  qact = getNext (q)
3:  if (isNotRoot (qact)) then
4:  cleanStack (parent (qact), nextL (qact))
5:  end if
6:  if (isRoot (qact) or isNotEmpty (Sparent (qact))) then
7:  cleanStack (qact, next (qact))
8:  moveStreamToStack (Tqact, Sqact, pointertotop (Sparent
    (qact)))
9:  if (is Leaf (qact)) then
10: showSolutionsWithBlocking (Sqact, 1)
11: pop (Sqact)
12: end if
13: else
14: advance (Tqact)
15: end if
16: end while
    // Phase 2
17: mergeAllPathSolutions ()
```

Algorithm TwigStack operates in two phases. In the first phase (lines 1-16), some (but not all) solutions to individual query root-to-leaf paths are computed. In the second phase (line 17), these solutions are merge-joined to compute the answers to the query twig pattern.

A. *Drawbacks of An Existing System*

The major drawbacks of an existing system are described below:

- XQuery and XPath is complicated to understand by non-database users.
- XQuery and XPath are not user friendly to non-expert users.
- Query Answering becomes little bit complicated using XQuery and XPath.
- TwigStack Algorithm fails to control the size of useless intermediate results.

VI. Proposed System

In proposed system keyword search is used to perform query processing in an XML Tree. Keyword Search is an simple and yet familiar to most of the internet users as it requires only the input of keywords. An XML Tree Pattern Matching algorithm called TreeMatch is used in the proposed system which is mainly used to overcome the drawbacks of the TwigStack Algorithm. In proposed system the exact matching is performed for users query not only for text but also for images, audio, video. TreeMatch algorithm is based on the concept of extended Dewey Labeling. According to the Labeling Scheme the root node, children, grand children are associated with the number or label. For instance 0 is assigned to the root node. The children of the root gets labeling such as 0.0, 0.1. The grand children of the first parent node start with 0.0.0 and continue like 0.0.1 etc. The TwigStack Algorithm is illustrated as follows:

1: locateMatchLabel (Q);
2: while(endroot)) do
3: f_{act} = getNext(topBranching Node);
4: if (f_{act} is a return node)
5: addToOutputList (NAB(f_{act} ,cur(Tf_{act}));
6: advance (Tf_{act}); //read the next element in Tf_{act}
7: updateSet (f_{act}); //update set-encoding
8: locateMatchLabel (Q); //locate next element with matching path
9: emptyAllSets (root);

Line 1 locates the first element whose paths match individual root-leaf path pattern. In each iteration, a leaf node f_{act} is selected by getNext function (line 3). The purpose of line 4, 5 is to insert the potential matching elements to outputlist. Line 6 advances the list Tfact and line 7 updates the set encoding. Line 8 locates the next matching element to the individual path. Finally, when all data have been processed, we need to empty all sets in Procedure EmptyAllSets (Line 9) to guarantee the completeness of output solutions.

The proposed system does not require complex query languages like XPath and XQuery. TreeMatch Algorithm matches with the extended Dewey Label for given query and then completes the query processing. Processing time of the TreeMatch Algorithm is less when compared to the decomposition-matching and merging algorithms. TreeMatch algorithm does not produce useless intermediate results. Thus by introducing TreeMatch Algorithm it takes less processing time. The major advantage of the TreeMatch Algorithm is that it solves the sub-optimality problem.

V. Experiments and results

The concept of TwigStack algorithm has been tested with the help of an open source tool called XPath Builder. XPath Builder is mainly used to generate XPath Expressions.

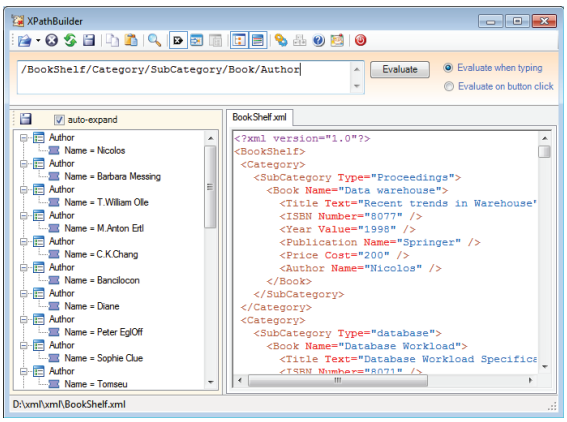

Fig.1 Execution of Queries containing P-C relationship. Large useless intermediate results for queries containing P-C relationships

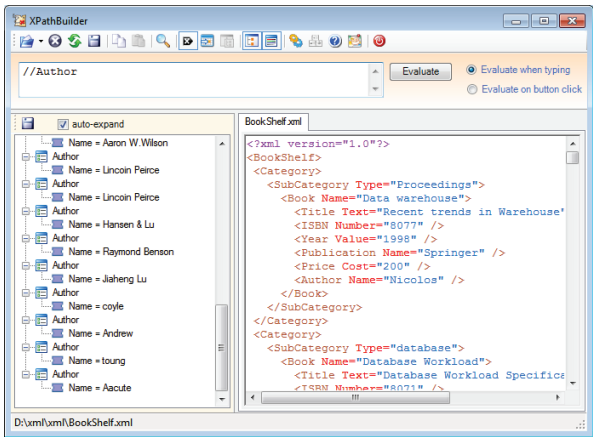

Fig.2 TwigStack Algorithm Reducing useless intermediate results for A-D edges

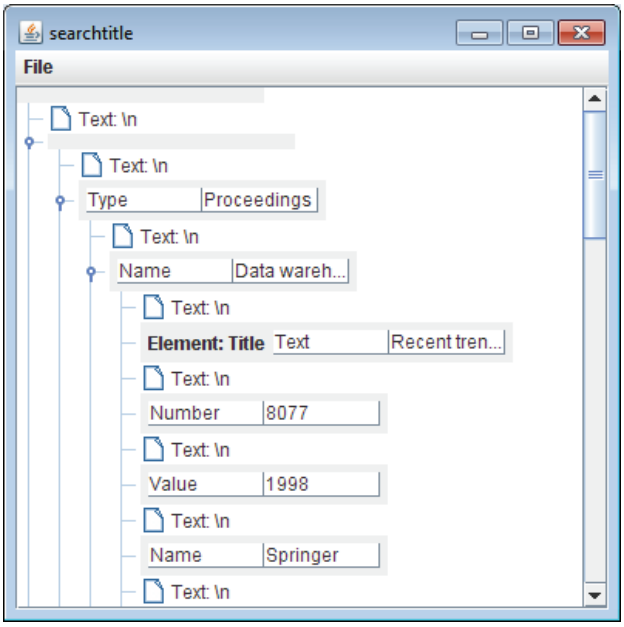

Fig.3 TreeMatch Algorithm Reduces useless intermediate results and makes query processing easy

Thus from our experimental our experimental results we conclude that TwigStack algorithm produces useless intermediate results where as the proposed holistic XML Tree Pattern Matching algorithm solves this sub optimality problem. Thus we have proved that TreeMatch Algorithm for XML Tree Pattern Matching is the best algorithm for performing query processing in a Large XML Trees.

References

[1] Mirjana Mazuran, Elisa Quintarelli, and Letizia Tanca, "Data Mining for XML Query Answering Support", *IEEE Transactions on Knowledge and Data Engineering,* 2012.

[2] Jiaheng Lu, Tok Wang Ling, Zhifeng Bao and Chen Wang, "Extended XML Tree Pattern Matching Theories and Algorithms", *IEEE Transactions on Knowledge and Data Engineering,* 2011.

[3] Jiaheng Lu, "Ordered and Generalized XML Tree Pattern Processing", *Springer Journal*, 2013,pp-120-145.

[4] M.Hachicha, "A survey on XML Tree Pattern", *IEEE Transactions on Knowledge and Data Engineering,* 2013.

[5] Choi.B, Mahoui.M and Wood, "On the optimality of the holistic twig join algorithms", *Proc. of DEXA*, 2003,pp. 28-37.

[6] Ling.T.W, Chan.C and Chen.T, "On efficient processing of CML Twig Pattern Matching", in VLDB, 2007, pp.193-204.

[7] Chen Wang.J,Naughton, "On supporting containment Queries in relational data base management systems", Proc.of SIGMOD Conference, 2006, pp.274-285.

[8] Chien.S, Vagena.Z, Zhang.D, and Tsotras, "Efficient structural joins on indexed XML documents", Proc .of VLDB, 2002, pp.263-274.

[9] Jianhua Feng, "Efficient fuzzy type ahead searches in XML data",IEEE Transactions on Knowledge and Data Engineering, 2012.

[10] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava, "Structural joins: A primitive for efficient XML query pattern matching*", In ICDE, February 2002, pp. 141–152.*