International Journal of Computer Science and Engineering Communications
Volume.6, Issue.1 (2018): Page.1794-1801
www.ijcsec.com

An Implementation of Tree Match Algorithm for Effective Xml Tree Pattern Matching

Balamurugan.V
[1]*Assistant Professor,*
Department of Computer Science and Engineering,
INFO Institute of Engineering, India
balamuruganvsrit@gmail.com[1]

Abstract— *XML (Extensible Markup Language) has become a popular standard for storing and sharing data across various platforms. The emergence of XML promised significant advances in B2B (Business-to-Business) integration. Due to its popularity there is an increasing demand for the efficient query processing on XML File. For performing query processing operations on XML file an input XML Dataset is required. Such an XML files are viewed as an XML Tree using XML DOM Parser. The core operation of our project is to perform Pattern Matching in XML Tree. The Existing XML Tree Pattern Matching Techniques uses XQuery, XPath and TwigStack Algorithm. But XQuery and XPath are complicated to understand by non-database users. In the proposed system Keyword Search Technique and TreeMatch algorithm is used to perform exact pattern matching for text, images and audio files. An XML Search engine is created to achieve this. The downloading time of images and audio files are compared with the local search engine. It is shown that XML Search engine takes less downloading time.*

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

Citation: Balamurugan.V. (2018). An Implementation of Tree Match Algorithm for Effective Xml Tree Pattern Matching. *International Journal of Computer Science and Engineering Communications,* 6(1), 1794-1801. Article ID 6117941801.

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 becomes query processing little bit complicated. In proposed system we are using keyword query to perform query answering. A 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

J.T.Yao, M.Zhang [3] have proposed 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
2. The intermediate path solutions in first step are merge-joined to produce the final solutions

Xiaoying Wu, Stefanos Souldatos [11] 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.[15] 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 [10] have 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. [19] 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 [1] have 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.

Y. Chen and D. Che [14] 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.

L.V.S. Lakshmanan[4] 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 [19] 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.

H.V. Jagadish et al. [19] have proposed a containment labeling scheme to process twig queries. Containment labeling scheme for twig Pattern processing decomposes a twig pattern into set of binary relationships which can be either P-C or A-D. Each binary relationship is processed using structural join techniques and the final match results are obtained by merging individual binary join results together. The main problem with the above solution is that it may generate large and possibly unnecessary intermediate results because the join results of individual binary relationships may not appear in the final results.

III. EXISTING SYSTEM

Existing System uses TwigStack algorithm for performing pattern matching in an XML File. TwigStack Algorithm supports XQuery and XPath only. 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. 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

IV. PROPOSED SYSTEM

In proposed system keyword and TreeMatch algorithm is used for performing exact pattern matching. Our input XML File is represented as a Tree using DOM Parser. An XML Search engine is created which gets the input query and performs pattern matching using an effective XML Tree Pattern Matching algorithm TreeMatch. TreeMatch algorithm is based on Extended Dewey Labeling concept. The input query matches with the Extended Dewey label and completes query processing. In proposed system we are performing pattern matching for text, images, audio and video files and the downloading time of audio and video files are computed. The downloading time of audio and video files are compared with local search engine. It is shown that XML search engine takes less downloading time. The concept of the TreeMatch Algorithm is given 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. The major advantage of introducing the TreeMatch Algorithm is to solve sub-optimality problem and to reduce the answering time of the queries.

The Proposed system is implemented by using the following modules:

Admin:

1. Insertion and deletion of data
2. Creation of an XML File

User:

1. Viewing XML Tree
2. XML Tree Pattern Matching
3. Comparison Module

In Admin module we are creating an XML File. After the successful creation if an XML File user can login with their own id and password. Users can view the XML Tree from the selected XML File. Then it is easy for the user to perform pattern matching for text, images, and audio files. Finally downloading time of audio and image files are compared with local search engine and it is proved that XML Search engine takes less time.

V. EXPERIMENTS AND RESULTS

We have implemented all tested algorithms in J.D.K 1.6 using the file system as a simple storage. We conducted all the experiments on a computer with Intel Pentium IV 1.7GHz CPU and 2G of RAM.

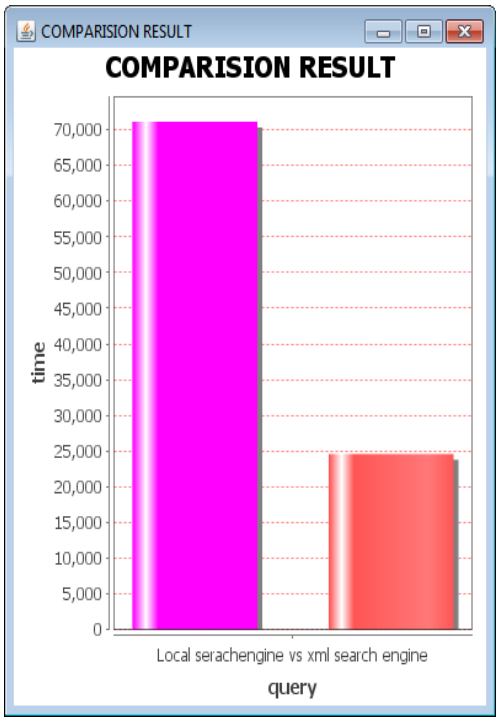

Fig 1 The downloading time of an audio file in XML Search engine is compared with Local Search engine

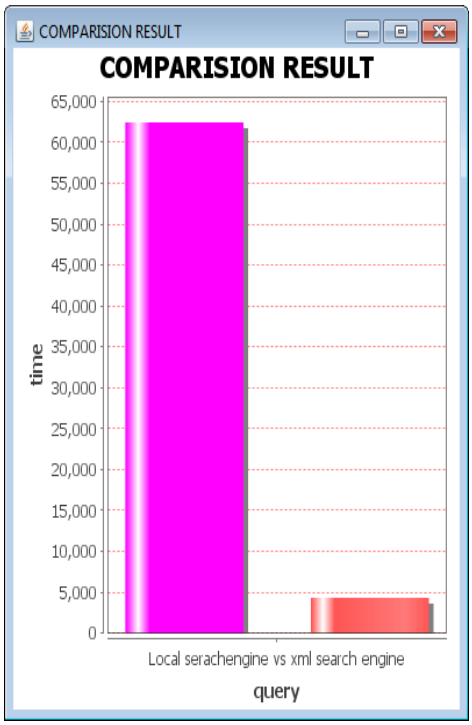

Fig 2 The downloading time of an image file in XML Search engine is compared with Local Search engine

The concept of TwigStack algorithm has been tested using a Tool called XPath Builder. The Tree Match algorithm has been implemented using Java Programming language. The front end is java and Back end is MySQL database. All the proposed modules have been implemented and the analysis of our project is shown in the bar chart given below. In the Bar graph X-Axis is Query and Y-axis is time in milliseconds.

REFERENCES

1. Wen-Chiao, Hsuand I. H. Kasimoglu, "A Compacted Indexing Scheme for efficient query evaluation of XML Documents " Elsevier Journal., vol. 241, no. 4, pp. 195-211, Mar. 2013.

2. J. Hidders, "Satisfiability of XPath Expressions " Proc. Ninth Int'l Workshop Database Programming Languages (DBPL '03), pp. 21-36, Oct.2004

3. J. Yao and M. Zhang, "A Fast Tree Pattern Matching Algorithm for XML Query" Proc. IEEE/WIC/ACM Int'l Conf. Web Intelligence (WI '04), pp. 235-241, Jan. 2004.

4. L.V.S. Lakshmanan, "XML Tree Pattern, XML Twig Query" Encyclopedia of Database Systems, pp. 3637-3640, Springer, 2009.

5. Mirjana Mazuran, Elisa Quintarelli, and Letizia Tanca, "Data Mining for XML Query Answering Support" IEEE Transactions on Knowledge and Data Engineering, pp.1393-1407,2012.

6. D. Beech, A. Malhotra, and M. Rys, "A Formal Data Model and Algebra for XML" technical report, W3C XML Query Working Group Note, 1999.

7. L.V.S. Lakshmanan, G. Ramesh, H. Wang, and Z.J. Zhao, "On Testing Satisfiability of Tree Pattern Queries" Proc. 30th Int'l Conf. Very Large Data Bases (VLDB '04), pp. 120-131, 2004.

8. L. Quin, "Extensible Markup Language (XML)" World Wide Web Consortium (W3C), http://www.w3.org/XML/, 2006.

9. W. Wang, H. Wang, H. Lu, H. Jiang, X. Lin, and J. Li, "Efficient processing of XML path queries using the disk-based F&B index" in VLDB, pages 145–156, 2005.

10. Jaihaeng Lu, "XML Tree Pattern Matching: Theories and Algorithms" IEEE Transactions on Knowledge and Data Engineering, pp.1393-1407, June, 2012.

11. Xiaoying Wu, Stefanos Souldatos, "Extended XML Tree Pattern Matching" Data and Knowledge Eng., vol. 64, no. 3, pp. 580-599, 2011.

12. R. Goldman and J. Widom, "Data Guides: Enabling Query Formulation and Optimization in Semi structured Databases" Proc. 23rd Int'l Conf. Very Large Data Bases, pp. 436-445, 1997.

13. R. Baca, M. Kra´tk1y, and V. Sna´sel, "On the Efficient Search of an XML Twig Query in Large Data Guide Trees" Proc. 12th Int'l Database Eng. and Applications Symposium (IDEAS '08), pp. 149-158, 2008.

14. Y. Chen and D. Che, "Efficient Processing of XML Tree Pattern Queries" Journal of Advanced Computational Intelligence and Intelligent Informatics, vol. 10, no. 5, pp. 738-743, 2006.

15. Li et al. "Queries and Computation on the Web" Theoretical Computer Science, vol. 239, no. 2, pp. 231-255, 2000

16. C.Y. Chan, W. Fan, P. Felber, M.N. Garofalakis, and R. Rastogi, "Tree Pattern Aggregation for Scalable XML Data Dissemination" Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02), pp. 826-837,2002.

17. J.D. Ullman, Principles of Database and Knowledge-Base Systems, vol. 1. Computer Science Press, 1988.

18. Y. Chen and D. Che, "Minimization of XML Tree Pattern Queries in the Presence of Integrity Constraints" Journal of Advanced Computational Intelligence and Intelligent Informatics, vol. 10, no. 5, pp. 744-751, 2006.

19. 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," Proc. 18th Int'l Conf. Data Eng. (ICDE '02), pp. 141- 149, 2002.

20. Giorgio Busatto, "Efficient Memory Representation of XML Documents" Proc. 15th Int'l Conf. Database Systems for Advanced Applications (DASFAA'10), pp. 170-178, 2010.

21. Sravan Kumar , Madhu "Efficient Handling of XML Tree Pattern Matching Queries – A Holistic Approach" International Journal of Advanced Research in Computer and Communication Volume 1 Issue 8, Oct 2012

22. Xiaoying Wu, Stefanos Souldatos, "XML Tree Pattern Processing Algorithms" Data and Knowledge Eng., vol. 64, no. 3, pp. 580-599, 2011.