An Effective Matching of Tree Patterns with XML Queries

Balamurugan.V[1], Sreeraj.S[2], Uthayakumar J[3]

[1]*Assistant Professor, Department of Computer Science and Engineering, Akshaya College of Engineering and Technology, Affiliated to Anna University Chennai.*

[2]*Assistant Professor, Department of M.Tech Computer Science and Engineering, Sri Krishna College of Engineering and Technology, Coimbatore*

[3]*Assistant Professor, Department of Computer Science and Engineering, Hindusthan Institute of Technology, Coimbatore*

balamuruganvsrit@gmail.com[1], sreerajs@skcet.ac.in [2], uthayakumar.j@hit.edu.in[3]

Abstract

Extensible Markup Language, or XML, is gaining importance as a standard for transferring and storing information across numerous platforms. Major improvements in B2B (Business-to-Business) connectivity were promised by the arrival of XML. Because of its widespread adoption, there is a rising demand for efficient XML file query processing. In order to process queries on a file with XML, an input XML dataset must exist. XML DOM the parser is used to display such a file as an XML Tree. Our project's primary function is to match patterns in XML trees. The TwigStack Algorithm, XQuery, and XPath are employed in the modern XML Tree Pattern Matching approaches. However, XQuery and XPath are difficult for non-dba users to comprehend. Exact pattern matching for text, image, and audio data is accomplished in the suggested system through the use of the TreeMatch algorithm and Keyword Search Technique. This is accomplished by creating an XML search engine. Images and audio files are compared with the local search engine in terms of download times. The results indicate that the XML Search engine has a faster download time.

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

I. INTRODUCTION

The process of analyzing data from multiple perspectives and compressing it into useful details is known as data mining. Mining is employed in this project to acquire information regarding an important amount of XML datasets. XML is now a language used on numerous devices for information sharing, keeping, and transfer. DOM Parser can be used to represent XML texts as a tree structure. The main functions of DOM parser are to save, retrieve, and work with XML trees. Traditional XML query languages to query XML data are XQuery (XML Query Language) and XPath (XML Path Language). The queries are answered by our current system utilizing these query languages. To handle queries in these query languages, certain complicated notations are needed. Even though they are robust, XQuery and XPath are not user-friendly for novices. The TwigStack Algorithm is used by the current system to answer queries.

However, questions including P-C (Parent-Child) and A-D (Ancestor-Descendant) links can be answered using the TwigStack algorithm. Processing queries gets a little more difficult as a result. To answer queries, the suggested method uses keyword queries. Using TreeMatch, an XML Tree Pattern Matching Algorithm, the current system's sub-optimality issue is resolved. The expanded Wey Labeling concept is the foundation of this algorithm. The base node, kids, and grand grandchildren are connected to the number or label that complies with the Naming System For example, the root node has the number 0. The offspring of the root are assigned labels like 0.0 and 0.1. The first parent the node's grandchildren begin at 0.0.0 and go up to 0.0.1, etc.

II. RELATED WORK

In order to process XML queries, Maurice Tchoupé Tchendji et al. [3] have presented holistic algorithms. TwigStack is a unique, all-encompassing XML twig pattern matching technique that avoids storing intermediate results unless they are important to the ultimate outcome. This method's main benefit is that it eliminates the need to compute a lot of unnecessary intermediate outcomes. However, TwigStack's primary drawback is that, in cases when searches involve parent-child relationships, it may yield a sizable number of "useless" intermediate answers. TwigStack's effectiveness has only been demonstrated for A-D edge queries; for P-D edge queries, it is still unable to regulate the amount of the intermediate results. TwigStack functions in two phases:

1. The intermediate results are a list of intermediary route solutions.
2. To create the final solutions, the intermediate path answers from the first stage are merge-joined.

The MPMGJN (multi-predicate Merge-Join) technique was proposed by Mulajkar Sneha Digambarrao [11]. Generally, this algorithm comprises of the following processes: decomposition-matching and combining:

- Divide down the tree pattern into straight trends, such as root-to-leaf pathways or binaries (parent-child or ancestor-descendant) associations among nodes pairs.
- For the best results, find each matching in each linear pattern then merge-join them.
- The primary way that MPMGJN varies in comparison to the TwigStack merge join method is that it necessitates numerous input list searches.

The Stack-Tree Algorithm was suggested by Li et al. and Chien et al. [15] and is primarily used to address the shortcomings of the MPMGJN algorithm. The primary disadvantage of the MPMGJN method is that it requires many input list tests, whereas the Stack-Tree approach needs just a single scan. Stacks are used by the Stack Tree method to preserve the original or grandparent nodes. The P-D and A-D sides may be employed to implement the Stack Tree Algorithm.

The OrderedTJ Algorithm was proposed by Zhixue He [10] and is mainly employed to tackle the weaknesses of decomposition-matching-merging algorithms.

An element is only beneficial to the final results of the OrderedTJ algorithm if the order of its progeny match the order of the associated query nodes. A branching edge which links to the nth child is referred to as the nth branching edge if we define the edges that divide branching nodes and the children they produce as branching edges. All sequential algorithms that read the complete input have an I/O the greatest efficiency that equals OrderedTJ. declared differently, OrderedTJ's optimality permits parent-child edges to exist in non-branching edges as well as the first branching edge. Despite generating a lot fewer intermediate results, the OrderedTJ algorithm's size rises linearly with database size.

The TJFast algorithm was created by Dinh Duc Luong and Vuong Quang Phuong [19] with the goal solve the weaknesses of the solitary confinement labeling system. Although the restricted labeling method preserves the positional data within an XML document's hierarchy, there are a few disadvantages to the strategy.

- A single confinement label can only hold a very small amount of information. For instance, no single containment label can provide us with path information.
- Wildcards are frequently used in XPath, but the confinement label system is unable to accommodate them.

It is challenging to respond to requests using wildcards in branching nodes using the confinement label technique. When a query has many return nodes, TJFast is unable to produce an individual address for each node. Negation procedures and ordered limitations are unsuitable with TJFast.

The CSI-X methodology was developed by Mustapha Machkour and Hassana Nassiri [1] to expedite the screening of questions in XML documents. CIS-X is mainly employed to process XML Path expressions while overcome the limitations of decomposition-matching-merging algorithms. A query is broken into many sub-queries using decomposition-matching-merging techniques. Each sub-query is executed separately, and the intermediate results are saved for later processing. These methods nevertheless still have the drawback of requiring an extended period to combine and offering large intermediate outcomes. Consequently, the CIS-X strategy, which enables advanced XQueries, has been proposed in the present research. However, the CIS-X Technique's shortcoming is that index developing takes more time.

Twig Square Stack is an innovative method indicated by Mulajkar Sneha Digambarrao [11] that is primarily employed for eliminating merging expenses in the second phase. Twig Square Stack is a single-phase method that effectively handles path matching while avoiding the costly merging phase. A basic enumeration function can be utilized to output the final solutions that are stored as hierarchical stacks of the overall solutions. But sustaining the data structures is too expensive and challenging.

TwigList, an updated version of Twig Square Stack, is a technique recommended by L.V.S. Lakshmanan [4]. It uses a much simpler data structure—a set of lists—to store solutions. TwigList has identical disadvantages as Twig Square Stack but some advantages. One disadvantage is that, even though some of the potential nodes for QP (Query Processing) are not essential to the solution, it will all be added to and erased from the temporary stack. Another drawback is that they are less successful at removing unnecessary nodes.

For the purpose of processing twig pattern matching, Dinh Duc Luong and Vuong Quang Phuong [19] implemented Structural Join techniques.

A twig query can be broken down into a lot binary P–C or A–D connections in the first step. Each binary sub-query receives separate assessments, giving its intermediate result. In the subsequent phase, these intermediate results are combined to generate the end product. Many intermediate results are produced by this approach, some of which might not make it into the final product. Furthermore, the merging step is costly.

A confinement labeling method has been proposed by Dinh Duc Luong and Vuong Quang Phuong [19] to handle twig queries. Twig containment labeling scheme A twig pattern is broken down into a number of binary relationships—P-C or A-D—by pattern processing. The final match results are obtained through the combination of the individual binary join results. Each bilateral relationship is evaluated utilizing structurally join techniques. The primary issue with the aforementioned technique is that, since the join outcomes for individual binary relationships might not show up in the end results, it could result in a lot of possibly unnecessary intermediate results.

The TJFast method was suggested by Al-Khalifa et al. [23] in order to address the shortcomings of the confinement labeling system. Although the confinement labeling method maintains the positional information within an XML document's hierarchy, there are a few drawbacks to the approach.

- A single confinement label can only hold a very small amount of information. For instance, no single containment label can provide us with path information. Wildcards are frequently used in XPath, but the confinement label system is unable to accommodate them.
- It is challenging to respond to requests using wildcards in branching nodes using the confinement label technique. When a query has numerous return nodes, TJFast does not generate a unique answer for each node. Negation functions and ordered restrictions are incompatible with TJFast.

III. EXISTING SYSTEM

The TwigStack technique is used by the current system to match patterns in an XML file. Only XQuery and XPath are supported by the TwigStack Algorithm. P-C and A-D relationship questions are answered using the TwigStack Algorithm. (/) indicates P-C edges, and (//) indicates A-D edges. An approach for matching and merging decompositions is called the TwigStack approach. This algorithm breaks down a query into many sub-questions. The interim results of each sub-query are kept for later processing and are conducted independently. These intermediate outcomes are combined to produce the final result. For queries with P-C relationships, the TwigStack Algorithm returns meaningless intermediate results; for queries with A-D links, it regulates the size of the intermediate result. The following describes the TwigStack algorithm:

// First Phase

1: while notEnding (q)

2: getNext (q) = qact

Step 4: cleanStack (parent (qact), nextL (qact)) if (isNotRoot (qact))

5: terminate if

Sixth: if (isNotEmpty (Sparent (qact)) or isRoot (qact)) 7: cleanStack (qact, next (qact)) follows.

8: moveStreamToStack (pointertotop (Sparent (qact)), Sqact, and Tqact))
10: showSolutionsWithBlocking (Sqact, 1) if (is Leaf (qact))
11: pop (Sqact)
12-end if 13: otherwise
14: in advance (Tqact)
15: stop if 16: stop while
// Stage Two
17: call combineAllPathSolutions
Algorithm Two stages make up TwigStack's operation. Some (but not all) of the root-to-leaf path solutions for each individual query are computed in the first phase (lines 1–16). These solutions are merge-joined to calculate the query twig pattern replies in the second phase (line 17). Below is a description of the principal shortcomings of the current system:

- XQuery and XPath are difficult for non-database people to understand.
- They are not user-friendly for non-expert users.
- Using XQuery and XPath makes query answering a little more difficult.
- The TwigStack algorithm is unable to limit the amount of pointless intermediate responses.

IV. PROPOSED SYSTEM

The TreeMatch algorithm and keywords are utilized in the approach suggested to match patterns precisely. DOM Parser takes our input XML file to represent it as a Tree. An XML search engine is developed that accepts the query as input and matches pattern with the help of the fast XML Tree Pattern Matching algorithm TreeMatch. TreeMatch algorithm is founded on the idea of Extended Dewey Labeling. After the input query and the Extended Dewey label match, the query processing is finished. The suggested system detects trends in text, images, audio recordings, and video files. It also calculates the duration of audio and video file downloads. Audio and video file download times are compared with those of a nearby search engine. It is demonstrated that the XML search engine downloads more quickly. The TreeMatch Algorithm's premise is as follows:
first, locateMatchLabel (Q);
second, do while(endroot)
4: if (fact is a return node)
3: fact= getNext(topBranching Node)
5: advance (Tfact);
6: addToOutputList (NAB(fact,cur(Tfact)); //view the subsequent Tfact element
7: updateSet (fact); //change the encoding of sets
Locate the next element with a path that matches
8: locateMatchLabel (Q);
9: root -> emptyAllSets;
The first element whose pathways correspond to each unique root-leaf path pattern is found in line 1. The getNext method chooses a leaf node fact for each iteration (line 3). Lines 4, 5, and 6 are used to add possible matching components to the outputlist. Line 7 modifies the set encoding, and line 6 advances the list Tfact.

The next piece that matches the individual path is found at line 8. In order to ensure that the output solutions are complete, we must empty every set in Procedure EmptyAllSets (Line 9) after processing every data.

XPath and XQuery are not necessary for the suggested system to function. After matching the given query's expanded Dewey Label, the TreeMatch Algorithm finishes processing the query. The Proposed system is implemented by using the following modules:

Administrator:
1. Inserting and removing data
2. Generating an XML file
User:
1. Seeing an XML Tree
2. Matching Patterns in XML Trees
3. Module of Comparison

We are producing an XML file in the admin module. An XML file can be successfully created, and the user can log in using their own login credentials. From the chosen XML File, users can view the XML Tree. The user can then match patterns for text, image, and audio files with ease. When the download times for audio and image files are finally compared to those of a local search engine, it can be seen that XML Search engine is faster.

Merits of Proposed System

The following are the suggested system's main benefits:

- The following benefits of the TreeMatch algorithm:
- It requires very little processing time
- It reduces superfluous intermediate results
- It matches accurate patterns with XML trees using the Extended Dewey Labeling concept
- It requires very little processing time
- The keyword search approach is easy to use even for non-expert people

V. RESULTS AND EXPERIMENTS

We used the file system as a straightforward storage mechanism to implement all of the tested algorithms in J.D.K 1.6. Every experiment was carried out on a PC equipped with an Intel Core i7 processor and 4GB of RAM. XPath Builder is a tool that has been used to test the TwigStack algorithm concept. Java programming has been used to implement the Tree Match algorithm. MySQL is the back end database, and Java is the front end. Every suggested module has been put into practice, and the bar chart below displays the project analysis. The bar graph has a time axis in milliseconds and an X-axis representing the query.

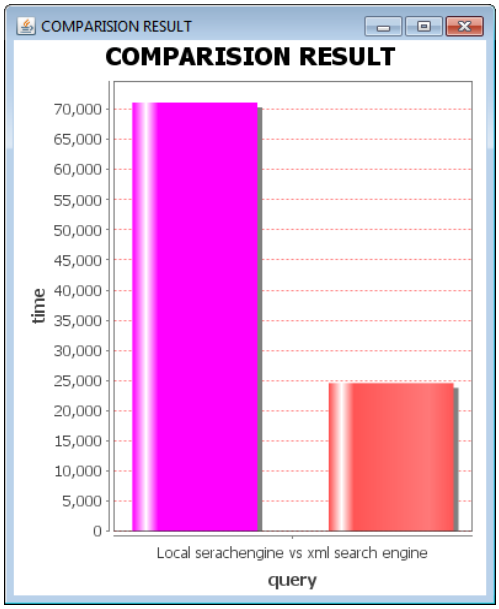

Fig. 1: Experimental Results of the TwigStack technique (The TwigStack technique takes a long time to run a query).

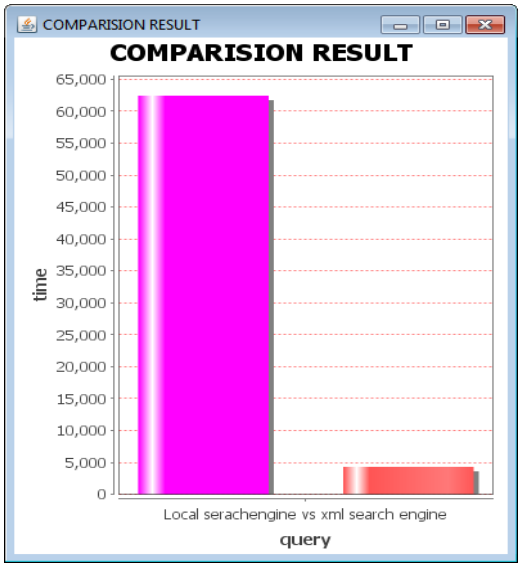

Fig. 2: execution of queries after TreeMatch Algorithm deployment

VI. Conclusion

XML is a technology for information transport that is agnostic of hardware and platforms. Thus, business-to-business (B2B) e-commerce makes extensive use of XML. Businesses now need to query XML data more frequently due to its growing popularity. XML query processing is quite challenging because of the complex notations included in XML query processing languages like XQuery and XPath for XML Tree Pattern Matching. Therefore, pattern matching in an XML tree is done via a keyword query in order to get around this. To do this, TreeMatch, an XML Tree Pattern Matching technique, is used. The outcomes of the experiment demonstrate how well the TreeMatch algorithm matches patterns in complicated audio and visual sources. Thus, it can be said that the TreeMatch method for matching XML tree patterns requires less time to answer the queries.

VII. FUTURE WORK

Pattern matching for text, image, and audio files is completed in this project. Our work will be expanded in the future to include pattern matching for large-sized video files, and the XML search engine's download times will be compared to those of several well-known search engines, such as Google and Yahoo.

References

1. *Hassana Nassiri, Mustapha Machkour "One query to retrieve XML and Relational Data " Elsevier Journal., vol. 241, no. 4, pp. 340-345 ,2018.*

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

3. *Maurice Tchoupé Tchendji, Lionel Tadonfouet, Thomas Tébougang Tchendji "A Tree Pattern Matching Algorithm for XML Queries with Structural Preferences" Journal of Computer and Communications, pp. 61-83, 2019.*

4. *L.V.S. Lakshmanan, "XML Tree Pattern, XML Twig Query" Encyclopedia of Database Systems, pp. 1-4, Springer, 2017.*

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. *Zhixue He, "An Optimization Method for XML Twig Query" International Journal of Performability Engineering , pp.1393-1407, June, 2018.*

11. *Mulajkar Sneha Digambarrao, "Extended XML Tree Pattern Matching" Journal of Emerging Technologies and Innovative Research vol. 9, no. 10, pp. 241-250, 2022.*

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. *Dinh Duc Luong, Vuong Quang Phuong "AN IMPROVED INDEXING METHOD FOR QUERYING BIG XML FILES," Journal of Computer Science and Cybernetics Vol 39, Issue 4, 2023.*

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.*

23. *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.*