www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 4 Issue 1 January 2015, Page No. 10116-10120

Review on Xml Tree Pattern Matching Using Holistic Algorithms

Miss. Bharti D. Wanjari[1] Professor Mr. Kapil N. Hande [2]
[1]RTMNU University, PBCE Nagpur, Maharashtra, India.
[1]Bharti.wanjari29@gmail.com
[2]RTMNU University, Department of Computer Science and Engineering,
PBCE Nagpur, Maharashtra, India.
[2]kapilhande@gmail.com

Abstract*: The extensible markup language XML has recently to come into view as a new standard for information representation and exchange on the internet. With XML becoming ever-present language for data interoperability purposes in various domains, efficiently querying XML data is a critical issue. XML has become a practice standard to store, share and exchange business data across similar and dissimilar platforms. The interoperability is possible though XML. As organizations are generating large amount of data in XML format, there is a need for processing XML tree pattern queries. This paper presents survey on some developments in the field of XML tree pattern query processing, especially focusing on holistic approaches. XML tree pattern query processing is a research flow within XML data management that focuses on efficient Tree Pattern Query (TPQ) answering. The existing holistic algorithms for XML tree pattern matching queries display suboptimmality problem as they consider intermediate results before taking final results. This causes suboptimal performance. This suboptimality is overcome by using TreeMatch algorithm. This paper presents the overview of prototype application that makes use of efficient Dewey labeling scheme to overcome suboptimality with TreeMatch algorithm.*

Keywords- XML, Query Processing and optimization, Holistic tree pattern\ Matching

.

I. Introduction

With XML becoming omnipresent language for data interoperability purposes in various domains, efficiently querying XML data is a extremely important issue. XML is the universal format for structured documents data on web. XML documents are used to carry over data from one place to another often over the Internet. XML is extensible markup language much like Hyper Text Markup Language (HTML). XML is not a replacement for HTML. XML is more manageable and adaptable than HTML. XML was designed to transfer data not to display data. XML and HTML were designed with different goals: XML was designed to store, transport and display required information. HTML was designed to display data, with focus on how data looks HTML is about displaying information, while XML is about transferring information.

An XML document consists of nested elements enclosed by user-defined tags shows an example of an XML document named “pubctn.xml”, which contains some publication information. The hierarchical structure of an XML documents can be modeled as a tree. Figure 2 is the tree representation of the XML file. The XML documents on the Internet are forest of XML trees and we call it an XML database.

```
<?xml version="1.0" ?>
        <publication>
                <journal title="DBMS">
                        <editor>Jack</editor>
                        <article>
                                <title>
                                Index Construction
                                </title>
                                <author>Smith</author>
                        </article>
                </journal>
                <journal title="Algorithm">
                </journal>
    </publication>
```

Due to the business alliance and for the purpose of adjustability organizations are storing data in XML format. This has become a common practice as XML is easily transported and irrespective of platforms in which applications were developed, they can share data through XML file format. Such XML files are also approved using DTD or Schema. XML parsers are available in all languages that ease the usage of XML programmatically. Besides XML is tree based and it is convenient to handle easily using Document Object Model(DOM) API. XML tree pattern queries are to be processed efficiently as that is the main operation of XML data.

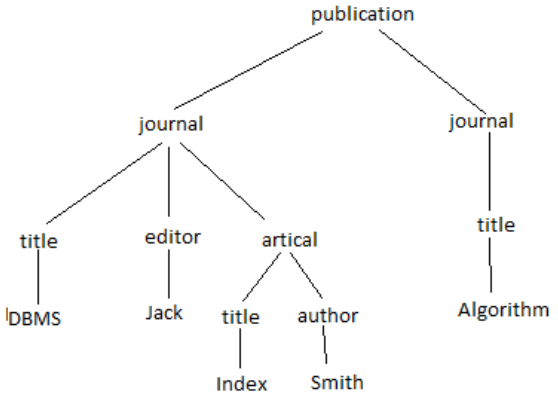


Figure 1: Tree representation of XML Document

XML has become the practice standard for storing and moving semi structured data due to its simplicity and flexibility, with XPath[1] and XQuery[2] as the standard query

languages. XML documents have tree structure, where members (tags) are internal tree nodes, and attributes and text values are leaf nodes. Information may be converted both in structure and content, and query languages need the expressional power to specify both.

With the rapidly increasing popularity of XML for data representation, there is a lot of concern in query processing over data that conforms to a *tree-structured* data model. An XML query pattern commonly can be represented as a rooted, labeled tree (Twig), for example Fig 2 shows an example XPath query:

Book [title = "JAVA"] // author [. = "chan"]

Such a complex query tree pattern can be naturally decomposed into a set of basic P-C and A-D relationship between pairs and nodes [4]. The above example query are the ancestor-descendent relationship (book, author) and the parent-child (book, title) and (title, JAVA) and (author, Chan).

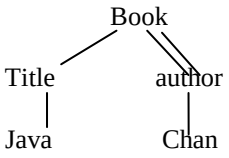


Figure 2: XPath Query

In practice, XML data may be complicated and have deep nested elements. Thus, very huge, efficiently finding all twig patterns in an XML database is a major interest of XML query processing. In the last few years, many algorithms ([4],[5]) have been proposed to match such twig patterns. These approaches (1) first develop a labeling scheme to capture the structural information of XML documents, and then after (2) perform tree pattern matching based on labels alone without traversing the original XML documents. For finding the first sub- problem of designing a proper labeling scheme, the former methods use a *tree-traversal* order or textual positions of *start* and *end* tags (e.g. region encoding [6]) or path expressions(e.g. Dewey ID [7]) or prime numbers. By applying these labeling schemes, one can determine the relationship (e.g. ancestor-descendant) between two elements in XML documents from their labels alone.

We present a fast tree matching algorithm called TreeMatch that can directly find all matching's of a tree pattern in one step. The only requirement for the data source is that the matching elements of the non-leaf pattern nodes do not contain sub-elements with the same tag. There are at least two advantages of TreeMatch. First, the TreeMatch algorithm does not need to decompose the query tree pattern, as it matches the pattern against the data source directly. Therefore, it does not produce any intermediate results and does not need the merging process. Second, the final results are compactly encoded in stacks and explicit representation of the results, either as a tree or a relation with each tuple representing one matching, can be generated efficiently.

The rest of the paper is organized as follows:

We first formally define the TP and related concepts in section II. Section III reviews the literature survey on different Xml tree pattern algorithm that gives insights into the research topic. Section IV gives the details of the holistic algorithm and comparative analysis of the different holistic algorithm while the section V conclude the paper which followed by references.

II. Background

In this section we first formally define all the concept used in this paper.

1) XML Document

XML is known to be a simple and very flexible text format. It is essentially employed to store and transfer text-type data. The content of an XML document is encapsulated within elements that are defined by tags. These elements can be seen as a hierarchy organized in a treelike structure.

2) Data Tree Collection

An XML document considered as a set of fragments may be modeled as a data tree collection (also named forest in TAX), which is itself a data tree.

3) Tree Pattern Matching

Matching a TP p against a data tree t is a function $f : p \rightarrow t$ that maps nodes of p to nodes of t such that.

- structural relationships are preserved, i.e., if nodes (x, y) are related in p through a parent-child relationship, denoted PC for short or by a simple edge/in XPath (respectively, an ancestor-descendant relationship, denoted AD for short or by a double edge // in XPath), their counterparts ($f(x)$, $f(y)$) in t must be related through a PC (respectively, an AD) relationship too;
- formula F of p is satisfied.

The output of matching a TP against a data tree is termed a witness tree in TAX.

III. XML Tree pattern matching algorithms

XML query contains two parts one is value match and another one is tree match. The above XPath query (fig 2) contains „XML" is a value match and another is a twig match. Labeling and Computing is the main view of the twig pattern, labeling assign each element in the XML document tree an integer label to capture the structural information of documents and computing use labels to answer the twig pattern without traversing the original document. Mainly there are two labeling schemes, such as containment labeling schemes and Dewey ID labeling schemes. Several algorithms based on the containment labeling scheme have been developed to process twig queries.

In the environment of semi-structured and XML databases, tree-based query pattern is a very practical and important class of queries. The recent papers (e.g. [9,10]) are proposed to efficiently process an XML twig pattern. Lore DBMS [11] and Timber [12] systems have considered various appearance of query processing on such data and queries. XML data and various issues in their storage as well as query processing using relational database systems have recently been considered in [7, 8]. In paper [11], a new holistic algorithm, called OrderedTJ, is proposed to process order-based XML tree query. In paper [12], an algorithm called TwigStackListNot is proposed to handle queries with negation function. Chen et al [13] pro- posed different data streaming schemes to boost the holism of XML tree pattern processing. They showed that bigger optimal class can be accomplish by refined data streaming schemes. In addition, Twig2Stack [14] is proposed for answering generalized XML tree pattern queries. Note the difference between *generalized* XML tree pattern and *extended* XML tree pattern here. Generalized XML tree pattern is defined to include optional axis which models the expression in LET and RETURN clauses of XQuery statements. But extended XML tree pattern is defined to include some intricate conditions like negative function, wildcard and order restriction.

In addition the holistic algorithms, there are other approaches to match an XML tree pattern, such as ViST ([15]) and PRIX ([16]), which transform an XML tree pattern match to sequence match. Their algorithms mainly focus on ordered queries, and it is non- trivial to extend those methods to manage unordered queries and extended queries studied in this article. Note that the paper [17] made exhaustive experiments to compare different XML tree query processing algorithm and concluded that the family of holistic processing methods, which provides performance guarantees, is the most strong approach. From the aspect of theoretical research about the optimality of XML tree pattern matching, Choi et al. [7] developed theorems to prove that it is impossible to devise a holistic algorithm to guarantee the optimality for queries with any fusion of P-C and A-D relationships.

Most of these works created on some labeling scheme of XML elements to facilitate the verification of the structural relationship. The most commonly used labels are the *containment* and *prefix* labeling scheme. The *containment* labeling was introduced by Zhang et al. [18] to facilitate the containment queries. The verification of *ancestor -descendant* structural relationship is of the same complexity as that of *parent-child* relationship by using regional labeling. Dewey *ID* is the *first* example of using prefix labeling to represent XML data. It can be used to preserve the path information during query processing. Recent work of Lu at el.[16] utilize the *extended Dewey* encoding which encodes path information including not only the element IDs but also the element names.

The following sections going to comparative analysis about some existing tree pattern matching techniques in specifically TwigStack, TJFast with TreeMatch [19][21][22].

IV. Different Holistic Algorithm for XML query processing

The following sections we have going to comparative analysis about few existent tree pattern matching techniques in particularly the holistic algorithm on real-life and synthetic data sets, including TreeMatch [22], TwigStack [19], TJFast [21].

I. TwigStack Algorithm

TwigStack [19] was the first holistic twig join algorithm. Using PathStack on each root-to-leaf path in a twig query and merging the matches, may lead to many useless intermediate results, because matches need not be part of complete matches. TwigStack improved on this, and achieved $O(I + O)$ complexity for queries with a-d edges only. When all edges in query pattern are ancestor – descendant (A-D) relationships, Twigstack ensures that each root–to–leaf intermediate solution is merge – joinable. TwigStack has been proved to be I/O optimal in terms of output sizes for queries with only A-D edges, their algorithms still cannot control the size of intermediate results for queries with parent-child (P-C) edges. To get a better understanding of this limitation, let us take an experimented with TreeBank datasets tested three twig queries patterns, each of which contains at least one Parent-Child (P-C) edge. TwigStack operates two steps: 1. a list of intermediate path solutions is output as intermediate results and 2. the intermediate path solutions in the first step are merge-joined to produce the final solutions.

Table 1: number of intermediate path solutions produced by TwigStack against treebank data

Query	Output result	Useful path	Useless path
VP[./DP]//PRP_DOLLER	10673	6	98.9%
S[./JJ]NP	70899	11	99.9%
S[.//VP/IN]//NP	703291	22565	96.8%

An immediate observation from the table 1 is that TwigStack resuls many intermediate paths that are not merge-joinable. For all three queries, more than 95% intermediate paths produced by TwigStack in the first step are "useless" to final answers [23]. The main reason for such bad performance is that in the TwigStack, it assumes that all edges in queries are A-D relationships and therefore output many useless intermediate results when queries contain P-C relationships. TwigStack cannot answer queries with wildcards in branching nodes.

For example in Fig 3, the parent of B should be an ancestor of C

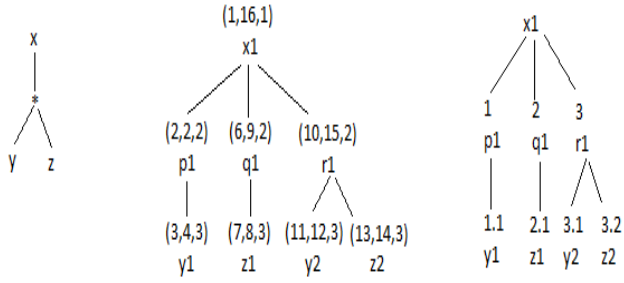
Figure 3: queries with wildcard

II. TJFast Algorithm

I have presented a holistic algorithm for answering XML twig queries in previous sections. Interestingly, that algorithm uses the same containment labeling scheme. While the containment scheme preserves the positional information within the hierarchy of an XML document, we observe that this is not the only labeling scheme that can be used for XML twig query processing. Certainly, there are at least two limitations in the containment scheme.

1. The information contained by a single containment label is very limited. For example, we cannot get the path information from any single containment label.
2. While wildcard steps in XPath are commonly used when element names are unknown or do not matter.

The containment labeling scheme is complex to answer queries with wildcards in branching nodes. For example, consider an XPath: "//x/*/[y]/z". where "*" denotes a wildcard symbol which can match any single element. The containment labels of x, y and z do not provide enough information to figure out whether they match the query or not. This is because even if y and z are descendants of x and their level difference with x is 2, y and z may not be query answers, as they do not have the common parent.



(a)query (b)containment (c)Dewey ID

Figure 4: wildcard query processing

However, Dewey ID labeling scheme can efficiently overcome the above two limitations. In Dewey ID, each element is labeled by a vector to show the path from the root to this element. This example shows that unlike containment, the Dewey ID labeling scheme can provide path information and thus support the evaluation of queries with wildcards in branching nodes. TJFast outputs one useless intermediate path and it is outputs the path solution for all nodes in query. It does not produce the individual solution for each node when there are multiple return nodes in a query. TJFast cannot work with ordered restriction and negation function.

III. TreeMatch Algorithm

Previous XML tree pattern matching algorithms do not fully exploit the "optimality" of holistic algorithms. TwigStack guarantees that there is no useless intermediate result for queries with only AD relationships. Therefore, TwigStack is *optimal* for queries with only A-D edges. Previous algorithms focus on XML tree pattern queries with only P-C and A-D relationships. Little work has been done on XML tree queries which may contain wildcards, negation function and order restriction, all of which are frequently used in XML query languages such as XPath and XQuery. In this analysis, we take an XML tree pattern with negation function, wildcards and/or order restriction as *extended XML tree pattern*. Fig 5, for example, shows four extended XML tree patterns. Query (a) includes a wildcard node "*", which can match any single node in an XML database. Query (b) includes a negative edge, denoted by "¬".

Here we have three categories of XML tree patterns (twigs) in Fig. 5.

- Q/,//,* means queries with P-C,A-D relationships and wildcards. Here "/" denotes Parent-Child (P-C) relationship, "//" denotes Ancestor-Descendant (A-D) relationship and a wildcard "*" means it can match any single node in an XML database.
- Q/,//,*,< means queries with P-C, A-D relationships, wildcards and order restriction. Here "<" shows that the nodes are ordered.
- Q/,//,*,<,¬ means queries with P-C, A-D relationships, wildcards, order restriction and negation function. Here "¬" represents negation function.

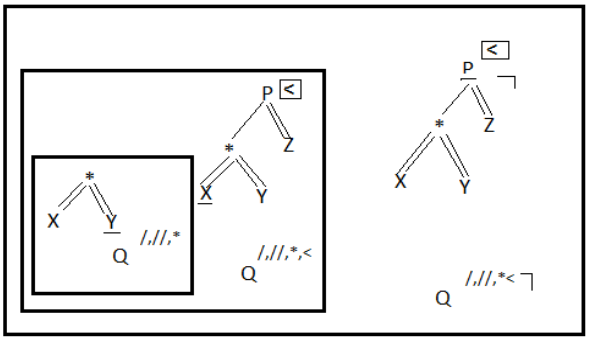


Figure 5: Examples for XML tree Patterns

The TreeMatch algorithm is proposed to achieve optimal query classes. It uses a concise encoding technique to match the outputs and also reduces the useless intermediate outputs. Most XML query processing algorithms on XML documents rely on certain labeling schemes, such as region encoding scheme [18], prefix scheme [24], ORDPATH [25], Dewey scheme [7]. In this paper, we use the Dewey labeling scheme, proposed in paper [7], to assign each node in XML documents a sequence of integers to capture the structure information of documents. Dewey labeling scheme is a derived scheme of the prefix labeling scheme. In the prefix labeling scheme, the root is labeled by an empty string and for a non-root element u, label (u) = label (v).n, where u is the nth child of v. In Dewey labeling scheme [7], each label gives complete information about ancestors" names and labels. For example, given an element e with label "1.2.3", prefix labeling schemes can tell us parent(e)="1.2" and grandparent(e)="1", but Dewey labeling scheme can also tell us the tag name of elements, say, tag(e)=„X", tag(parent(e))=„Y" and tag(grandparent(e))=„Z ". In order to achieve this goal, paper [9] uses module function to encode the element tag information to prefix labels, and use finite state transducer (FST) to decode the type's information for a single extended Dewey label. The complete path information in Dewey labels enables holistic algorithms to scan only leaf query nodes to answer an XML query.

Through this survey, we illustrate two differences between TJFast and TreeMatch. (1) TJFast outputs one useless intermediate and TreeMatch uses the bitVector encoding to solve this problem. (2) TJFast outputs the path solution for all nodes in query, but TreeMatch only outputs nodes for return nodes (i.e. node B in the query) to reduce I/O cost.

Analogous analysis table of previous algorithms with TreeMatch

Table 2: overall summary of algorithm analysis

Algorithm	Labeling scheme	Optimality	Query	Result
TwigStack	Containment	optimal in terms of output sizes and not optimal for PC	Unordered	Many useless intermediate results when queries contain P-C relationships
TJfast	Dewey Labeling	Not fully optimal	Unordered	one useless intermediate path and it is outputs the path solution for all nodes in query
TreeMatch	Dewey labeling and bitvector	Fully optimal	Wildcard, Negation, Order restriction	No useless path

Based on previous detailed discussions, table 2 illustrates the analogous analysis of previous tree pattern matching algorithms with TreeMatch with the key factors of labeling schemes, optimality, query and result.

V. Conclusion

In this paper, we proposed the problem of XML tree pattern matching and surveyed some recent works and algorithms. The previous twig pattern matching algorithms (*TwigStack, TwigStackList, OrderedTJ, and TJFast*) requires bounded main memory for small queries and requires more features than TreeMatch algorithm. TreeMatch has an overall good performance in terms of labeling schemes, optimality, query processing, result (table 2) and the ability to process extended XML tree patterns (twigs). TreeMatch to achieve such optimal query classes so, from this points we can say that TreeMatch twig pattern matching algorithm can answer complicated queries and has good performance.

References

[1] A. Berglund, S. Boag, and D. Chamberlin. XML path language (XPath) 2.0. W3C Recommendation 23 January 2007 http://www.w3.org/TR/xpath20/.

[2] S. Boag, D. Chamberlin, and M. F. Fernandez. Xquery 1.0: An XML query language. W3C Working Draft 22 August 2003.

[3] Marouane Hachicha and Je´ roˆme Darmont, Member, IEEE Computer Society *"A Survey of XML Tree Patterns*" IEEE Transactions On Knowledge And Data Engineering Vol:25 No:1 Year 2013

[4] H. Jiang, H. Lu, and W. Wang. "*Efficient processing of XML twig queries with OR-predicates*". In *Proc. of SIGMOD Conference*, pages 274{285, 2004.

[5] H. Jiang et al. "*Holistic twig joins on indexed XML documents*". In *Proc. of VLDB*, pages 273{284, 2003.

[6] N. Bruno, D. Srivastava, and N. Koudas. "*Holistic twig joins: optimal XML pattern matching*". In *Proc. of SIGMOD Conference*, pages 310{321, 2002.

[7] I. Tatarinov, S. Viglas, K. S. Beyer, J. , E. J. Shekita, and C. Zhang:. *"Storing and querying ordered XML using a relational database system"*. In *Proc. of SIGMOD*, pages 204{215, 2002.

[8] X. Wu, M. Lee, and W. Hsu. "*A prime number labeling scheme for dynamic ordered XML trees*". In *Proc. of ICDE*, pages 66{78, 2004.

[9] J. Lu, T. W. Ling, T. Yu, C. Li, and W. Ni. "*Efficient processing of ordered XML twig pattern matching*". In *DEXA*, pages 300{309, 2005.

[10] T. Yu, T. W. Ling, and J. Lu. Twigstacklistnot: "*A holistic twig join algorithm for twig query with not-predicates on xml data*". In *DASFAA*, pages 249{263, 2006.

[11] R. Goldman and J. Widom. Dataguides: "*Enabling query formulation and optimization in semistructured databases*". In *Proc. of VLDB*, pages 436{445, 1997.

[12] H. V. Jagadish and S. AL-Khalifa. Timber: "*A native XML database*". Technical report, University of Michigan, 2002.

[13] 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.

[14] S. Chen, H.-G. Li, J. Tatemura, W.-P. Hsiung, D. Agrawal, and K. S. Candan.Twig2stack: Bottom-up processing of generalized-tree-pattern queries over xml document. In *Proc. of VLDB Conference*, pages 19{30, 2006.

[15] H. Wang, S. Park, W. Fan, and P. S. Yu. ViST: "*A dynamic index method for querying XML data by tree structures*". In *SIGMOD*, pages 110{121, 2003.

[16] P. Rao and B. Moon. PRIX: "*Indexing and querying XML using prufer sequences*". In *ICDE*, pages 288{300, 2004.

[17] M. Moro, Z. Vagena, and V. J. Tsotras. "*Tree-pattern queries on a lightweight XML processor*". In *VLDB*, pages 205{216, 2005.

[18] C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, and G. M. Lohman. "*On supporting containment queries in relational database management systems*". In *Proc. of SIGMOD Conference*, pages 425{436, 2001.

[19] N. Bruno, D. Srivastava, and N. Koudas, *Holistic twig joins: optimal XML pattern matching, In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2002, pp. 310–321.*

[20] J. Lu, T. W. Ling, T. Yu, C. Li, and W. Ni. *Efficient processing of ordered XML twig pattern matching. In DEXA, pages 300–309, 2005.*

[21] J. Lu, T. Chen, and T. W. Ling. *TJFast: Efficient processing of XML twig pattern matching. Technical report, National university of Singapore, 2004.*

[22] J. Lu, T. W. Ling, Z. Bao, and C. Wang. *Extended xml tree pattern matching: theories and algorithms. IEEE transactions on knowledge and data engineering, vol.23, no. 3, march 2011*

[23] Lu Jiaheng, "*Efficient Processing Of Xml Twig Pattern Matching*", doctoral diss., Shanghai Jiao Tong University, China, 2006

[24] Q. Li and B. Moon. "*Indexing and querying XML data for regular path expressions*". Proceedings of the 27th VLDB Conference, pp361-370, 2001.

[25] 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.