XML Twig Pattern Matching Processing

Yi Mon Thet
University of Computer Studies, Yangon
yimonthet.ucsy@gmail.com

Abstract

Finding all occurrences of a twig pattern query is a core operation of XML query processing system. Many algorithms have been proposed. Existing algorithms generate individual path matches and then stitch them together through a merge operation to form twig matches. Due to the merging operation they have very expensive cost. To avoid the merging cost, many one phase algorithms have been proposed. However, these algorithms are quite efficient for queries with only ancestor-descendent relationship. TwigMMatch algorithm is proposed. Like Holistic TwigStack, a chain of linked stacks is applied to process the twig matches. The proposed algorithm not only avoids the merging process but also process the twig queries with mixed edges.
Keywords: a twig pattern query, parent-child and ancestor-descendant relationship.

1. Introduction

The eXtensible markup language (XML) as a de facto standard is emerged for data representation and the information exchange mechanism on the internet. Although XML document has rather complex internal structures and many nested tags, they can be modeled as rooted, ordered and labeled tree. XML database is a collection of tree data nodes. XML tree structure is important for querying process. The ability to process XML queries efficiently plays an important role in the deployment of the XML technology. Twig pattern queries matching is the process of retrieving information from an XML documents. In XML twig queries processing, Al-Khalifa et al [6] proposed a stack-based algorithm which decomposed the twig pattern into a set of binary (parent-child and ancestor-descendent) relationships between pairs of nodes. The query twig pattern can then be matched by (i) matching each of the binary structural relationships against the XML database and (ii) merging together these basic matches. Bruno et al [1] proposed a holistic twig join algorithm TwigStack to avoid producing large intermediate results by pushing only nodes that are sure to contribute to the final results onto stacks for ancestor-descendent queries. However, like most other algorithms, it is also has two-phase overhead. To address this problem, Chen et al [2] proposed Twig2Stack, bottom-up processing of generalized-tree-pattern queries over XML documents. Although it is a one phase holistic twig evaluation algorithm, it missing the idea of holistic approach of TwigStack. It pushes the nodes that do not contribute to final results onto stacks. Another novel one phase algorithm, HolisticTwigStack, proposed by Jiang et al [5], based on TwigStack algorithm to process the XML twig pattern efficiently. It avoids producing intermediate results and merging process by using linked stacks. It pushes elements onto stacks by satisfying two conditions (i) an element is pushed onto stack whose top element is an ancestor of the incoming element

and (ii) the incoming element must have the same closet pattern ancestor as the top element. For an element that does not satisfying the above conditions, it will be stored in a new stack. However, these algorithms considered twig queries with ancestor-descendent edges. In this paper, a new holistic twig algorithm, namely TwigMMatch is developed. With this algorithm, we can find all matches of twig pattern queries with both ancestor-descendent and parent-child relationships. Like HolisticTwigStack algorithm, outputs twig matches without a later merge process. Instead of outputting individual path matches as soon as they are formed, our method holds the path matches until entire twig matches are formed. The proposed algorithm yields no intermediate results and requires no additional merging phase.

The remainder of this paper is organized as follows. Section 2 reviews previous researches on XML query processing. Preliminaries are introduced in Section3. Section 4 describes algorithm, notation and data structure. We close this paper in Section 5.

2. Related Work

The core in XML query processing is the twig pattern matching over an XML database. Zhang et al. in [14] introduced the region encoding to process XML queries and proposed a multi-predicate merge join algorithm using inverted list. Lu et al [8] proposed a new labeling scheme called extended Dewey to efficiently process XML twig patterns, and the algorithm they developed is called TJFast. Unlike the previous algorithm based on region encoding, to answer a twig query, TJFast only needs to access the labels of the leaf query nodes. The result is enhanced functionality (can process wildcard), reduced disk access, and increased total query performance. Due to the merging operation they have very expensive cost in two phase twig pattern matching algorithms, Li et al[10] proposed two novel one-phase holistic twig matching algorithms,TwigMix and TwigFast. TwigMix introduce the getNext () function of TwigStack into Twiglist to avoid manipulation of useless elements in the stack and lists. TwigFast further improve this by introducing some pointers in the lists to completely avoid the use of stacks. Qin et al [11] proposed another one-phase algorithm called Twiglist for processing twig-pattern matching queries. The algorithm used simple list data structure for time and space that are linear with respect to the total number of pattern occurrences and the size of XML tree. Lu et al [7] proposed a new algorithm called, TwigStackList, for processing xml twig patterns with parent-child edges. A list data structured is used to cache limited elements to identify a large optimal query class.

3. Preliminaries

In this section, we describe xml data model, example of xml document and Twig pattern queries.

3.1 XML Data Model

An XML document can be modeled as rooted, ordered, and node-labeled tree where a node represent s an XML elements, attributes ,texts and edge represents a parent-child relationship between element-sub element, element-attribute and element-text pairs. Each node v is coded with a tuple of three values: (v.start, v.end:v.level). Such a coding scheme has several useful properties: (i) v1 is an ancestor of v2 if v1.start < v2.start <=v2.end < v1.end (ii) v1 is the parent of v2 if it is the parent of v2 and v2.level = v1.level+1.

```
< ? xml version="1.0"?>
<publication>
<book category="WEB">
    <title>Learning XML </title>
    <year>2005</year>
</book>
<book category="Data Base">
<title>Database Management System</title>
<price>39.95</price>
</book>
</publication>
```

Figure1. Example of XML document

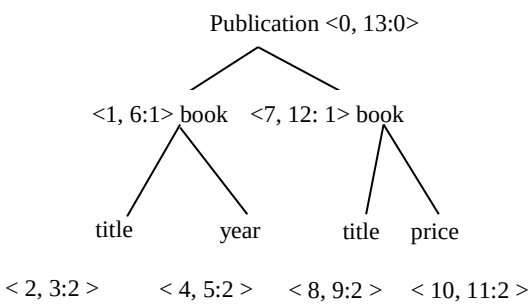

Figure2. XML document tree with region encoding

3.2. Twig Pattern Queries

XPath query is a language for addressing parts of an XML document. It supports the use of functions for interacting with the selected data from the document and also provides functions for the accessing information about document nodes as well as for the manipulation of strings, numbers and Booleans. For example, the XPath expressions are

Q1: book//title//price.

Q2: book/year//title.

This expression can be represented as a node-labeled twig pattern or a small tree. The twig pattern node may be elements, attributes and character data. The query twig pattern edges are either parent-child edges (a single line) or ancestor-descendent edges (a double line). The following figure illustrates twig patterns for above each query.

(a)Q1 **(b)Q2**

Figure 3. Twig Pattern Query Trees

4. Algorithm

Algorithm1: TwigMMatch (Q)

Input: a twig query (Q) with n nodes (v_1,……,v_n) and XML document tree (T) with v_i-typed query nodes.

Output: all matched answers for twig query.

begin

1. Let T_r be the streams of XML tree nodes for twig query root node and T_l be the sets of streams of twig query leaf nodes.

2. Let QP be the query pattern, query.root ∈ root (QP) ∧ query.leftleaf ∪ query.rightleaf ∈ leaf (QP).

3. Let qr_i and ql_i be the first element of each streams. $1 \le i \le n$.

4. Initialize stacks as empty.

5. While (¬ end (Q) ∧ $q_i \neq 0$) do
6. For (v is root (Q)) do

```
7.      for (qr_i ∈ v) do
8.          If (leaf (QP) ∈qr_i) then
9.              Push (S_r, qr_i)
10.         end
11.     Advance (T_r.qr_i)
12.     end
13.end
14.  for (is leaf (Q)) do
15.    for (ql_i ∈ leaf (QP)) do
16.     If (ql_i∈ADchildren ) ∨ ( ql_i∈ PCchildren) then
17.        Push (S_l, ql_i)
18.     end
19.     Advance (T_l.ql_i)
20.    end
21.  end
22. end
23.   While ( ¬ empty (S_r)) do
24.       If (noleafstack is empty) then
25.           ShowTwigSolution (S_r, S_l)
26.       end
27.          TopElem=S_r.pop()
28.        If (isempty (S_r)) then
29.            Clean all elements of leafstacks
30.            end
31.  end
```

4.1. Notation and Data structure

Given a twig query Q and an XML data tree D, a match of Qin D is identified by a mapping from the nodes in Q to the elements in D, such that: (i) the query node is satisfied by the corresponding database elements and (ii) the structural relationships (i.e. P-C and A-D relationships) between query nodes are satisfied by the corresponding database elements. Like most twig query processing approaches we adopt the region encoding scheme. Each node in the XML document tree is assigned a unique 3-ary tuples (Start, End, and Level No) which represents the start, end positions and level number of the node respectively. The region encodings support efficient evaluation of structural relationships of elements in document tree. Formally, element u is an ancestor of element v if and only if u.start <v.start <u.end. For parent-child relationship, u.level = v.level -1 is checked. Start positions and end positions of XML data tree are calculated by performing a pre-order of the traversal of the document tree. A query twig pattern can be represented with a tree. The self-explaining function is root (q) and is leaf (q) examines whether a query node q is a root or a leaf node. As in all the stack-based algorithms, there is a data stream T associated with each pattern node q of the twig query. The elements in each stream are sorted by their start position. There is a pointer that point to the current element in each streams T_r and T_l. The pointer can be forwarded to the next element in each stream with the procedure advance (T_r) and advance (T_l). Given a twig query, stack S_q for each node in twig query. The operations over stack are: empty, pop and push. In the rest of paper, node refers to a tree node in the query twig pattern and element refers to an element in document tree.

4.2. TwigMMatch

Algorithm TwigMMatch is presented in Algorithm 1. The algorithm computes the answer to a twig query pattern in one phase. Initially, stacks are to be empty. Consider the twig query pattern tree in figure 3(b) and xml document tree in figure 2. Given an XML tree T and a twig pattern matching query Q, represented as a query tree is to find all the occurrences of such twig query nodes book, year, title. Book is the root node and year and title are the leaf nodes. The relationship between book and title is the ancestor-descendant and book and year is the parent-child relationship. Suppose root node book is set as variable a and leaf nodes title and year are set as variable b and c. Elements in XML document are set as variables equal to the twig query tree illustrated in figure 4.

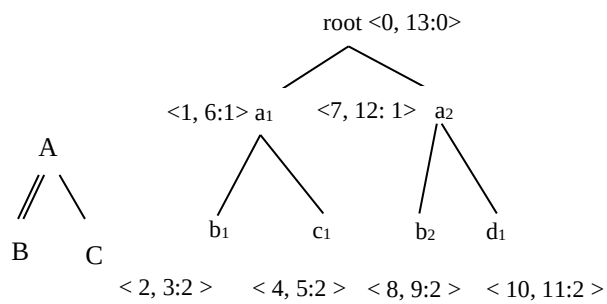

Figure4. Twig pattern tree and XML document tree

Accordingly query tree pattern, there are three data streams T_r = (a1, a2), PC children T_l= (c_1), AD children T_l = (b1, b2). Figure 5 shows the example stack structure of our algorithm for twig pattern tree and xml document tree in figure 4. According to the twig query tree in figure4, three stacks are needed. One stack for root node and two stacks for each leaf nodes. For root node, Line (6-8) is repeatedly checked for element that is pushed onto root stack S_r only if it is surely to contribute to a twig match, otherwise we simply advance to next element (line11). For leaf nodes, line (14-16) is repeatedly checked to push onto leaf stacks S_l. The link is created from leaf stacks to root stack if the top element in root stack is the parent of top elements in leaf stacks. ShowTwigSolution is called to output twig matches if root stack S_r and leaf stacks are not empty. The actual match answer for query tree in figure 4 is (a_1, b1, c1).

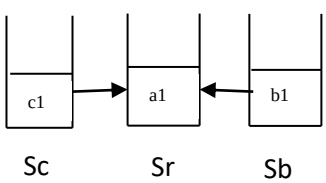

Figure5. Stack structure of TwigMMatch

5. Conclusion

In this paper, TwigMMatch twig pattern matching algorithm is proposed. Unlike the previous HolisticTwigStack algorithm, level information in region encoding is used for parent-child relationship of elements in document tree. In proposed algorithm, a chain of linked stack structure is applied that contains a compact encoding of overall of solutions to the query twig pattern without generating intermediate results and merging phases. Moreover, unlike existing one-phase algorithms, proposed algorithm perform twig query patterns with both ancestor-descendant and parent-child relationships.

References

[1] B. Nicolas, K. Nick and S. Divesh. Holistic twig joins: optimal XML pattern matching. In Proceedings of SIGMOD, pages 310-321, 2002.

[2] C. Songting, L. Hua-Gang, T. Junichi, H. Wang-Pin, A. Divyakant and C. K. Selcuk. Twig2Stack: bottom-up processing of generalized tree-pattern queries over XMl documents. In Proc. VLDB, 2006.

[3] C.Ting, L. Jiaheng and L. Tok Wang. On boosting holism in XML twig pattern matching using structural indexing techniques. In Proceedings of SIGMOD, 2005.

[4] J. Haifeng, W. Wei, L. Hongjun and Y. Jeffery Xu. Holistic twig joins on indexed XML documents. In Proceeding of VLDB, pages 273-284, 2003.

[5] J.Zhewei, L. Cheng, H. Wen-Chi and C. Qiang Zhu Dunren. Efficient processing of XML twig pattern: A novel one-phase holistic solution. In Proc. DEXA, 2007.

[6] K. Al. Shurug, H. V. Jagadish, K. Nick, P. Jignesh M, S. Divesh and W. Yuqing. Structural joins: A primitive for efficient XML query pattern matching. In Proceedings of ICDE, pages 141-152.2002.

[7] L. Jiaheng, C. Ting and L. Tok Wang. Efficient processing of XML twig patterns with parent child edges: A look-ahead approach. In Proceedings of CIKM, pages 533-542, 2004.

[8] L. Jiaheng, L. Tok Wang, C. Chee-Yong and C. Ting. From region encoding to extended dewey: On efficient processing of XML twig pattern matching. In Proceeding of VLDB, 2005.

[9] L. Jiaheng, L. Tok Wang, Y. Tian, L. Changqing and N. Wei. Efficient processing of ordered XML twig pattern. In Proceeding of DEXA, 2005.

[10] L. Jiang and W. Junhu. Fast matching of twig patterns. In Proc. DEXA, 2008.

[11] Q. Lu, Y.Jeffery Xu and D. Bolin. Twiglist: Make twig pattern matching fast. In Proc. DASFAA, 2007.

[12] R. Praveen and M. Bongki. PRIX: Indexing and querying XMl using Prufer sequences. In Proc. DASFAA, 2007.

[13] XPath. http://www.w3.org/TR/xpath.

[14] Z. Chun, N. Jeffery, W. David De, L. Qiong and L. Guy. On supporting containment queries in relational database management systems. In proceedings of SIGMOD, pages 425-436, 2001.