Holistically Processing XML Twig Queries with AND, OR, and NOT Predicates

Dunren Che
Department of Computer Science
Southern Illinois University Carbondale
Carbondale, IL 62901, USA
dche@cs.siu.edu

ABSTRACT
Structural joins are important for XML queries, but suffer from producing large, unused intermediate result sets. Holistic twig joins claim to solve this problem, but previously proposed algorithms fail to support XML queries involving all the three types of logical operations predicates: AND, OR, and NOT, which are however highly desired (such queries are referred to as All-twigs). Currently, there is no holistic twig join algorithm designed for All-twigs. In this paper, we first propose to normalize All-twigs to harness their complexity and then present a holistic join framework based on normalized All-twigs.

Categories and Subject Descriptors
D.2.8 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—Pattern matching

General Terms
Algorithms

Keywords
XML query, query processing, query evaluation, twig pattern matching, holistic twig join

1. INTRODUCTION
Since the advent of the World-Wide-Web, the volumes of Web published data (particularly in the form of XML) keeps mounting up. Effective and scalable techniques for querying very large XML data repositories, typically stored in a database, become extremely important. Essentially, an XML database is a tree database — consisting of collections of trees, called data trees. Accordingly, XML queries specify tree-shaped search patterns, called twig patterns [2], which may be accompanied by additional predicates imposed on the contents or attribute values of the data tree nodes. XML queries are thus called twig queries. Answering a twig query requests to find all instances in a database that match the twig pattern and satisfy the specified predicates (if any) in the query. A naive way of executing a query is to scan the database (typically for many times) in order to identify all the matches. A better alternative is to use structural joins (e.g., [1]) in a bulk way to compute the matches for each edge of a twig pattern, and then “stitch” the matches found for the individual edges to form the total matches for the entire twig query. This approach typically creates large sets of unused intermediate results, even if the final result set is pretty small. Yet, a superb alternative, called holistic twig join, is to compute all the matches in a holistic way so that irrelevant intermediate results (which are detriment to performance) will not be generated. The first holistic twig join algorithm, TwigStack, was proposed by Bruno et al [2] in 2002. Since then the idea of “holistic twig join” has been widely followed and generalized by numerous researchers such as [3, 4, 5, 6, 8, 7, 9]. Most of these algorithms deal with queries whose sibling edges are (implicitly) connected by the AND logic only. However, general XML queries typically contain arbitrarily specified logical operations, including AND, OR, and NOT (or referred to as AND-predicate, OR-Predicate, and NOT-predicate, respectively). For example, query "/dblp/paper[NOT reference]” finds papers that do not have references, though contains just a single NOT operation \(^1\). Twig queries that may involve all the three logical operations are called All-twigs(this is in contrast to the mere AND/OR-twigs in [5] that contains AND and OR only). From now on, twigs or twig queries refer to All-twig queries.

The lack of support for dealing with all the three logical operations within a query can make even the best holistic twig join algorithm completely useless if the input query involves all these logical operations. So far, we see only paper [5] discussing the issue of OR predicates, and paper [9] addressing the NOT predicates in a twig query. There is no integral method proposed that can deal with all the three logical operations. To the best of our knowledge, there is no reported work that aims at solving the matching problem of general twig queries that may involve all the three logical operations, AND, OR, and NOT. We are thus motivated to solve this important problem and report our findings in this paper. We propose a framework and algorithms to solve the

\(^1\)The NOT logic in XQuery is typically represented via the empty() function, however, in this paper, to make the NOT logic more explicit, we directly use the word ‘NOT’ in our query expressions.
A twig query is represented as a tree, which consists of various nodes and edges. We are interested in solving the problem of the most general twig queries, All-twigs, that may contain any or all kinds of logical operations: AND, OR, and NOT. So, we define a twig query as a tree that may consist of the following types of tree nodes:

- QNode: A location step node stands for one location step in the original twig query.
- ANode: An AND-logical operation/predicate node, always takes the text ‘AND’ in the query tree.
- ONode: A OR-logical operation/predicate node, always takes the text ‘OR’ in the query tree.
- NNode: The NOT-logical operation/predicate, always takes the text “NOT” in the query tree. NNodes are commonly combined with the subsequent node to form composite nodes in a twig query. We have the following types of composite nodes that accommodate NOT:
  - QNode: combination of NOT with a following QNode; the interpretation of this type of nodes is that the parent elements must not contain any of the sub-elements associated to the QNode.
  - ANode: combination of NOT with a following ANode.
  - ONode: combination of NOT with a following ONode.

With the above types of tree nodes, our All-twig representation scheme is apparently a superset of that in [5], which represents AND/OR-twigs only.

Generally, the result of a twig query is a set of output twig instances. In previous algorithms, an output twig instance contains elements from all QNodes in the query. When the twig queries are generalized to include NOT and OR logic, it is no longer the case that every QNode will contribute elements to the output instance because a QNode may simply serve as a filter. Here, we generalize the output model of [5] as follows: each output twig instance for an All-twig query comprises of elements from only the QNodes that are not inside any OR or NOT predicate. The QNodes that produce output (or contribute to the output instances) are called output nodes. In our subsequent discussion, the term “query node” refers to either a QNode or an NQNode or both.

2. Query Normalization

A query tree may contain redundant nodes, which may be syntactically redundant or semantically redundant (according to certain constraint rules). Ideally, the query trees are simplified and normalized, and our holistic twig join algorithm (to be presented) can then be beneficially applied. We define a normalized All-twig query as the following:

Definition 2.1 (Normalized All-twig query). A normalized All-twig query is a query tree that has only four types of nodes: QNodes, NQNodes, ONodes, and ANodes, and satisfy the following conditions: (1) all OR-predicates are in DNF (disjunctive normal form); (2) NQNodes (if any) must be leaves; (3) ANodes (if any) can only appear within an OR-predicate branch.

We developed a procedure for normalizing All-twigs. This procedure has three steps: (1) NOT-pushdown, (2) AND-pushdown, and (3) simplification (details omitted due to space limitation). We can prove that every All-twig has an equivalent normal form and that can be obtained by the above normalization procedure (which performs a rule-based transformation).

2.3 Auxiliary Operations

Given a query tree Q, we will use q (and its variants such as qi and q0) to denote a QNode (occasionally an NQNode as well) in Q or the subtree rooted at q when there is no ambiguity, and use n (and its variants such as ni and n0) to refer to a node of any type in Q. We define a series of operations on an All-twig and its tree nodes that are either necessary or helpful for our subsequent discussion.

- children(n) returns all child nodes of n; parent(n) returns the parent node of n; Qchildren(n) stands for the set of QNodes in subtree n that are reachable from n without traversing other QNodes; NQchildren(n) stands for the set of NQNodes in subtree n that are reachable from n without traversing other tree nodes; Pparent(n) returns the nearest ancestor QNode of n; Qsibling(q) returns all sibling QNodes of n (not including q itself); NQsibling(q) returns all NQNodes in sub-tree q (q is inclusive); isLeaf(n) tests whether node n is a leaf; isRoot(n) tests whether node n is the root; isQNode(n) tests whether node n is a QNode; isNQNode(n) tests whether node n is a NQNode;
isONode(n) tests whether node n is an ONode; isANode(n) tests whether node n is an ANode.

Furthermore, we assume each QNode or NQNode q in an All-twig is associated with a stream, named T_q. Each stream maintains a list of elements that satisfy the node test and any additional predicate (if any). The elements in the streams are sorted by their regional code (start, end, and level) (here level is the nesting level). Each stream T_q is associated with a cursor, named C_q, for accessing the elements in the stream. We define the following operations regarding a stream and its cursor: end(C_q) tests whether the cursor C_q has reached the end of the stream; C_q→advance advances the cursor along forward by one position; C_q→reset resets the cursor to the beginning of the stream.

Each QNode q in an All-twig Q is assigned a stack, named S_q. As specified in the “classic” paper [2], each element in a stack consists of a pair: (its region code, a pointer to a matching parent element in S_{parent(q) }). The common stack operations, pop(), push(), and top(), are assumed.

The stacks must have the following properties[2]: (i) the nodes in stack S_q (from bottom to top) are guaranteed to lie on a root-to-leaf path in the XML database, and (ii) the set of stacks contain a compact encoding of partial and total answers to the twig pattern query, which represents in linear space a potentially exponential (in the number of query nodes) number of answers to the twig pattern query.

3. SUPPORTING MACHANISMS

Our interest is in solving the pattern matching of All-twigs, and we only need to care about normalized twigs because every twig can be normalized. Normalized All-twigs many only contain 4 different types of nodes: QNode, NQNode, ONode, and ANode. We will define the mechanisms for satisfying the conditions induced by each type of nodes that may appear in a normalized All-twig.

A QNode is trivially satisfied if the element associated to it is a descendant or child depending on the specific type of the QNode. The edgeTest function introduced by Jiang et al [5] can still be used for this purpose. NQNode is unique to All-twigs and introduces a new dimension of challenge in twig matching. We will introduce a new function, called nEdgeTest, to help solving this problem. As for the evaluation of an ONode in an All-twig, the case is more complicated due to the introduction of NQNodes (compared with that in [5]).

We use the following convention for ease of presentation: each QNode q_i (or n_i) is associated with an element node e_i (by changing ‘q’ or ‘n’ to ‘e’) such that tag(e_i) = tag(q_i). The following definition for the edgeTest function is adapted from [5]:

Definition 3.1. Let q be a QNode in an All-twig and q0 be Qparent(q), and e and e0 be the associated elements of q and q0, respectively. Boolean function edgeTest(e0, e) or edgeTest(e0, q0) evaluates true if element e0 is an ancestor (respectively, the parent) of element e if q is an ancestor-descendant (respectively, a parent-child) QNode.

Definition 3.2. Let q be a NQNode in an All-twig and q0 be Qparent(q), and e0 and e be the associated element of q0. Boolean function nEdgeTest(e0, q) evaluates true if for all elements e_i (if any) that can be associated to the QNode corresponding q, edgeTest(e0, e_i) returns false.

Definition 3.3. Let ONode n be the root of an OR-predicate subtree, and q is QParent(n) associated with element e. Boolean function ONodeTest(e, n) evaluates true if e satisfies the OR-predicate associated to the ONode n.

Definition 3.4. Let Q be a query tree with N nodes n_1, n_2, n_k, where n_1 is the root QNode. By convention, e_i is the associated element of n_i if n_i is a QNode. We say element e_i has a match for an All-twig n_i if the following holds for each child subtree n_k_i of n_i: (1) if n_k_i is an ONode, then ONodeTest(e_i, n_k_i) evaluates true; (2) if n_k_i is an NQNode, then nEdgeTest(e_i, n_k_i) evaluates true; (3) otherwise (i.e., n_k_i is a QNode) edgeTest(e_i, n_k_i) evaluates true and element e_k_i has a match for the subtree rooted at n_k_i in case n_k_i is not a leaf node.

Definition 3.4 implies that, in order to identify a match instance for an All-twig, we need to call upon three functions: ONodeTest, nEdgeTest, and edgeTest. Their implementation becomes a key issue that is addressed next.

Solving edgeTest and nEdgeTest (per their definitions) is relatively easy, and solving ONodeTest is a little more tricky. In paper [5], Jiang et al introduced the concept of OR-block to help solving simple OR-predicates (without embedded NOT logic). We found this concept is still useful, but needs essential extension to cover more general OR-predicates (with embedded NOT logic). In the following, we first extend the OR-block concept, and then develop correspondingly a more sophisticated evaluation strategy for general OR-predicates.

Definition 3.5 (OR-block). Given a twig query Q, an OR-block is a tree t embedded in Q such that the root of t is an ONode n, parent(n) is a QNode, and the leaf nodes of t are Qchildren(n) or NQchildren(n). In addition, a logical formula, denoted as P(n), is recorded in the root structure of the OR-block.

In an OR-block, all ANodes are “fused” into the recorded logical formula P(n). So there are no explicit ANodes any more. Notice that, as we work with normalized All-twigs, our OR-block is different from that in [5]: (1) our OR-blocks are single blocks — no embedded OR-blocks; (2) our OR-blocks may contain both QNodes and NQNodes, and NQNodes must be leaves if there are any. After all OR-predicate branches being replaced by corresponding OR-blocks, a normalized All-twig is represented by using only QNodes, NQNodes, and OR-blocks.

For an All-twig query, the evaluation has to enforce the semantics of the NOT logic implied by the NQNodes. We introduce the following definition for evaluating the OR-predicates that may involve the NOT logic:

Definition 3.6 (OR-predicate evaluation). Let ONode n be the root of an OR-predicate connected to QNode q, whose associated element is e. We say element e satisfies OR-predicate n or ONodeTest(e, n) is true if P(n) is true by replacing each QNode or NQNode n_i in P(n) with a Boolean function as follows: if n_i is a leaf QNode or a leaf NQNode, replace n_i with edgeTest(e, n_i) or nEdgeTest(e, n_i) accordingly; otherwise (i.e., n_i is a non-leaf QNode), replace n_i with the Boolean value (edgeTest(e, n_i) AND e_i has a match for subtree n_i).
The above definition embodies our strategy for implementing the ONodeTest function, which is critical to Alltwig evaluation and in turn calls the other two supportive functions, edgeTest and nEdgeTest. The implementation of these three functions are given in Figure 1 and 2, respectively. Together, they form the basic supporting mechanisms in our holistic twig join algorithm.

Algorithm nEdgeTest(e,n)
1: while not end(C_n) do
2: if edgeTest(e,C_n) == TRUE then
3: return FALSE
4: C_n = advance()
5: end while
6: C_n = reset()
7: return TRUE

FUNCTION edgeTest(e,q)
/* assume ancestor-descendant edge only */
1: if e.start < C_q = start and e.end > C_q = end then
2: return TRUE
4: else
5: return FALSE

Figure 1: The nEdgeTest Algorithm

Algorithm ONodeTest(e,n)
1: for each n_i in P(n) do
2: if isLeaf(n_i) and isQNode(n_i)
3: replace n_i by edgeTest(e,n_i)
4: else if isLeaf(n_i) and isQNode(n_i)
5: replace n_i by nEdgeTest(e,n_i)
6: else /* n_i is a non-leaf QNode */
7: replace n_i by (edgeTest(e,n_i) and hasExtension(n_i))
8: end for
9: evaluate P(n) and return the result

Figure 2: The ONodeTest Algorithm

5. Summary

Holistic twig joins are critical operations for XML tree queries. All three types of logical operations, AND, OR, and NOT, are equally important to general XML queries. However, existing holistic twig join algorithms fail to integrate the mechanisms needed for all these logical operations into a single algorithmic framework, resulting in unsolvable XML queries when all three types of logical operations are involved. In this paper, we introduced the concept of normalized All-twig queries and the procedure for obtaining Alltwig normalization. We summarized the first approach and algorithms for solving the All-twig pattern matching problems holistically, based on the normalized form of All-twigs. We are currently developing alternative solutions for Alltwig pattern matching, and doing systematic experiment study.

Due to space limitation, this paper is highly compacted. Interested readers are welcome to contact us for a full version, while we are in the process of preparing a more formal, full-version publication of this work (with experimental results included).

6. References