Answering Order-Based Queries Over XML Data*

Zografoula Vagena
UC Riverside
foula@cs.ucr.edu

Nick Koudas
University of Toronto
koudas@cs.toronto.edu

Divesh Srivastava
AT&T Labs-Research
divesh@research.att.com

Vassilis J. Tsotras
UC Riverside
tsotras@cs.ucr.edu

ABSTRACT

Order-based queries over XML data include XPath navigation axes such as `following-sibling` and `following`. In this paper, we present holistic algorithms that evaluate such order-based queries. An experimental comparison with previous approaches shows the performance benefits of our algorithms.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*query processing*

General Terms: Algorithms, Experimentation

Keywords: Holistic algorithms, order-based queries, XML

1. INTRODUCTION

XML is poised to become the basis for web-based and database-centric applications. Thanks to standard specifications for web services (such as SOAP, WSDL, etc.), applications can receive requests for data and return their answers tagged in XML. A key issue in XML data management is effective support of the *ordered, tree-structured* data model that the language employs.

Consider an XML database of articles. For each article entry, the tree structure and the order in which its sections are listed is relevant. Information extraction tools can extract the sections of an article following the related work section using the query:

```
/article/section[title = 'Related Work']/
   following-sibling::section
```

Queries with navigation axes such as `following-sibling` and `following` are referred to as *order-based queries*. This paper complements previous approaches [4, 5, 6] and investigates whether holistic techniques can be devised to answer order-based queries in XPath and XQuery. Our contributions are:

1. We devise algorithms for the `following-sibling` and `following` axes as well as their backward counterparts.
2. We propose holistic approaches for "twig" queries with order-based forward and backward axes of the same type.
3. We experimentally show that our solutions offer better query performance than previous approaches.

We proceed with Section 2 that describes our techniques, while in Section 3, their efficiency is experimentally investigated. Section 4 concludes the paper.

*The research of Zografoula Vagena and Vassilis J. Tsotras was partially supported by NSF grant IIS-0339032, UC MICRO and Lotus Interworks.

WWW 2005, May 10–14, 2005, Chiba, Japan.
ACM 1-59593-051-5/05/0005.

2. ALGORITHMIC APPROACHES

We describe set-based processing techniques for the order-based axes. For queries that contain both forward and backward axes, we first convert the backward axes into their forward counterparts, based on the ideas presented in [2]. Efficient algorithms are then developed to answer the modified queries.

2.1 The Tree Encoding

To enable set-at-a-time structural matching, we map the document into sequences of nodes. Each sequence maintains nodes with the same tag and each node is augmented with information that identifies its position within the XML tree, as in [1]. The position of an XML node is represented as: `(LeftPos, RightPos, PRightPos)` where: (a) `LeftPos` and `RightPos` are generated by counting tags from the beginning of the document until the start and the end tags of the element are visited, respectively, and, (c) `PRightPos` is the `RightPos` of its parent node. Several structural relationships between elements can thus be identified. An element y follows an element x if the `RightPos` of x is smaller than the `LeftPos` of y. Similarly, an element y is `following-sibling` of an element x if the `RightPos` of x is smaller than the `LeftPos` of y and the `PRightPos` of x is equal to the `PRightPos` of y.

2.2 Single Forward Axis Step

Consider the query a`/following-sibling::`b. Our processing algorithm's input consists of two streams, one with a nodes and one with b nodes. These streams are sequentially accessed, effectively computing a (merge) join. An important observation is that the following-siblings for some document node a_x occur after its descendants have been encountered. In other words, node a_x has to be buffered until the following-siblings of its descendants have been processed first. This is achieved by maintaining a stack S_a that keeps a nodes which have been accessed and are still needed to identify future following-sibling nodes. Note that an a node may be followed by many other nodes with the same label. Such nodes conceptually form a linked-list (`CSL` list). New nodes are appended to the end of such a list. If a node b_y becomes a following-sibling to the a_x node at the end of a CSL list, it is automatically a following-sibling to all other nodes in that list.

If the query contains a `following` axis, we need to identify for each document node a_x with label a, all b nodes that occur after a_x in document order and are not its descendants. The processing algorithm resembles the one for the following-sibling axis. The main difference is that after having identified the first match for a node a_x, all other b nodes in the stream have to be joined with it.

Since the processing of the two forward axes are similar, we focus on the `following-sibling` axis from now on.

2.3 Non-Branching Forward Path

Such queries contain a number of steps where all the intermediate nodes correspond to context nodes, while the leaf corresponds to a test node. The algorithm maintains one stack for each query context node. Each stack buffers nodes with following-siblings yet to be visited. In addition, a number of CSL lists are associated at each time with a stack. The role of a CSL list is similar to that in Section 2.2, i.e., to hold context-siblings. What is different, however, is that now an element b_y that is added in a CSL list, "remembers" the latest element a_x (in document order) in the previous query step for which b_y is a following-sibling. This is achieved by maintaining a *step*-pointer from b_y to the current last element in the corresponding CSL list of a_x. Step-pointers combine information between query steps and, along with the CSL lists, encode all partial results of the query. When a test node is accessed and the stack of the previous step is not empty, the following-sibling paths containing that node are decoded and returned.

2.4 Branching Forward Path

The most general query has the form of a subtree (or twig). Such queries can be processed holistically through a variation of the `TwigStack` [3] algorithm, which dealt with twig queries containing `descendant` and `child` axes. One thing to note here is that although TwigStack can guarantee node participation in the final result for queries with the `descendant` axis by inspecting a bounded number of lookahead symbols, such guarantees are not possible in the case of the `following-sibling` axis. The reason is that, as with the `child` axis, the necessary number of lookahead symbols is in the order of the size of the document.

2.5 Forward and Backward Axes

By adapting the method proposed in [2], a twig query with both `following-sibling` and `preceding-sibling` axes can be converted to a DAG query with only `following-sibling` axes. The key issue that needs to be addressed for DAG queries is that a query node a may have multiple parents in the query, i.e., may need to satisfy the following-sibling constraint for multiple context query nodes. We call such a query node a a join node from now on. For a document node a_x to participate in one total result, it is necessary that: (a) a_x has a following-sibling b_y in each of the document node sequences that correspond to children of a in the query, (b) a_x is following-sibling for at least one c_z in each of the document node sequences that correspond to parent nodes of a in the query, (c) each of the following-sibling nodes b_y recursively satisfies this property, (d) each of the nodes c_z recursively satisfies this property, and (e) the join node conditions are satisfied. With that in mind, we modify the approaches, described in the previous sections, so as to check that a produced DAG instance satisfies all these constraints. The special shape of the query DAG (i.e., if a node has two or more parents, their unique common ancestor is the `ROOT` node) enables the efficient checking of the above constraints.

3. EXPERIMENTAL EVALUATION

In this section, we present experimental results comparing the performance of the proposed algorithms (we refer to them as GNF) with the `Staircase` [4] and `Arb` [5] approaches. The dataset we used was the 1G (text) database generated by the XMark benchmark. We used the queries shown in Table 1. `Staircase Join` was excluded from query Q3 as it does not directly support twig queries. For the `Arb` algorithm, we report the time after the Arb database has been created.

The results are presented in Figure 1. In each case, the `GNF` technique performed better. For the single step query Q1, `Staircase`

Q1 :	incategory/following-sibling::mailbox
Q2 :	location/following-sibling::incategory/ following-sibling::mailbox
Q3 :	location[./following-sibling::incategory]/ following-sibling::mailbox

Table 1: Queries for XMark data

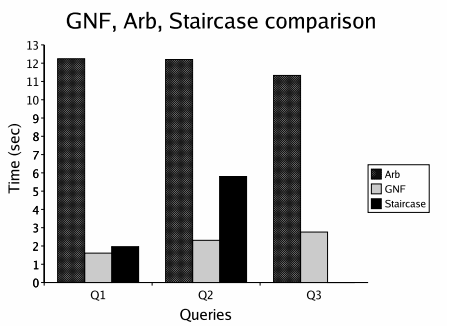

Figure 1: Staircase Join, Arb and GNF.

and `GNF` perform very similarly because both algorithms take into consideration only elements that are needed to create the results. When the number of query steps increases, the `Staircase` becomes worse because it incurs the overhead of the intermediate result materialization. In every case, `Arb` performed considerably worse; this is because it needs to access the whole document twice for each query, while the other two approaches take into consideration only relevant parts of the documents.

4. CONCLUSIONS

We studied the problem of supporting the ordered, tree shaped model of XML data. We proposed efficient algorithms to answer queries with the order-based navigation axes (both forward and backward ones), and validated them experimentally. To the best of our knowledge, this is the first approach that addresses those navigational axes in a complete, scalable, XML model-aware fashion.

5. REFERENCES

[1] S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava, and Y. Wu. Structural joins: A primitive for efficient XML query pattern matching. *ICDE*, 2002.

[2] C. Barton, P. Charles, M. Fontoura, and V. Josifovski. Streaming XPath processing with forward and backward axes. *ICDE*, 2003.

[3] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: Optimal XML pattern matching. *SIGMOD*, 2002.

[4] T. Grust, M. van Keulen, and J. Teubnem. Staircase join: Teach a relational DBMS to watch its (axis) steps. *VLDB*, 2003.

[5] C. Koch. Efficient processing of expressive node-selecting queries on XML data in secondary storage: A tree automata-based approach. *VLDB*, 2003.

[6] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. *SIGMOD*, 2002.