Expressing and Processing Path-centric XML Queries

Huayu Wu[1], Dongxu Shao[1], Ruiming Tang[2], Tok Wang Ling[3], and Stéphane Bressan[3]

[1] Institute for Infocomm Research, A*STAR, Singapore
{huwu, shaod}@i2r.a-star.edu.sg
[2] Huawei Noah's Ark Lab
tangruiming@huawei.com
[3] School of Computing, National University of Singapore
{lingtw, steph}@comp.nus.edu.sg

Abstract. A family of practical queries, which aim to return or manipulate paths as first class objects, cannot be expressed by XPath or XQuery FLWOR expressions. In this paper, we propose a seamless extension to XQuery FLWOR to elegantly express path-centric queries. We further investigate the expression and processing of intra-path aggregation, an analytical operation in path-centric queries.

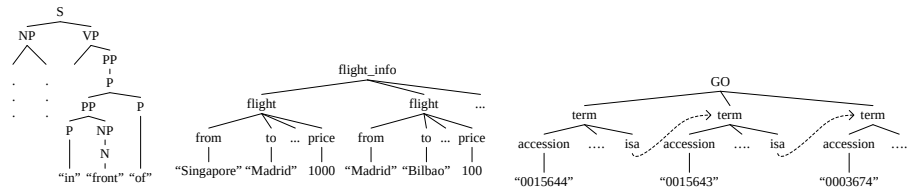
1 Introduction

A family of practical queries cannot be conveniently expressed in XPath or XQuery. These queries intend to return or manipulate paths as first class objects. We call them *path-centric* queries. Consider an XML document containing TreeBank [7], a parsed corpus of English texts annotated with their syntactic structure using XML tags, in Fig. 1(a). One user, a linguist, may legitimately be interested in finding the nesting of prepositional phrases ("PP") enclosing the word "front" in verb phrases ("VP") to study grammatical patterns. He/She could naturally think of expressing this query in XPath as:

```
//VP[.//PP//*/text()="front"]
```

However such a query in XPath, or corresponding queries in XQuery, would return a collection of "VP" elements, that is entire sub-trees rooted at a "VP" node. Yet the user is only interested in the paths between "VP" and "front". Without being able to project an interesting path, many path-centric query purposes cannot be easily met. For example, the user could not straightforwardly express a query counting, for every word "front" contained in a verb phrase, the number of prepositional phrases nesting it in the same verb phrase using the FLWOR constructs in XQuery.

Further, consider the XML document outlined in Fig. 1(b), which contains information about flights. One user may be interested in finding the routes from Singapore to Bilbao. There is no simple XQuery FLWOR query that can produce this result, except writing an XQuery program (user-defined function) that



(a) TreeBank (b) Air flight (c) GeneOntology

Fig. 1. Three documents to illustrate structural path and semantic path

explicitly iterates over the different "flight" elements and tests the connections. Similar queries are natural in the presence of XML ID references. In the GeneOntology data [3], as illustrated in Fig. 1(c), where genes reference their super-class genes. A query to find all other family genes between two known genes cannot be expressed by the current XQuery FLWOR constructs.

Actually XQuery is Turing complete only with the companion of user-defined functions to make it an XQuery program. By doing this, we actually expect an XML database user to be equipped with programming skill on recursive functions, for such a simple query purpose. It will be appreciative if the basic form of XQuery, i.e., FLWOR, can be extended so that such path-centric queries can be easily expressed in a more declarative way.

In this paper we propose seamless extensions to XQuery that can elegantly express queries that return paths and perform intra-path aggregation.

2 SPath: extending XQuery for path-centric queries

We name our extension `SPath`, with two functions, `StrucPath` and `SemPath`. The function output is a set of chains of document nodes that are structurally or semantically connected. Further, each node may have additional descendent nodes to describe it. We simply call the chains of nodes outputted by the two functions *paths*. More detailed description about *path* can be found in [1].

Definition 1. *(*Structural Path, Semantic Path*) In a returned path, if the condition to link adjacent nodes is the structural constraint of parent-child relationship in the document tree, the path is called a structural path; if the condition is semantics-based join, the path is called a semantic path.*

2.1 SPath expressions

Definition 2. *(*Definition of SPath functions*)*
StrucPath function ::= StrucPath$[\![tp \times tp \rightarrow sp\_set]\!]$
SemPath function ::= SemPath$[\![tp \times tp \times tp \times cond \rightarrow sp\_set]\!]$
tp ::= XPathExpr ('/' | '//') tpRel
*tpRel ::= nodeName | tpRel ('[' ('//')? tpRel | XPathExpr op value ']')**

cond ::= 'parent' XPathExpr op 'child' XPathExpr
op ::= '=' | '≠' | '>' | '<' | '≥' | '≤'
sp_set ::= {path}*

XPathExpr *is the simplified XPath expression with only axis of '/' and '//',* nodeName *is the XML tag label,* **parent** *and* **child** *are two reserved keywords to represent the parent node and child node in a pair of adjacent nodes, and* value *denotes constant string or numeric values.*

Definition 3. *(*Semantics of SPath functions*) Let* sp *be a structural path or a semantic path, in terms of a list of nodes, where* sp[first] *(*sp[last]*) denotes the first (last) node in* sp. sp_set *is the set of paths to be returned. For a node* n, n.next *denotes the next adjacent node of* n *in the path. Let the predicate* sat(X, Y) *return a Boolean value meaning whether* X *satisfies the constraint specified in* Y *and the rel(X, Y) stands for the linking relationship between node* X *and* Y. diff(X, Y) *represents the set difference between* X *and* Y. PC *denotes the constraint of parent-child relationship in an XML tree:*
StrucPath(start, end) ::= {sp | sat(sp[first], start) ∧ sat(sp[last], end) ∧
(∀ n ∈ diff(sp, {sp[last]}), sat(rel(n, n.next), PC)}
SemPath(nodePattern, start, end, linkCond) ::=
{sp | sat(sp[first], start) ∧ sat(sp[last], end) ∧
(∀ n ∈ diff(sp, {sp[last]}), sat(n, nodePattern) ∧
sat(rel(n, n.next), linkCond))}

As mentioned, each path node in the output paths may have additional descendant nodes, i.e., it is similar to element. Thus the existing XPath functions can work on the `SPath` output in the same way as they do on an element.

2.2 XQuery extension

In an XQuery query, a `SPath` function can be appended to an XPath expression, to find all paths under the element selected by the XPath expression.

Definition 4. *(*Extended Path*) Let* PathExpr *be the original XPath expression (or some built-in function returning element) used in XQuery, whose syntax and semantics is defined in XQuery specification [8] and will not be repeated here. Let* ExtPathExpr *be the extended path expression, and* tp *and* cond *are defined in Definition 2:*
ExtPathExpr ::= PathExpr | PathExpr '.StrucPath(' tp ',' tp ')' |
PathExpr '.SemPath(' tp ',' tp ',' tp ',' cond ')'

The following example illustrates how desired structural path and semantic path are found by the two `SPath` functions under an element.

Example 1. **Structural path.** In the TreeBank document in Fig. 1(a), all structural paths starting at "VP" and ending at some POS tag with child value of "front" are specified by

```
doc("TreeBank.xml").StrucPath(//VP, //*[text()="front"])
```

Semantic path. In the Flight document in Fig. 1(b), all routes from Singapore to Bilbao (the price of each flight must be outputted for analytical purpose) can be found by

```
doc("Flight.xml").SemPath(//flight[from][to][price], //flight
  [from ="Singapore"], //flight[to="Bilbao"], parent/to=child/from)
```

Because each path is an ordered list of nodes with additional descendants, in the extended XQuery FOR and LET can be used to iterate over a path and bind variables to nodes.

Example 2. The XQuery expression and result for the query to find the structural paths starting at "VP" and ending at some tag with child value of "front" in the document in Fig. 1(a) are:

```
FOR $p IN doc("TreeBank.xml").StrucPath(//VP, //*[text()="front"])
RETURN
    <structural_path>{FOR $e IN $p RETURN $e}</structural_path>
```

3 Intra-path aggregation: a practical case

In this section, we discuss a practical query purpose, intra-path aggregation, as an example to illustrate the importance of the proposed `SPath` functions in expressing path-centric queries with the basic form (FLWOR) of XQuery.

3.1 Expression

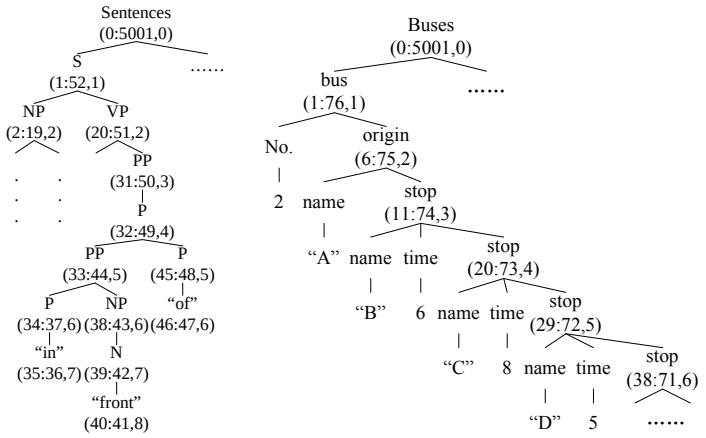
Many query purposes aim to aggregate nodes along a single path, which is referred as *intra-path* aggregation. The existing XQuery language is inconvenient in expressing a single path, and so is it for intra-path aggregation.

Example 3. (**Tag Aggregation**) Consider the TreeBank data in Fig. 1(a), and the query to find the total number of *PP* in each path that starts at *VP* and ends at some tag with value of "front". The XQuery expression with our extended functions is shown as:

```
FOR $p IN doc("TreeBank.xml").StrucPath(//VP, //*[text()="front"])
RETURN {count(FOR $e IN $p WHERE $e.name()="PP" RETURN $e)}
```

In the outer FOR clause, `StrucPath` returns a set of paths and $p iterates over these paths. In other words, in each iteration, the variable is bound to a *path*, rather than the element returned by the doc() function. In the nested FOR clause, $e iterate over the nodes of a certain path, and is bound to each *node*. Only with these new output units, intra-path aggregation for path nodes is expressible under FLWOR constructs.

The new GROUP BY construct introduced in XQuery 3.0 can be used in intra-path aggregation. If the query asks for the total number of different syntactic tags in each path starting at VP and ending at "front", it will be:



(a) TreeBank (b) BusInfo

Fig. 2. XML data with containment labeling (partial)

```
FOR $p IN doc("TreeBank.xml").StrucPath(//VP, //*[text()="front"])
RETURN { FOR $e IN $p
         GROUP BY $e.name()
         RETURN <{$e.name()}>{count($e)}</{$e.name()}> }
```

Example 4. (**Value Aggregation**) Now we consider the intra-path aggregation for values in semantic paths. Consider the query to find the quotation of all direct and transit flight from Singapore to Bilbao in the document in Fig. 1(b). The XQuery expression is:

```
FOR $p IN doc("Flight.xml").SemPath(
  //flight[from][to][price], //flight[from="Singapore"],
            //flight[to="Bilbao"],parent/to =child/from)
RETURN {sum(FOR $e IN $p RETURN $e/price)}
```

3.2 Execution

In this section, we study how to extend the existing query processing algorithms to support intra-path aggregation. Structural path and semantic path have different characteristics, so the query execution for the two types of paths should adopt different approaches. Basically, the query processing over semantic paths can reduce to graph search problem, in which problems such as recursion handling can be solved accordingly. In this paper, we put the focus on structural path. More discussion about the query processing over the two types of paths can be found in our technical report [1].

We use two examples to illustrate our algorithm for tag-based and value-based intra-path aggregation.

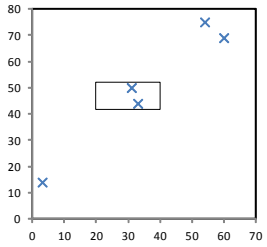


Fig. 3. Example R-tree encoding with range query rectangle

Example 5. Consider the query in Example 3. The TreeBank data with labels is shown in Fig. 2(a). We first find all structural paths by issuing a structural join query `//VP//"front"`. Then many pairs of labels corresponding to *VP* and *front* are returned, each of which represents a structural path. (20:51,2) and (40:41,8) is one such pair. Then we perform an outer structural join between all tuples of labels representing paths and the inverted list of *PP*. For the path starting at (20:51,2) and ending at (40:41,8), there are two PP labels (31:50,3), (33:44,5) are inside the path. Then for this path, the aggregation result is 2.

Example 6. Consider the BusInfo XML document shown in Fig. 2(b). Consider a query to find the total traveling time from stop B to stop D:

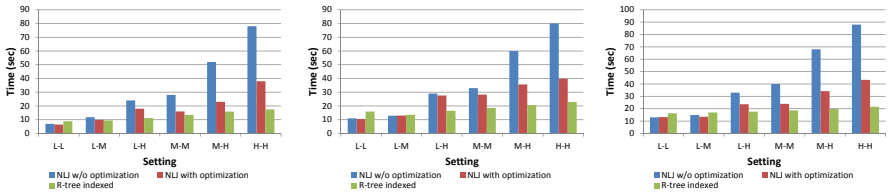
```
FOR $p IN doc("BusInfo.xml").StrucPath(
          //stop[name="B"], //stop[name="D"])
RETURN {sum(FOR $e IN $p WHERE $e/name/text() != "B"
            RETURN $e/time/text())  }
```

In this query, the node, the aggregate attribute and the predicates form a twig, `//stop[name!="B"]/time`. By matching this twig, we get the labels of each satisfied *stop* and the corresponding time values, i.e., {(20:73,4), 8} and {(29:72,5), 5}. The desired structural path can be found by another pattern matching for `//stop[name="B"]//stop[name="D"]`. The path shown in Fig. 2(b) is one of the answers, i.e., the path from (11:74,3) to (29:72,5). Last, we perform an outer structural join between the path and the tuples of answers found by first twig matching. Since both (20:73,4) and (29:72,5) are within the path from (11:74,3) to (29:72,5), the time value 8 and 5 are summed up as the result.

The pseudo code and more details on the algorithm can be found in [1].

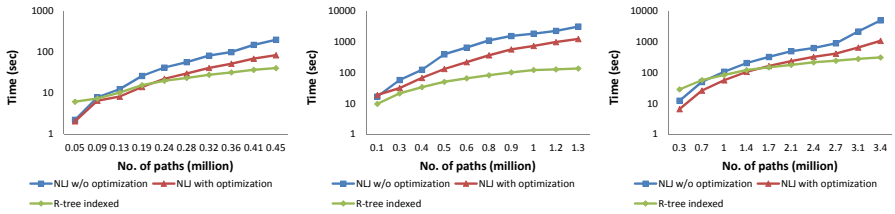
We further proposed an R-tree based optimization to address all nodes contained in a given path.

Example 7. Consider the query in Example 5. The R-tree for the inverted list of PP is shown in Fig. 3, in which each label in the inverted list will be matched by a point in the two-dimensional R-tree space. Each path instance is encoded as a range-search rectangle in R-tree. For example, the path starting at (20:51,2) and ending at (40:41,8) corresponds to the rectangle in Fig. 3. The PP nodes within the path will be the points enclosed by the rectangle.



(a) Average depth of 5 (b) Average depth of 10 (c) Average depth of 15

Fig. 4. Efficiency test on synthetic data



(a) count(PP)-//NP//VP (b) count(NP)-//VP//PP (c) count(PP)-//VP//NP

Fig. 5. Efficiency test on TreeBank data

4 Experiments

We assess the usability of our `SPath` extension compared to the XQuery user-defined function to express path-centric queries and conclude that our extension is more user friendly. The details can be found in [1]. In this section, we report the query performance evaluations for intra- structural path aggregation.

We implement three algorithms for intra- structural path aggregation. The first one is the baseline algorithm we proposed. In the second algorithm, we optimize the outer structural join. For each path, we only start structurally joining it with an inverted list when a label in the inverted list falls behind the starting node of the path, and skip the rest of the labels in the inverted list once we find one label falls behind the end node of the path. We name it *NLJ with optimization*. Finally, we implement the R-tree index to avoid structural joins.

We first randomly generate a 50MB XML data and queries to test. The detailed settings can be found in [1]. The experimental result is shown in Fig. 4. We can see that the result is similar for all the three tested data with different depth, i.e., the optimization can improve the performance, and R-tree index has obvious advantage when the frequency of path nodes increases.

In the second part, we use a real-life data set, TreeBank for evaluation. We choose simple path predicates that generate a large number of paths. The experimental results are shown in Fig. 5, in which x-axis stands for the number of path instances obtained by executing the corresponding path query for different sizes of documents, and y-axis is the running time in log-scale. We can see that for all queries, as the document size increases, the running time of the naive nested

loop join increases very fast. After optimizing the algorithm, the performance is better. The best one is the R-tree indexed algorithm, in which the increasing rate is rather low. However, for the first few smaller documents, the R-tree indexed algorithm is not always better than others due to the overhead.

5 Related Work

There are a number of extensions to XPath and XQuery in recent years. [9] extends XPath by introducing a *Related Axis*, to specify the related relationship between query nodes. [2] introduces functions to express fuzzy search in XPath expression, which relaxes the strict requirement on the knowledge of XML structure for query issuers. SXPath [5] allows spatial navigation for Web documents.

Although some existing extensions enhance the expressivity of XPath or XQuery, they are still built on element selection and cannot project interesting path from the element subtree. In [6], a new XML query language XSQuirrel is proposed, which projects a sub-document from an XML document. However, it requires the user to input the detailed pattern of the path, which is similar to the "hard coding" in the XQuery RETURN clause. [4] presents a projection technique to filter unnecessary elements for in-memory XQuery processors, but it is not for expressing path-centric queries discussed in this paper.

6 Conclusion

We argue in this paper that the support for path-centric queries in XPath and XQuery is not sufficient. We propose a simple and seamless extension to XQuery that can express such queries: the `SPath` extension with two functions for path expressions. We show how XQuery with this extension can be used to effectively and elegantly express path-centric queries of interest. We use intra-path aggregation as a practical example to illustrate the importance of our extension, and also discuss query processing issues when the number of path instances is large.

References

1. http://www1.i2r.a-star.edu.sg/∼huwu/report.pdf.
2. A. Campi, E. Damiani, S. Guinea, S. Marrara, G. Pasi, and P. Spoletini. A fuzzy extension of the XPath query language. *J. Intell. Inf. Syst.*, 33(3):285–305, 2009.
3. GeneOntology. http://www.geneontology.org/.
4. A. Marian and J. Siméon. Projecting XML documents. In *VLDB*, 2003.
5. E. Oro, M. Ruffolo, and S. Staab. Sxpath - extending xpath towards spatial querying on web documents. *PVLDB*, 4(2):129–140, 2010.
6. A. Sahuguet and B. Alexe. Sub-document queries over XML with XSQirrel. In *WWW*, pages 268–277, 2005.
7. The Penn Treebank Project. http://www.cis.upenn.edu/∼treebank/.
8. W3C Consortium. XQuery 1.0: An XML query language, http://www.w3.org/TR/xquery/. 2007.
9. J. Zhou, T. W. Ling, Z. Bao, and X. Meng. Related axis: The extension to XPath towards effective XML search. *J. Comput. Sci. Technol.*, 27(1):195–212, 2012.