Order-Aware Twigs: Adding Order Semantics to Twigs

M. Abdul Nizar[1], P. Sreenivasa Kumar[2]

[1] College of Engineering Trivandrum, India- 695016
nizar@cet.ac.in
[2] Indian Institute of Technology Madras, Chennai, India-600036
psk@iitm.ac.in

Abstract. Although many algorithms have been proposed for evaluating XPath queries containing *child* and *descendant* axes and predicates ($XPATH^{/,//,[\,]}$) against XML data, there are very few efforts towards developing algorithms for processing path expressions with other axes namely, *following*, *preceding*, *following-sibling* and *preceding-sibling*. In this article, we demonstrate that the conventional twig-structure, which is used to represent expressions in the subset $XPATH^{/,//,[\,]}$, has no order information built into it and hence fails to model XPath expressions with ordered axes. This motivates us to introduce a new type of twig-structure, which we call *Order-aware Twig*, to effectively represent XPath expressions with ordered axes.

Categories and Subject Descriptors: H.3 [**Information Storage and Retrieval**]: Query Formulation; H.2.4 [**Systems**]: Query Processing

Keywords: Information Storage and Retrieval, Query Formulation, XML, XPath, Ordered Axes, Twig Query, Order-aware Twig

1. INTRODUCTION

Due to the wide-spread use of XML data, especially in the web context, many query processing systems have been developed to operate on indexed XML document collections and XML streams. These systems use XPath[Berglund et al. 2007] as the query specification component either independently or as part of languages like XQuery[Boag et al. 2007]. Complex queries can be expressed in XPath using path expressions consisting of axis specifiers and predicates. For instance, the path expression */descendant::section[child::name = "Motivation"]/child::figure* returns all *figure* elements present in the section titled *Motivation* in an XML document representing a journal article or book. Such path expressions consisting of *child* and *descendant* axes and predicates ('[...]') are conventionally represented using tree structures known as twig queries or Tree Pattern Queries (TPQs). Twig representation of XPath expressions is quite convenient for query processing as the process is closely tied to the representation scheme adopted for the XML data. Naturally enough, the majority of XML query processing algorithms ([Bruno et al. 2002; Lu et al. 2005; Chen et al. 2005; Moro et al. 2005; Chen et al. 2006; Chen et al. 2006; Gou and Chirkova 2007a; Mandreoli et al. 2009]) use twig structure to encode the query expression.

There are four ordered axes in XPath namely, *following*, *following-sibling*, *preceding*, *preceding-sibling* which can effectively express retrievals satisfying order among the chosen elements of XML data. In the context of stored and streaming data, there are very few efforts towards developing systems for processing XPath expressions with ordered axes. Conventional twig structures fail to capture the meaning of ordered axes (detailed analysis is given later in the article). Hence twig-based

algorithms can not be directly extended to handle ordered axes. However, for path expressions having only *child* and *descendant* axes, twig-based algorithms ([Chen et al. 2006; Bruno et al. 2002; Gou and Chirkova 2007a; Chen et al. 2006]) have been shown to outperform systems based on other approaches and formalisms ([Li and Moon 2001; Olteanu 2007]), particularly when the data is recursive. Thus, it would be interesting to investigate if conventional twig representation can be extended to represent path expressions with ordered axes.

In this article, we introduce a new kind of twig structure called Order-aware Twig (OaT) which is capable of effectively representing XPath expressions with ordered axes, in addition to *child* and *descendant* axes. Our contributions are summarized below:

(1) We provide a detailed analysis that brings out the difference between *un-ordered* processing of twigs, *ordered* processing of twigs and processing of XPath expressions having ordered axes.
(2) We show how conventional twigs can be extended to Order-aware Twigs by incorporating constraint edges that capture the meaning of ordered axes.
(3) We introduce an algorithm for producing order-aware twigs from XPath expressions with ordered axes and argue the equivalence of Order-aware Twigs and path expressions containing ordered axes.

We have already established that the framework of Order-aware Twigs is in fact leading to better algorithms for stream processing of XPath queries with ordered axes. Based on this framework, we have proposed a stream processor for the XPath subset containing forward axes [Nizar and Kumar 2008] whose performance is better than the existing algorithm. We have also demonstrated the effectiveness of the framework by developing a stream processor for backward ordered axes [Nizar and Kumar 2009] which also improves on the existing algorithm. While these works use some aspects OaT, the present work formally discusses the framework in full detail with encoding algorithm and proof of correctness.

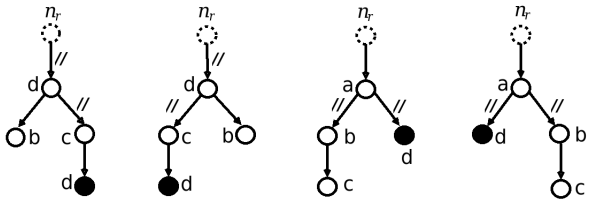
The rest of the article is organized as follows: Section 2 motivates the work. Section 3 formally introduces the concept of ordering and shows how ordering constraints can be effectively represented. Section 3.3 distinguishes between OaT matches and ordered twig matches. In Section 3.4, we discuss the algorithm for translating path expressions to OaTs and prove the equivalence of a path expression and the corresponding OaT generated by the algorithm.

2. BACKGROUND, RELATED WORK AND MOTIVATION

An XML document is a nested collection of elements. Each element is described by a pair of tags — an open tag (<*element-name*>) and a close tag (</*element-name*>), where *element-name* is a user-defined tag name. An element can have other elements, text or a mix of elements and text defined inside it. Elements with empty contents are also supported. The nesting of elements can be arbitrarily deep. The element within which all other elements are nested in is called the *document element*. The logical model of XML document recommended by W3C(World Wide Web Consortium) [Berglund et al. 2007] is an ordered node-labelled tree where tree nodes are labelled with tag names. The element-subelement relationship is depicted using edges between nodes. Order among children of a node represents the sequential order of corresponding elements in the document. The node corresponding to the document element becomes root node of the tree. The root node is also known as *document node*. W3C standard assumes presence of a 'cosmic' root node with the document node as the *only* child.

2.1 Twigs and Twig Match

Conventionally, XPath queries with *child* and *descendant* axes and predicates are effectively modelled as twig queries. A twig is an unordered tree where nodes are labelled with node tests and



(a)Twig T_{1a} (b)Twig T_{1b} (c)Twig T_{2a} (d)Twig T_{2b}

Fig. 1. Sample Twig Structures

directed edges are used to represent axes. Figure 1(a) shows twig T_{1a} representing the path expression /desc::d[ch::b]/desc::c /ch::d[1]. In the figure, edges without labels are called P-C (Parent-Child) edges and they represent *child* axes. Edges with '//' as label are called A-D (Ancestor-Descendant) edges and represent *descendant* axes. There is a root node labelled n_r which always corresponds to the cosmic 'root' node of any XML document tree. The node in black represents the result node, that is, the node corresponding to node test of the *last* axis step of path expression. According to XPath semantics, the result of a path expression is the result of evaluating the last axis step.

Twig processing algorithms identify subtrees in the XML document tree matching the twig where twig match is formally defined as follows:

Definition 1. *Given a twig* $\boldsymbol{Q}$ *and a document* $\boldsymbol{D}$*, a match* μ *of* $\boldsymbol{Q}$ *in* $\boldsymbol{D}$ *is a mapping from nodes of* $\boldsymbol{Q}$ *to the nodes of* $\boldsymbol{D}$ *respecting the following constraints:*

(*1*) *The root node* n_r *is mapped to the cosmic root node of* $\boldsymbol{D}$*.*

(*2*) *A node with label* $\boldsymbol{l}$ *in* $\boldsymbol{Q}$ *is mapped to a node with label* $\boldsymbol{l}$ *in* $\boldsymbol{D}$*.*

(*3*) *A node with label* ***'*'*** *is mapped to any node in* $\boldsymbol{D}$*.*

(*4*) *The parent-child(P-C) and ancestor-descendant(A-D) relationships between nodes in* $\boldsymbol{Q}$ *are satisfied by the mapped nodes. That is, if there is a P-C (resp., A-D) edge from twig node* n_1 *to twig node* n_2*,* $\mu(n_2)$ *is a child (resp., descendant) of* $\mu(n_1)$ *in* D*.*

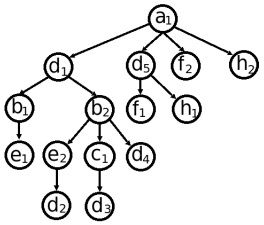
For example, twig T_{1a} in Figure 1(a), which represents the path expression /desc::d[ch::b]/desc::c /ch::d, has two matches in the document D_1 of Figure 2(a) – (d_1, b_1, c_1, d_3) and (d_1, b_2, c_1, d_3) while T_{2b} (Figure 1(c)) representing the expression /ch::a[desc::b/ch::c]/desc::d has five matches – (a_1, b_2, c_1, d_1), (a_1, b_2, c_1, d_2), (a_1, b_2, c_1, d_3), (a_1, b_2, c_1, d_4), (a_1, b_2, c_1, d_5). Note that twig T_{1b} (Figure 1(b)), which is isomorphic to T_{1a} represents the same path expression and has the same set of matches. Similarly, twig T_{2b} of Figure 1(d) is isomorphic to T_{2a} and has the same matches as the latter in document D_1.

Note that, each match according to *Definition 1* is a *structural match*, that is, set of document nodes that satisfy node labels and inter-node relationships of the twig. However, XPath's evaluation model follows navigational semantics and returns the node set after evaluation of the last axis step of the path expression. To be consistent with this XPath evaluation model, we define an evaluation function ε for twigs as follows:

Definition 2. *Let* $\boldsymbol{Q}$ *be a query twig and* $\boldsymbol{s}$ *be its result node. Let* $\{\mu_1, \mu_2, \ldots, \mu_n\}$ *be the set of mappings of* $\boldsymbol{Q}$ *against some document tree* $\boldsymbol{D}$*. Then* $\varepsilon(\boldsymbol{Q},\boldsymbol{D}) = \bigcup_{i=1}^{n} \{\mu_i(\boldsymbol{s})\}$

We overload ε to represent the evaluation of path expressions also.

[1]In this article we use ch, desc, fl, fs, pr and ps as abbreviations for child, descendant, following, following-sibling, preceding and preceding-sibling, respectively. Also we don't use the *abbreviated syntax* of XPath to avoid any possible ambiguities.



(a)Document D_1

Matches:

$\mu(T_{1a}) = (T_{1b}) = \{(d_1,b_1,c_1,d_3), (d_1,b_2,c_1,d_3)\}$

$\mu(T_{2a}) = (T_{2b}) = \{(a_1,b_2,c_1,d_1), (a_1,b_2,c_1,d_2), (a_1,b_2,c_1,d_3), (a_1,b_2,c_1,d_4), (a_1,b_2,c_1,d_5)\}$

Ordered Matches:

$\mu(T_{1a}) = \{(d_1,b_1,c_1,d_3)\}$ $\mu(T_{1b}) = \{\}$

$\mu(T_{2a}) = \{(a_1,b_2,c_1,d_5)\}$ $\mu(T_{2b}) = \{\}$

(b) Twig Matches

Fig. 2. Sample Document and Twig Matches

For instance,

$\varepsilon(/\text{desc::d[ch::b]/desc::c /child::d}, D_1) = \varepsilon(T_{1a}, D_1) = \varepsilon(T_{1b}, D_1) = \{d_3\}$ and,

$\varepsilon(/\text{ch::a[desc::b/ch::c]/desc::d}) = \varepsilon(T_{2a}, D_1) = \varepsilon(T_{2b}, D_1) = \{d_1, d_2, d_3, d_4, d_5\}$.

It may be noted that a twig can have more than one structural match which leads to the same result node. For instance, the twig T_{1a} has two matches in the document D_1 – – (d_1, b_1, c_1, d_3) and (d_1, b_2, c_1, d_3) – and they lead to the evaluation $\{d_3\}$. From a database point of view, the users are interested in the results, i.e, evaluation, and not in the structural matches that lead to them.

2.2 Related Work and Motivation

XPath expressions containing ordered axes – (*following*, *following-sibling*, *preceding*, *preceding-sibling*) – can be used to effectively identify data satisfying ordering constraints. Queries with ordered axes assume special relevance in *document-centric* XML documents. Such documents are characterized by irregular recursive structure and mixed content. Examples are user's manuals and marketing brochures, wordnet collections, news articles, medical information records, to name a few.

Example 1. *In the context of surgical procedures which consist of a number of procedures, actions, observations etc., the order in which events take place is important. For instance the query "Actions taken after it was found that there is a lesion in the bowel" can be expressed as* ***/desc::observations [ch::obs ='lesion in the bowel']/fl::action****.*
As a more concrete example, in the context of the XML document representing a journal article, the path expression ***/desc::section[ch::name= 'Motivation']/fl::figure*** *returns all figure elements that appear in the article after the section titled* ***Motivation****, Similarly, the expression* ***/desc::section [ch:: name = 'Datasets']/pr::section*** *returns all sections that appear before the section titled* ***'Datasets'*** *excluding sections in which the* ***'Datasets'*** *section is nested in.*

Conventional twig structures fail to convey the semantics of ordered axes as they do not impose any order on the selected nodes. Note that the relative order among children of a node in the twig does not specifically say anything about the order of occurances of nodes in the document. When we consider XPath expressions that do constrain the order of the selected nodes, twigs would be inadequate to model such expressions. For instance, we can not represent the XPath query /ch::a/desc::b/ch::c/fl::d as twig of Figure 1(c) as the twig retrieves d-nodes in the document irrespective of their position with respect to b-node or c-node.

There is a wide array of proposals for processing XPath expressions consisting of *child* and *descendant* axes and predicates containing these axes against streaming and stored data. Interestingly enough, the majority of these algorithms represents the path expression as twig and systematically finds matches ([Bruno et al. 2002; Lu et al. 2005; Chen et al. 2005; Moro et al. 2005; Chen et al. 2006; Chen et al. 2006; Gou and Chirkova 2007a; Mandreoli et al. 2009]). Twig encoding is quite convenient for query processing as the process is closely tied to the tree model of XML. As twigs are

tree structures, query matching boils down to finding subtrees in the XML tree. A detailed account of twig-based processing of XPath expressions on XML repositories is given in [Gou and Chirkova 2007b].

The research efforts towards handling ordered axes are limited to systems such as [Grust et al. 2004], [Olteanu 2007] and [Onizuka 2010]. These systems do not use twig encoding of XPath expressions. The system proposed in [Grust et al. 2004] is built on the top of a relational engine. Each element in the XML document is encoded as a tuple along with its depth in the document tree and position in the mapping of the document into a (pre-order, post-order) plane. The system constructs an index structure known as *XPath accelerator*, closely tracking the semantics of XPath and taking into consideration the inherent tree-topology of XML document. XPath queries are transformed into SQL expressions that use joins to retrieve results from the relational database. SPEX [Olteanu 2007] is a stream query processing system which compiles the XPath query into a network of push down transducers. The transducer system responds to the events from the input XML stream to find matches for the path expression. The system proposed in [Onizuka 2010] uses a double layer NFA.

Systems like [Chan et al. 2002; Wang et al. 2003; Rao and Moon 2004; Lu et al. 2005; Prasad and Kumar 2005; Kwon et al. 2005; Raj and Kumar 2007; Jiang et al. 2008; Mandreoli et al. 2009] propose *ordered matching* of twig queries. In an ordered twig match, the matched nodes should respect the left-to-right order of children of a given twig node. Formally, it is a twig match as per *Definition 1* with the following additional constraint:

5. *If a twig node* b *appears after node* a *in preorder, then* $\mu(\mathsf{b})$ *should appear, in document order, after* $\mu(\mathsf{a})$ *and postorder rank of* $\mu(\mathsf{b})$ *in the document tree is greater than postorder rank of* $\mu(\mathsf{a})$.

For instance, *ordered match* of the twig query T_{2a} of Figure 1 yields one match – (a_1, b_2, c_1, d_5) – in document D_1 (Figure 2(a)) while *ordered match* of T_{2b}, which is isomorphic to T_{2a}, returns no matches.

Thus, ordered twig query processing refers to a *way of processing* twig representation of a path expression, where the matching algorithm, in addition to considering the node labels and inter-node relationships in the twig, respects the topology of the twig.

Such naive extensions of twig processing do not hold an integrated view of ordered axes processing. Note that XPath queries can have axis steps appearing in a sequence in all possible combinations and query processors need to use such path expressions as starting points for query evaluation. The extended twig processors do not consider all the ordering constraints that ordered axes might bring in. They operate in special ways on twigs which were originally introduced as an encoding for path expressions with unordered forward axes. We will elaborate on this in Section 3.3.

Twig-based algorithms rely on the accuracy and compactness with which twigs encode the constraints imposed by path expressions with *child* and *descendant* axes and predicates. These algorithms, particularly those for stream processing, effectively use twig encoding and semantics of path expressions the twigs represent for pruning out execution paths and quickly identifying of result node matches. We believe that a compact and accurate encoding of path expressions with ordered axes built on the top of twigs can lead to algorithms for ordered axes processing that can make effective use of conventional twig processing strategies and semantics of ordered axes to achieve efficiency.

3. ORDER-AWARE TWIGS

In this section, we formally introduce OaTs and their matching. We assume that the path expressions do not contain disjunctions and negations of predicates. We believe that, as in the case of un-ordered twigs, these cases can be handled by extending the basic algorithms with the support for boolean evaluation of predicates. Note that, without loss of generality, conjunction of predicates in path

expressions can be replaced by a sequence of individual predicates. Given below is the (abstract) syntax of the expressions we consider in this article. The subset is denoted as $XPATH^{ch,desc,\rightarrow,[\,]}$. We have excluded value based comparisons as they have no effect on the concepts developed.

pathExpr	::=	*startExpr expr*
startExpr	::=	*startAxis::nodeTest* |
		startAxis::nodeTest '['pred']'
expr	::=	*axisStep expr* | *axisStep*
axisStep	::=	*axis::nodeTest* |
		axis::nodeTest'['pred']'
startAxis	::=	*'child'* | *'descendant'*
axis	::=	*'child'* | *'descendant'* |
		'following' | *'following-sibling'* |
		'preceding' | *'preceding-sibling'*
nodeTest	::=	*tagname* | *'*'*
pred	::=	*expr*

3.1 LR and SLR Orderings

In order to represent $XPATH^{ch,desc,\rightarrow,[\,]}$–expressions, we generalize twigs by adding additional ordering constraints. Each such constraint is specified either between a pair of nodes appearing in two distinct downward paths from the twig root n_r (*LR (Left-to-Right)-Ordering*) or between a pair of nodes which are P-C children (i.e., twig nodes connected to their parent node using P-C edges) of branching nodes (*SLR (Sibling Left-to-Right)-Ordering*).

3.1.1 *LR Ordering*

Definition 3. *Let x and y be nodes in a twig T and are descendants of the n_r along two distinct downward paths P_1 and P_2, respectively. Let p and q be the nodes corresponding to x and y in a match of T in some document tree D as per Definition 1. An LR ordering from x to y is an ordering constraint specifying that q should appear after p in document order in D, but should not be a descendant of p.*

Intuitively, LR ordering from node x to node y specifies y-descendants of the root which are neither appearing in preorder before x nor are descendants of x. LR ordering from node x to node y is denoted as x≪y. x is called the *tail* of LR ordering and y is called the *head* of LR ordering. In graphical notation, LR ordering from node x to be node y is denoted using a dotted arrow from x to y.

LR ordering can effectively represent the semantics of *following* and *preceding* axes.

Example 2. *Figure 3 illustrates how LR ordering can be used to represent following and preceding axes. The XPath expression /ch::a/desc::b/ch::c/fl::d looks for d-nodes which are appearing after the*

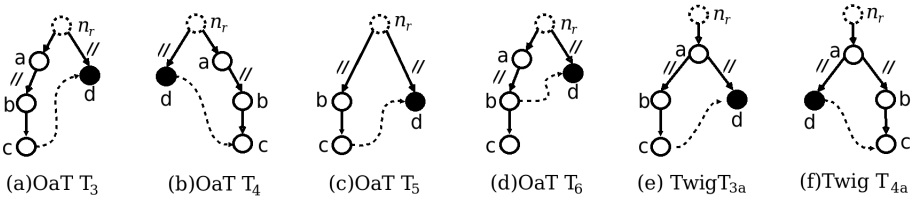
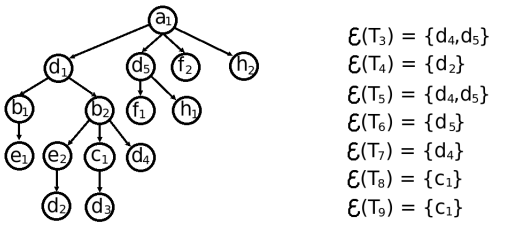


Fig. 3. Illustrating LR ordering



(a)Document D_1 (b)Evaluations on D_1

Fig. 4. Document and Evaluations of OaTs in Figures 3 and 5

close-tag[2] *of c-node child of a b-node descendant of the document node a. OaT* T_3 *of Figure 3(a) represents this path expression. In* T_3 *there is an LR ordering from node* c *to node* d *indicating that in a structural match of* T_3*, a d-node should appear after close-tag of c-node. It is clear that semantics of* T_3 *is the same as that of the expression. For instance,* /ch::a/desc::b/ch::c/fl::d *when evaluated on document* D_1 *of Figure 4(a) (which is the same as Figure 2(a)) leads to the node set* $\{d_4, d_5\}$*. Twig* T_3 *has two structural matches in* D_1 *–* (a_1, b_2, c_1, d_4) *and* (a_1, b_2, c_1, d_5) *leading to the evaluation* $\{d_4, d_5\}$*. Thus,* ϵ*(/ch::a/desc::b/ch::c/fl::d,* D_1*) =* $\epsilon(T_3, D_1)$*.*
Similarly, /ch::a/desc::b/ch::c/pr::d *looks for d-nodes preceding, in document order, a c-node child of b-node descendant of the root. This means that a valid result node should have a c-node (with a parent node b, which is also descendant of the document node with label a) appearing after it, but not as a descendant. Twig* T_4 *(Figure 3(b)) with the LR ordering has the same interpretation. In document* D_1*, both the path expression and twig* T_4 *have the same evaluation –* $\{d_2\}$*.*
Twig T_5 *in Figure 3(c) represents the path expression* /desc::b/ch::c/fl::d*. Note that twig* T_6 *(Figure 3(d)) represents the expression* /ch::a/desc::b[ch::c]/fl::d *and is different from twig* T_3*.*

Note that T_{3a} (Figure 3(e)) is an alternate representation of T_3 (Figure 3(a)) where node d is connected to node a. This is correct as node a will invariably match with the document node, the only element node child of the 'cosmic' root node of the document tree (since there is a P-C edge from n_r to node a) and therefore every other node in the twig can match with descendants of the document node only. Following a similar argument, T_{4a} (Figure 3(f)) is equivalent to T_4 (Figure 3(b)).

3.1.2 *SLR Ordering*

Definition 4. *Let x and y be sibling nodes in a twig T connected to its common parent b by a P-C edge (that is, x and y are P-C children of their parent). Let p and q be nodes corresponding to x and y in a match of T in some document tree D as per Definition 1. An SLR ordering from x to y specifies that p and q are siblings in D and q appears after p in document order.*

SLR ordering from node x to node y is denoted as x<y. x is called the *tail* of SLR ordering and y is called the *head* of SLR ordering. In graphical notation, SLR ordering from node a to be node b is denoted using a solid arrow from x to y.

SLR ordering can effectively represent the constraints of *following-sibling* and *preceding-sibling* axes.

Example 3. *Figure 5 shows how SLR ordering is used to represent the ordering constraints of following-sibling and preceding-sibling axes.* /desc::b/ch::c/fs::d *looks for d-nodes which are children of b-node but appear after a c-child. Twig* T_7 *of Figure 5(a) with an SLR ordering from c to d has the*

[2]Here 'close-tag' means close-tag of the element denoted by the node in the textual representation of the document tree.

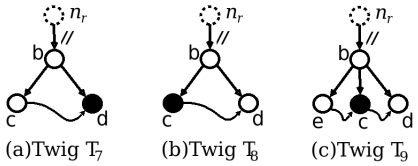


Fig. 5. Illustrating SLR ordering

same semantics. Similarly, twig T_8 (Figure 5(b)) represents the path expression /desc::b/ch::d/ps::c while T_9 of Figure 5(c) represents /desc::b/ch::e/fs::c[fs::d].

The examples above illustrate that both forward ordered axes (*following* and *following-sibling*) and backward ordered axes (*preceding*, *preceding-sibling*) can be represented effectively using LR- and SLR edges. Later, in Section 3.5, we claim that twig structures augmented with LR edges, SLR edges and closure edges (to be introduced shortly) can be used to correctly represent the constraints of XPath expressions with ordered axes.

3.2 Closure Edges

The basic twigs need to be further extended to handle XPath expressions containing an axis step with *following-sibling* or *preceding-sibling* axis that appears (inside predicate or otherwise) *immediately* after an axis step with *descendant* axis. For example, in the query /ch::a/desc::f/fs::h axis step fs::h appears immediately after /desc::f. Here h can be a right-sibling of an f-child of a or right-sibling of f-child of descendant of a. A straightforward way to represent the query is to extend the twig in Figure 6 (a), which represents the expression /ch::a/desc::f, to two twigs in Figure 6(b). The downside of this approach is that the number of twigs for a query increases exponentially with such sibling axis occurrences and it can lead to a proportional increase in time and space requirements of query evaluation. As the OaTs representing the query are structurally the same except for the deviation due to the sibling axis occurrences, the same matching steps need to be repeated extensively for all the twigs. Further, additional union operation is needed to get the final results. We circumvent this problem by introducing a new type of edge known as *closure edge* which is defined below:

Definition 5. *A **closure edge** from node n_1 to n_2, where $n_2.label =$'*' represents the relationship constraint that the data node d_{n_2} matching n_2 is the same as the data node d_{n_1} that matches n_1 or any descendant node of d_{n_1}. Node n_2 is called the **closure node**.*

The twig in Figure 6(c) shows how a closure edge can be used to represent the path expression /ch::a/desc::f /fs::h using a single twig. Here the *zig-zag* edge from the a-node to the *-node (that is closure node) is the closure edge. Note that, only the first sibling axis that appears after *descendant* axis raises the need for closure edge. For instance, if we extend the path expression /ch::a/desc::f/fs::h to /ch::a/desc::f/fs::h[fs::g], the equivalent twig can be obtained by extending the twig in Figure 6(c) to the one shown in Figure 6(d).

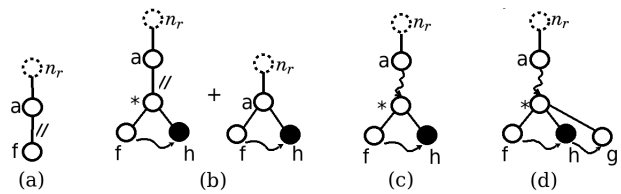


Fig. 6. Illustrating Closure-Edge

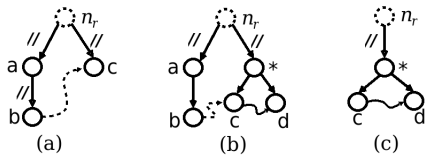


Fig. 7. Cases where closure edge is not needed

Recall from Section 3.1 that corresponding to a *following* or *preceding* axis, an A-D edge is added to the root node n_r. However, there is no need of a closure edge when a *following-sibling* or *preceding-sibling* axis appears immediately after one of the above axes. This can be best explained using an example. Figure 7(a) shows twig for the path expression /desc::a/desc::b/fl::c. If the expression is extended to /desc::a/desc::b/fl::c/fs::d, it can be represented using the twig in Figure 7(b). Note that the root node n_r is connected to the *-node using an A-D edge, not using a closure edge for, the root node of the document has only *one* child – the document node.
A similar situation arises if the path expression starts with a *descendant* axis and followed by a *following-sibling* or *preceding-sibling* axis as illustrated using twig in Figure 7(c) which represents the path expression /desc::c/fs::d.

To summarize, an Order-aware Twig is a tree structure rooted at a node labelled n_r known as the root of the OaT. There are three types of relationship edges – P-C edge, A-D edge and closure edge – and two types of constraint edges – LR edge and SLR edge.

We now define OaT matching by extending *Definition 1*

Definition 6. *Given a twig* $\boldsymbol{Q}$ *and a document* $\boldsymbol{D}$*, a match* μ *of* $\boldsymbol{Q}$ *in* $\boldsymbol{D}$ *is a mapping from nodes of* $\boldsymbol{Q}$ *to the nodes of* $\boldsymbol{D}$ *respecting the following constraints:*

(*1*) *The root node* n_r *is mapped to the cosmic root node of* $\boldsymbol{D}$*.*
(*2*) *A node with label* $\boldsymbol{l}$ *in* $\boldsymbol{Q}$ *is mapped to a node with label* $\boldsymbol{l}$ *in* $\boldsymbol{D}$*.*
(*3*) *A node with label* ***'*'*** *is mapped to any node in* $\boldsymbol{D}$*.*
(*4*) *The parent-child(P-C) and ancestor-descendant(A-D) relationships between nodes in* $\boldsymbol{Q}$ *are satisfied by the mapped nodes. That is, if there is a P-C (resp., A-D) edge from OaT node* n_1 *to OaT node* n_2*,* $\mu(n_2)$ *is a child (resp., descendant) of* $\mu(n_1)$ *in D.*
(*5*) *LR ordering and SLR ordering constraints specified on nodes in* $\boldsymbol{Q}$ *are satisfied by the mapped nodes. That is, if there is an LR edge from OaT node* n_1 *to OaT node* n_2*,* $\mu(n_2)$ *appears in document order after* $\mu(n_1)$ *in D but not as a descendant of* $\mu(n_1)$*. Similarly, if there is an SLR edge from OaT node* n_1 *to OaT node* n_2*,* $\mu(n_2)$ *appears in document order after* $\mu(n_1)$ *in D and* $\mu(n_1)$ *and* $\mu(n_2)$ *have a common parent.*

Definition of evaluation function ε (*Definition 2*) can be similarly extended.

3.3 OaT Match *vs* Ordered Twig Match

In this section, we illustrate how the constraints captured by OaTs are different from the constraints maintained by ordered twig matching. Figure 8 shows a twig and three OaTs. Note that all the twigs have the same P-C and A-D constraints.

Ordered Match of Twig T_{10} (Figure 8(a)) leads to one structural match in document D_2(Figure 8(b)) – $a_1b_1c_1d_2e_2$. OaT T_{11} (Figure 8(c)) with an LR edge from node b to d represents the same constraint as the one ordered twig match of T_{10} maintains and hence returns the same structural match as result. However, the constraints imposed by OaTs T_{12} and T_{13} (Figures 8(d)&(e)) can not be maintained by ordered match of twig T_{10}. OaT T_{12} has two structural matches in document D_2 – $a_1b_1c_1d_1e_1$ and

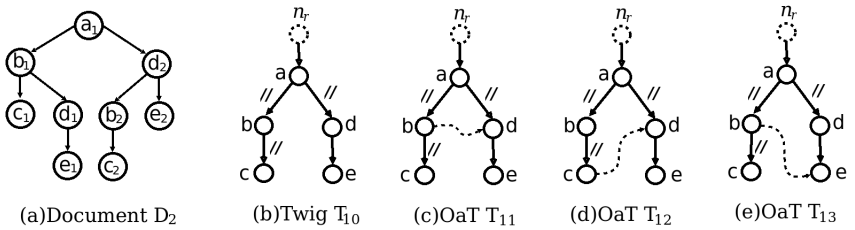


Fig. 8. OaT Match and Ordered Twig Match

$a_1b_1c_1d_2e_2$. Similarly, OaT T_{13} has two matches – $a_1b_1c_1d_2e_2$ and $a_1b_2c_2d_2e_2$. This clearly illustrates that ordering constraints imposed by OaTs can not always be maintained by ordered matching of conventional twigs.

3.4 Translating $XPATH^{ch,desc,\rightarrow,[\ ]}$ to OaTs

In this section we present a systematic way of translating an $XPATH^{ch,desc,\rightarrow,[\ ]}$–expression to an OaT. Algorithm 1 performs the translation process. Intuitively the algorithm accepts a partial OaT which is the translation of the expression up to certain position and extends the OaT by translating the next predicate or axis step to generate a new OaT. *expr* is the expression to be translated. Recall that an expression consists of a sequence of axis steps and each axis step consists of an axis with node-test and an optional list of predicates following it. Each axis step is chopped into *units* where a unit is either (i) an axis with node-test (a *simple step*) or (ii) a predicate. The translation process is done one unit at a time. The OaT T is assumed to be global to the translation algorithm. The argument *cNode* of *translateExpr()* represents the node in the current OaT from where extension should start by translating the next unit in *expr*. We call this node the *context node* of T. *rNodetest* represents the node-test in *expr* corresponding to the result node. It is the node-test of the last axis step that appears *outside* any predicate.

Algorithm 1: Translating Path Expression to OaT

```
   Input: expr: path expression, cNode: context node, rNodetest: result node-test, n_r: root of the
          OaT T
 1 translateExpr(expr, cNode, rNodetest, n_r)
 2 begin
 3     nextUnitBegin← 0;
 4     <nextUnit,nextUnitBegin> ← getNextUnit(expr,nextUnitBegin)
 5     c ← cNode
 6     while (nextUnit ≠ NULL) do
 7         if nextUnit is a predicate then
 8             Strip off '[' and ']' from nextUnit
 9             translateExpr(nextUnit, c, " ", n_r)
10         else //A simple Step
11             c ← processSimpleStep(nextUnit, c, rNodetest,n_r)
12         <nextUnit,nextUnitBegin> ← getNextUnit(expr,nextUnitBegin)
13     Return
14 end
```

The translation process first constructs an initial OaT consisting of the root node n_r. The algorithm invokes the function *getNextUnit()* that accepts a path expression and the beginning of the next unit in the expression. It returns the next unit and beginning of the subsequent unit.

translateExpr() needs slightly different processing depending on whether the next unit is a predicate or a simple step, because new context node of the OaT after extension can be different in each case. For example, suppose that T represents the partial OaT after processing an expression up to $X :: N$ and that the context node c of T is the node representing N. After processing every predicate in the predicate list, if any, that is associated to $X : N$, c continues to be the context node. In case the next unit is an axis step, say $Y :: P$, after processing the unit, the node in T corresponding to P becomes the new context node.

Algorithm 2: Axis Step Translation

```
   Input: X :: N: axis step, c: context node, rNodetest: result node-test, n_r: root of T
 1 processSimpleStep(X :: N, c, rNodetest, n_r)
 2 begin
 3     n ← newNode(N)
 4     if N = rNodetest then mark n as resultnode
 5     switch (X) do
 6         case child
 7             Add P-C edge from c to n
 8         case descendant
 9             Add A-D edge from c to n
10         case following-sibling,preceding-sibling
11             p ← parent(T, c)
12             if there is a P-C edge from p to c then
13                 Add P-C edge from p to n
14             else
15                 n_1 ← newNode('*')
16                 Add P-C edges from n_1 to c and n_1 to n
17                 if p = n_r then
18                     Add an A-D edge from p to n_1
19                 else
20                     Add a closure edge from p to n_1
21                 if X is following-sibling then
22                     Add an SLR ordering c < n
23                 else
24                     Add an SLR ordering n < c
25         case following
26             Add A-D edge from n_r to n; Add LR ordering c ≪ n
27         case preceding
28             Add A-D edge from n_r to n; Add LR ordering n ≪ c
29     Return n
30 end
```

A predicate (after stripping off '[ ]') is a path expression that is a sequence of axis steps, with arbitrarily deep nesting due to presence of other predicates. Thus *translateExpr()* can be invoked recursively on the path expression representing the predicate (line 9). Since a result node-test is not relevant for a predicate, we pass a dummy value for this invocation. Note that, the context node of the OaT will not change after processing a predicate and hence the recursive invocation of *translateExpr()* need not return infomation about the new context node.

The simple step processing routine (Algorithm 2) accepts as input the next axis step of the form $X :: N$, the current context node, root of the OaT T and $rNodeTest$. It returns the new node added in T to the calling routine. It uses an auxiliary routine *parent*() which returns parent of the a given

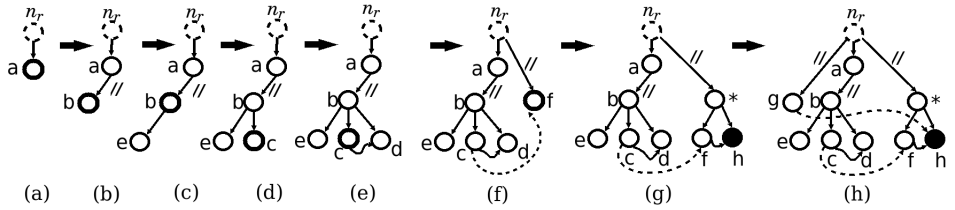


Fig. 9. Translating /ch::a/desc::b[ch::e]/ch::c[fs::d]/fl::f/fs::h[pr::g] to OaT

node in the OaT T. Processing of *child* and *descendant* axes are straightforward. In the case of *following* axis, the new node is added as a descendant of n_r and an LR ordering is set up from context node to the new node. Processing of *preceding* axis is done symmetrically.

Processing of *following-sibling* and *preceding-sibling* axes involves handling the special case where the axis appears *immediately* after *descendant* axis or *following* and *preceding* axes. Lines 15–20 show steps for handling the special case for *following-sibling* and *preceding-sibling* axes.

Figure 9 shows snapshots of translating the path expression, /ch::a/desc::b[ch::e]/ch::c[fs::d]/fl::f /fs::h[pr::g] to OaT. In each snapshot, the node with thick border represents the context node at that point during processing.

The time requirement of the translation process is linear on the number of axis steps including those appearing in predicates. Note that, for *child* and *descendant* axis, the mapping involves setting up a P-C edge or A-D edge between the context node and the new node, as the case may be. For *following*, and *preceding* axes the algorithm establishes an LR ordering in addition to setting up an A-D edge from the root of the OaT to the newly created node. The sibling axis mapping is done by setting up SLR ordering. For the special case of sibling axes, the algorithm adds a new '*'-node, a closure edge/A-D edge and SLR ordering. Thus the translation time remains linear.

3.5 Equivalence of $XPATH^{ch,desc,\rightarrow,[\ ]}$ and OaT

We now establish the equivalence between an $XPATH^{ch,desc,\rightarrow,[\ ]}$–expression and its OaT using semantics of P-C, A-D and closure edges and LR and SLR orderings, and steps of the algorithm.

Lemma 1. *Let T be the OaT generated by Algorithm 1 from the $XPATH^{ch,desc,\rightarrow,[\ ]}$–expression P. Let D be an arbitrary XML document and $\epsilon(P, D)$ and $\epsilon(T, D)$ are evaluation functions for P and D, respectively. Then $\epsilon(P, D) = \epsilon(T, D)$.*

Proof (by induction on number of axis steps): As predicates are path expressions, the equivalence established for a linear path expression (i.e., expression without predicates) can be extended to a path expression with predicates. Thus, without loss of generally, we can assume that P is a linear path expression. Let P_m represent a linear path expression with m axis steps and let T_m represent the OaT obtained by translating P_m. We assume that n_a denotes the OaT node corresponding to node-test a.

Basis: If $m = 1$, P_m can be */child::a* or */descendant::a* only. */child::a* returns *maximum* one match which is the document node of D, if its label is the same as a. For */child::a*, the while loop in $TranslateExpr()$ iterates once and invokes *processSimpleStep* (Algorithm 2). The routine stops at line 6 and returns T_1 which is the root node n_r connected to n_a by a P-C edge. By the semantics of P-C edge, this OaT has one structural match (according to *Definition 6*) containing the root of the document and document node if the document node's label is a leading to the evaluation containing the document node only. Similar arguments hold for */descendant::a* except that evaluation of T_1 contains all tree nodes labelled a. Hence $\epsilon(P_1, D) = \epsilon(T_1, D)$.

Induction Step: A path expression P_m can be written in the form $P_{m-1}/X :: b$ where X can be one of *child*, *descendant*, *following*, *preceding*, *following-sibling* and *preceding-sibling*.

Let a represent node-test of the last axis step in P_{m-1} and T_{m-1} be the OaT equivalent to P_{m-1}. Then by induction hypothesis, $\epsilon(P_{m-1}, D) = \epsilon(T_{m-1}, D)$.

The first two cases (*child* and *descendant*) can be argued in a way similar to the basis step.

—*following*: $X :: b$ uses each node $t \in \epsilon(P_{m-1}, D)$ as context node to get matches u for $\epsilon(P_m, D)$ such that $u.label = b$ and u is descendant of the root node and appears after t in document order, but not a descendant t. The algorithm generates T_m by extending the OaT T_{m-1} by adding n_b as a descendant of n_r and adding an LR ordering from n_a to n_b (Line 26, Algorithm 2). T_m returns all structural matches such that each match will contain a node $t \in \epsilon(T_{m-1}, D)$ and satisfies the newly added L-R ordering. By the semantics of LR ordering and A-D edge $\epsilon(P_m, D) = \epsilon(T_m, D)$.

—*preceding*: Similar to the above.

—*following-sibling*: Let the last two axis steps in P_{m-1} be *U::p* and *W::a*. If W is *child*, each t in the node set $\epsilon(P_{m-1}, D)$ is a child of some $u \in \epsilon(P_{m-2}, D)$. Thus, for each new node v identified by $X :: b$ using t as context node, v is a right sibling of t and u is the parent of v also. In the algorithm, a P-C edge is added from n_p to n_b and an SLR ordering $n_a < n_b$ is introduced. T_m returns all structural matches such that each match will contain a node $t \in \epsilon(T_{m-1}, D)$ and satisfies the newly added constraints. Therefore, $\epsilon(P_m, D) = \epsilon(T_m, D)$.
If W is *descendant*, each t in the node set $\epsilon(P_{m-1}, D)$ is at an arbitrary depth from the some $u \in \epsilon(P_{m-2}, D)$. Thus, for each new node v identified by $X :: b$ using t as context node, v is a right sibling of t and the common parent of t and v can be a node in the path from u to t, *excluding* t. In the algorithm, the A-D edge from n_p and n_a is replaced by a closure edge from n_p to a new wildcard ('*') node and n_a and n_b are connected to the new node with P-C edges. An SLR ordering $n_a < n_b$ is also introduced. T_m returns all structural matches such that each match will contain a node $t \in \epsilon(T_{m-1}, D)$ and satisfies the newly added constraints. By the semantics of closure-edges and SLR ordering, $\epsilon(P_m, D) = \epsilon(T_m, D)$.
If W is *following*, each $t \in \epsilon(P_{m-1}, D)$, appears after some $u \in \epsilon(P_{m-2}, D)$ at an arbitrary depth from the root of D but not below u. Thus, for each node v identified by $X :: b$, the common parent for t and v is a node in the path from root to t excluding t and *excluding the cosmic root as it has no child other than the document node*. The algorithm, while mapping *W::a*, had introduced an A-D edge from n_r to n_a. Now, the algorithm replaces the A-D edge from n_r and n_a by an A-D edge from n_r to a new wildcard ('*') node (line 15) and n_a and n_b are connected to the new node with P-C edges. An SLR ordering $n_a < n_b$ is also introduced. And each structural match of T_m contains a node $t \in \epsilon(T_{m-1}, D)$ and satisfies the newly added constraints. By semantics the of SLR ordering, P-C and A-D edges, $\epsilon(P_m, D) = \epsilon(T_m, D)$. A symmetric case holds if W is *preceding*. Also, the cases where W is either *following-sibling* or *preceding-sibling* can be argued is way similar to the one for *child*.

—*preceding-sibling*: Similar to the above.

Hence the claim.

4. CONCLUSION

In this article, we analyze the issues arising out of representing XPath queries with ordered axes using tree structures. We establish that conventional twig structures are not capable of capturing the constraints involved. While there are some attempts to evaluate twig patterns by taking their topology into account, none of them propose an integrated view of modelling and evaluating XPath expressions with ordered axes. In this scenario, the new type of twig structures called Order-aware Twigs (OaTs), proposed in this article, are capable of effectively representing path expressions with ordered axes.

We introduce two types of ordering constraints namely, LR ordering and SLR ordering and a new relationship constraint called closure relationship which together can encode any path expression with ordered axes. We also formally prove the equivalence of path expressions and their representation in the new framework. We have already proposed algorithmic solutions for processing XPath subsets containing ordered axes based on the new framework. The results show that the approach of *holistic* evaluation of XML queries with ordered axes built around the proposed model is promising.

REFERENCES

Berglund, A., Boag, S., Chamberlin, D., Fernández, M. F., Kay, M., Robie, J., and Siméon, J. *XML Path Language (XPath) 2.0*, 2007. available at http://www.w3.org/TR/2007/REC-xpath20-20070123/.

Berglund, A., Fernández, M., Malhotra, A., Marsh, J., Nagy, M., and Walsh, N. *XQuery 1.0 and XPath 2.0 Data Model (XDM)*, 2007. available at http://www.w3.org/TR/xpath-datamodel/.

Boag, S., Chamberlin, D., Fernández, M. F., Florescu, D., Robie, J., and Siméon, J. *XQuery 1.0: An XML Query Language*, 2007. available at http://www.w3.org/TR/2007/REC-xquery-20070123/.

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

Chan, C. Y., Felber, P., Garofalakis, M. N., and Rastogi, R. Efficient filtering of xml documents with xpath expressions. *VLDB Journal* 11 (4): 354–379, 2002.

Chen, S., Li, H.-G., Tatemura, J., Hsiung, W.-P., Agrawal, D., and Candan, K. S. Twig^{2}stack: Bottom-up processing of generalized-tree-pattern queries over XML documents. In *Proceedings of the International Conference on Very Large Data Bases*. Seoul, Korea, pp. 283–294, 2006.

Chen, T., Lu, J., and Ling, T. W. On boosting holism in XML twig pattern matching using structural indexing techniques. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Baltimore, Maryland, USA, pp. 455–466, 2005.

Chen, Y., Davidson, S. B., and Zheng, Y. An efficient xpath query processor for XML streams. In *Proceedings of the International Conference on Data Engineering*. Atlanta, Georgia, USA, pp. 79–91, 2006.

Gou, G. and Chirkova, R. Efficient algorithms for evaluating XPath over streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Beijing, China, pp. 269–280, 2007a.

Gou, G. and Chirkova, R. Efficiently querying large xml data repositories: A survey. *IEEE Trans. on Knowl. and Data Eng.* 19 (10): 1381–1403, 2007b.

Grust, T., van Keulen, M., and Teubner, J. Accelerating xpath evaluation in any rdbms. *ACM Trans. Database Syst.* 29 (1): 91–131, 2004.

Jiang, J., Chen, G., Shou, L., and 0005, K. C. Otjfast: Processing ordered xml twig join fast. In *Proceedings of the IEEE Asia-Pacific Services Computing Conference*. Yilan, Taiwan, pp. 1289–1294, 2008.

Kwon, J., Rao, P., Moon, B., and Lee, S. FiST: Scalable XML document filtering by sequencing twig patterns. In *Proceedings of the International Conference on Very Large Data Bases*. Trondheim, Norway, pp. 217–228, 2005.

Li, Q. and Moon, B. Indexing and querying xml data for regular path expressions. In *Proceedings of the International Conference on Very Large Data Bases*. Roma, Italy, pp. 361–370, 2001.

Lu, J., Chen, T., and Ling, T. W. Tjfast: Effective processing of xml twig pattern matching. In *Proceedings of the International Conference on World Wide Web (Special interest tracks and posters)*. Chiba, Japan, pp. 1118–1119, 2005.

Lu, J., Ling, T. W., Yu, T., Li, C., and Ni, W. Efficient processing of ordered xml twig pattern. In *Proceedings of the International Conference on Database and Expert Systems Applications*. Copenhagen, Denmark, pp. 300–309, 2005.

Mandreoli, F., Martoglia, R., and Zezula, P. Principles of holism for sequential twig pattern matching. *The VLDB Journal* 18 (6): 1369–1392, 2009.

Moro, M. M., Vagena, Z., and Tsotras, V. J. Tree-pattern queries on a lightweight xml processor. In *Proceedings of the International Conference on Very Large Data Bases*. Trondheim, Norway, pp. 205–216, 2005.

Nizar, A. and Kumar, P. S. Efficient evaluation of forward xpath axes over xml streams. In *Proceedings of the International Conference on Management of Data*. Mumbai, India, pp. 222–233, 2008.

Nizar, A. and Kumar, P. S. Ordered backward xpath axis processing against xml streams. In *Proceedings of the International VLDB XML Database Symposium*. Lyon, France, pp. 1–16, 2009.

Olteanu, D. SPEX: Streamed and progressive evaluation of XPath. *IEEE Trans. Knowl. Data Eng.* 19 (7): 934–949, 2007.

Onizuka, M. Processing xpath queries with forward and downward axes over xml streams. In *Proceedings of the International Conference on Extending Database Technology*. Lausanne, Switzerland, pp. 27–38, 2010.

Prasad, K. H. and Kumar, P. S. Efficient indexing and querying of XML data using modified prufer sequences. In *Proceedings of the International Conference on Information and Knowledge Management*. Bremen, Germany, pp. 397–404, 2005.

Raj, A. and Kumar, P. S. Branch sequencing based XML message broker architecture. In *Proceedings of the International Conference on Data Engineering*. Istanbul, Turkey, pp. 217–228, 2007.

Rao, P. and Moon, B. PRIX: Indexing and querying XML using prüfer sequences. In *Proceedings of the International Conference on Data Engineering*. Boston, MA, USA, pp. 288–300, 2004.

Wang, H., Park, S., Fan, W., and Yu, P. S. ViST: a dynamic index method for querying XML data by tree structures. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. San Diego, California, USA, pp. 110–121, 2003.