Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Systems and Software 79 (2006) 180–190

The Journal of Systems and Software

www.elsevier.com/locate/jss

Efficient evaluation of linear path expressions on large-scale heterogeneous XML documents using information retrieval techniques

Young-Ho Park [a,*], Kyu-Young Whang [a], Byung Suk Lee [b], Wook-Shin Han [c]

[a] *Department of Computer Science and Advanced Information Technology Research Center (AITrc), Korea Advanced Institute of Science and Technology (KAIST), 373-1, Koo-Sung Dong, Yoo-Sung Ku, Daejeon 305-701, Republic of Korea*

[b] *Department of Computer Science, University of Vermont, Burlington, VT 05405, USA*

[c] *Department of Computer Engineering, Kyungpook National University, Daegu 702-701, Republic of Korea*

Received 28 May 2004; received in revised form 9 May 2005; accepted 9 May 2005
Available online 5 July 2005

Abstract

We propose XIR-Linear, a method for efficiently evaluating linear path expressions (LPEs) on large-scale heterogeneous XML documents using information retrieval (IR) techniques. LPEs are the primary form of XPath queries, and their evaluation techniques have been researched actively. XPath queries in their general form are partial match queries, and these queries are particularly useful for searching documents of heterogeneous schemas. Thus, XIR-Linear is geared for partial match queries expressed as LPEs. XIR-Linear has its basis on existing methods using relational tables (e.g., XRel, XParent), and drastically improves their efficiency using the inverted index technique. Specifically, it indexes the labels in label paths (i.e., sequences of node labels) like keywords in texts, and finds the label paths matching the LPE far more efficiently than string match used in the existing methods. We demonstrate the efficiency of XIR-Linear by comparing it with XRel and XParent using XML documents crawled from the Internet. The results show that XIR-Linear outperforms XRel and XParent by an order of magnitude with the performance gap widening as database size grows.
© 2005 Elsevier Inc. All rights reserved.

Keywords: XML; Inverted indexes; Partial match queries; Information retrieval

1. Introduction

Recently, there have been significant research on processing queries against XML documents (XML, 2004). To our knowledge, however, most of them considered only a limited number of documents with a fixed schema, and thus, are not suitable for large-scale applications dealing with heterogeneous schemas—such as an Internet search engine (Naughton et al., 2001; Xyleme, 2004). A novel method is needed for these applications, and we address it in this paper.

Partial match queries in XPath (Clark and DeRose, 1999) can be particularly useful for searching XML documents when their schemas are heterogeneous while only partial schema information is known to the user (Aboulnaga et al., 2001; Petrou et al., 1999). Here, a partial match query is defined as the one having the descendent-or-self axis "//" in its path expression. A full match query (Mandreoli et al., 2002) can be considered a special case of a partial match query. For example, let us consider XML documents on papers stored at the four web sites in Fig. 1. We note that the schemas in these web sites are heterogeneous. To find the names

[*] Corresponding author. Fax: +82 42 869 3510.
E-mail addresses: yhpark@mozart.kaist.ac.kr (Y.-H. Park), kywhang@mozart.kaist.ac.kr (K.-Y. Whang), bslee@cs.uvm.edu (B.S. Lee), wshan@knu.ac.kr (W.-S. Han) .

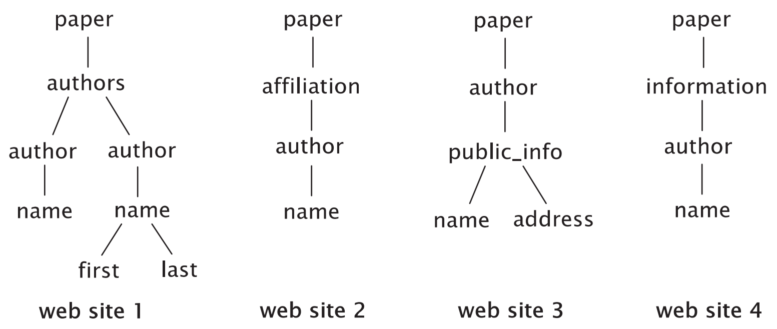


Fig. 1. XML documents for publications in four web sites.

of authors in the heterogeneous XML documents, the user should use the partial match query `//author//name`.

Partial match queries can be classified into linear path expressions (LPEs) and branching path expressions (BPEs). An LPE is defined as a path expression consisting of a sequence of labels having a parent–child relationship or an ancestor–descendent relationship between labels; a BPE is defined as a path expression having branching conditions for one or more labels in the LPE.[1] LPEs retrieve documents based on the path information only, without any predicate-based selection along the path, and is a primary query form used very popularly in XML document search. Thus, we focus on LPEs as the target query type in this paper. Our objective in this paper is to propose an efficient method to evaluate LPEs for partial match queries on large-scale documents of heterogeneous schemas. Note that full match queries can be regarded as a special form of partial match queries.

Existing methods for providing partial match queries can be classified into two types: schema-level methods and instance-level methods. Examples of schema-level methods are XRel (Yoshikawa et al., 2001), XParent (Jiang et al., 2002a,b), and Index Fabric (Cooper et al., 2001). Those of instance-level methods are Element Numbering Scheme (Li and Moon, 2001), Multi-Predicate Merge Join (Zhang et al., 2001), Structural Join (Al-Khalifa et al., 2002), Holistic Twig Join (Bruno et al., 2002) and its variants (Jiang et al., 2003a,b), XQuery/IR (Bremer and Gertz, 2002), Keyword Search on XML-QL (Florescu et al., 2000), XRANK (Guo et al., 2003), and Mixed Mode (Halverson et al., 2003). Among these methods, the ones of the first type are usable for partial match queries, but they are not designed for use in large-scale documents of heterogeneous schemas (Yoshikawa et al., 2001; Jiang et al., 2002a,b) or have only limited support for partial match queries (Cooper et al., 2001). The ones of the second type can support partial match queries, but can not be best used in a large-scale database because of inefficiency. Between these two classes of methods, the schema-level methods are much more feasible than the instance-level methods for large-scale XML documents because of their abilities to "filter out" document instances at the schema-level. We thus adopt the schema-level methods as the basis of our method. More details on this will appear in Section 3.

We particularly base our method on the schema-level methods using relational tables, such as XRel (Yoshikawa et al., 2001) and XParent (Jiang et al., 2002a,b). There are two reasons for this. First, those methods can utilize well-established techniques on relational DBMSs instead of a few native XML storages. Second, those methods can also utilize SQLs to query XML documents. For the query processing, they store the schema information and instance information of XML documents in relational tables, and process partial match queries in two phases: first, find the XML documents whose schemas match a query's path expression, and second, among the documents, find those that belong to the path expression.

However, query processing efficiencies of the two existing methods, XRel and XParent, are too limited for large-scale applications, as we will show in our experiments in Section 6. The main hurdle in existing methods is the large amount of schema information. The goal of our method (we name it XIR-Linear) is to improve the efficiency in such an environment. Specifically, we present a method that adopts the inverted index (Salton and McGill, 1983) technique, used traditionally in the information retrieval (IR) field, for searching a very large amount of schema information. IR techniques have been successfully used for searching large-scale documents with only a few keywords (constituting partial schema information). If we treat the schema of an XML document as a text document and convert partial match queries to keyword-based text search queries, we can effectively search against heterogeneous XML documents using partial match queries.

In this paper, we first describe the relational table structures for storing the XML document schema and instance information, and then, describe the structure

[1] We define LPE more formally in Section 2.2.

of the inverted index. We then present the algorithms for processing queries. For this purpose, we present the rules for mapping an LPE to a search expression on the inverted index and present an algorithm for finding the nodes matching the LPE. Then, we discuss the performance of XIR-Linear in comparison with those of XRel and XParent, and verify our comparison through experiments using real XML document sets collected by crawlers from the Internet. The results show that XIR-Linear outperforms both XRel and XParent by several orders of magnitude for LPEs.

The rest of the paper is organized as follows. Section 2 introduces the XML document model and query model supported in our XIR-Linear method. Section 3 discusses existing XPath query processing methods. Section 4 presents the storage structures used in XIR-Linear, and Section 5 presents the query processing algorithm based on these structures. Section 6 presents the performance evaluation of XIR-Linear compared with XRel and XParent. Section 7 concludes the paper.

2. Preliminaries

2.1. XML document model

Our XML document model is based on the one proposed by Bruno et al. (Bruno et al., 2002). In this model, an XML document is represented as a rooted, ordered, labeled tree. A node in the tree represents an element, an attribute, or a value; an edge in the tree represents an element–subelement relationship, element–attribute relationship, element–value relationship, or attribute–value relationship. Element and attribute nodes collectively define the document structure, and we assign labels (i.e., names) and unique identifiers to them.

Fig. 2 shows an example XML tree of a document. In this figure, all leaf nodes except those numbered 15 and 27 (representing the two attribute values "*R*" and "*T*") represent values, and all non-leaf nodes except those numbered 14 and 26 (representing the attribute `@category`) represent elements. Note that attributes are distinguished from elements using a prefix '@' in the labels.

In principal, an element can have a single attribute of type *ID* whose value is a unique identifier that can be referenced by attributes of type *IDREF* or *IDREFS* of other elements, constructing a graph structure. However, our work does not take into account for the cyclic references, assuming a tree model as in existing methods (Li and Moon, 2001; Jiang et al., 2003a,b). We leave the cyclic reference problem in query processing as a future work because it can be considered as a separate important research issue.

We modify this model so that a node represents either an element (element node) or an attribute (attribute node) but not a value. We also modify the model with the notion of label paths as defined in Definition 1.

Definition 1. A label path in an XML tree is defined as a sequence of node labels $l_1, l_2, \ldots, l_p$ ($p \geqslant 1$) from the root to a node p in the tree, and is denoted as $l_1.l_2.\ldots.l_p$.

We say a label path *matches* a partial match query. For example, in Fig. 2, `issue.editor.first` is a label path matching a path expression `//editor//first`. Note that there may be multiple nodes belonging to the same label path.

There are other definitions of the label path, such as those in DataGuides (Goldman and Widom, 1997) or XParent (Jiang et al., 2002a). DataGuides defines the path as a sequence of nodes from an internal node, which is not necessarily the root, to a leaf node; XParent defines the path as a sequence of *edges* instead of nodes.

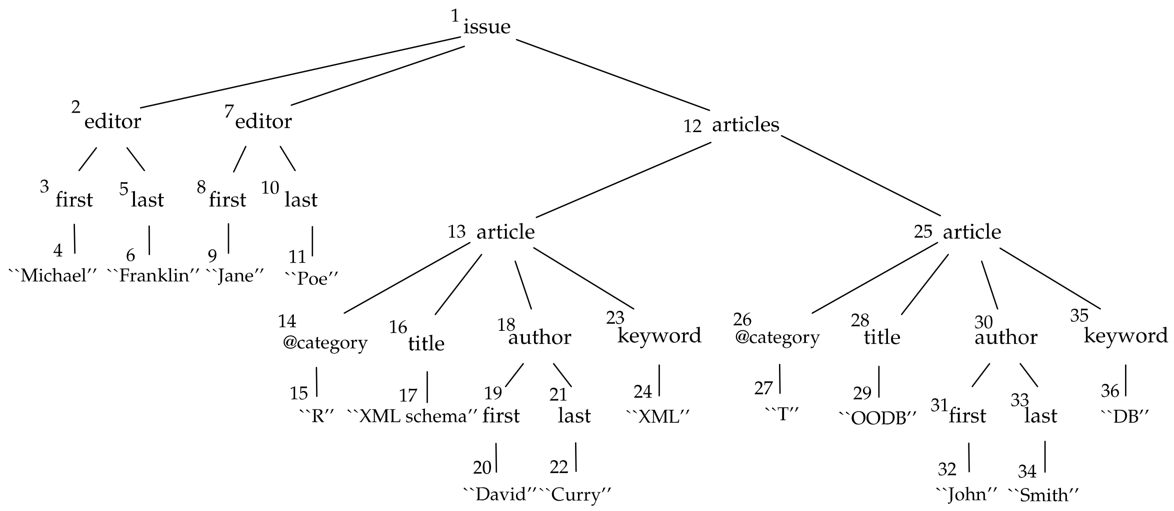


Fig. 2. An example XML tree of a document.

2.2. XML query model

Our query language belongs to the tree pattern query (TPQ) class (Amer-Yahia et al., 2001; Ramanan, 2003). The query language supports two kinds of path expressions: (1) linear path expressions (LPEs) and (2) branching path expressions (BPEs). Among them, we define the LPE, the query type of our focus in this paper, as a sequence of labels connected with '/' or '//' as in Definition 2.

Definition 2. A linear path expression (LPE) is defined as $l_0o_1l_1o_2l_2\cdots o_nl_n$, where l_i $(i = 0,1,\ldots,n)$ is the i-th label in the path, and o_j $(j = 1,2,\ldots,n)$ is either '/' or '//' which, respectively, denotes a parent–child relationship or an ancestor–descendant relationship between l_{j-1} and l_j. Here, l_0 is the root of the XML tree denoting the set of all XML documents and may be omitted.

For example, the LPE `/issue//article/title` is for retrieving all `title` elements that are children of the `article` elements that are descendants of the `issue` elements.

3. Related Work

As mentioned in Introduction, there are two kinds of methods for evaluating path expressions: schema-level methods and instance-level methods. A schema-level method uses structural information like the label paths to find nodes matching a path expression (Jiang et al., 2002a,b; Yoshikawa et al., 2001; Cooper et al., 2001), whereas an instance-level method uses only node identification information like the start and end positions of a node (Al-Khalifa et al., 2002; Bruno et al., 2002; Jiang et al., 2003a,b). In this section, we briefly discuss instance-level methods, and then, focus on schema-level methods.

3.1. Instance-level methods

There have been three different approaches for the instance-level method. The first uses XML tree navigation (Altinel and and Franklin, 2000; Ives et al., 2000; McHugh and Widom, 1999). It converts a path expression to a "state machine",[2] and then evaluates the path expression by navigating the XML tree guided by the state machine. The second uses node instance information stored for each node in an XML tree (Al-Khalifa et al., 2002; Bruno et al., 2002; Jiang et al., 2003a,b; Li and Moon, 2001; Tatarinov et al., 2002; Zhang et al., 2001). It converts a path expression to a (structural) join query, and then evaluates the join query using the node instance information. The third uses information retrieval (IR) technique, particularly an inverted index created on XML documents (Bremer and Gertz, 2002; Florescu et al., 2000; Guo et al., 2003; Halverson et al., 2003). In the remainder of this subsection, we further discuss the second and the third approaches as they are far more efficient than the first one (which requires navigating the XML tree).

[2] A representation of the sequence of labels in the path expression as a sequence of states in finite state automata.

Existing methods of the second approach include Multi-Predicate Merge Join (Zhang et al., 2001), Structural Join (Al-Khalifa et al., 2002), and Holistic Twig Join (Bruno et al., 2002) and its variants (Jiang et al., 2003a,b). These methods have the advantage that partial match queries can be processed with only instance-level information. However, query evaluation involves comparing the node instance information, and therefore, tends to be more expensive than in the schema-level methods, which can filter out node instances significantly by using the schema information.

Existing methods of the third approach include XQuery/IR (Bremer and Gertz, 2002), Keyword Search on XML-QL (Florescu et al., 2000), XRANK (Guo et al., 2003), and Mixed Mode (Halverson et al., 2003). Inverted indexes are created on instance-level information, i.e., on the values in the XML document in XQuery/IR (Bremer and Gertz, 2002) and XRANK (Guo et al., 2003) and on the nodes (i.e., elements, attributes) as well as values in Keyword Search on XML-QL (Florescu et al., 2000) and Mixed Mode (Halverson et al., 2003). We note that, although using inverted indexes, they are fundamentally different from XIR, which creates an inverted index on the label paths, which are schema-level information.

3.2. Schema-level methods

Schema-level methods are categorized into those using special purpose indexes (Cooper et al., 2001; Goldman and Widom, 1997) and those using relational tables (Yoshikawa et al., 2001; Jiang et al., 2002a,b) depending on where and how label paths are stored. In the former case, label paths are stored dynamically as they are used in the queries. In the latter case, all label paths in the documents are stored in the tables of a relational DBMS a priori.

Index Fabric (Cooper et al., 2001) is considered the representative method in the schema-level methods using special purpose indexes. Index Fabric uses the Patricia trie to index the label paths and values that have occurred in the queries occurring frequently. However, Index Fabric is not meant to support partial match queries, which are very effective for searching in a heterogeneous environment. This is a critical drawback that render the method inapplicable in a large-scale, heterogeneous environment. Thus, in this section, we primarily focus on the schema-level methods using relational tables.

XRel (Yoshikawa et al., 2001) and XParent (Jiang et al., 2002a,b), which are the two representative ones among the schema-level methods using relational tables, provide a basis for our XIR method. We describe each method in this subsection. We use the term node interchangeably with element or attribute as these are represented as nodes in the XML document model and the query pattern.

3.2.1. XRel

In XRel, the XML tree structure information is stored in the following four tables:

> **Path** (label_path_id, label_path)
> **Element** (document_id, label_path_id, start_position, end_position, sibling_order)
> **Text** (document_id, label_path_id, start_position, end_position, value)
> **Attribute** (document_id, label_path_id, start_position, end_position, value)

The table Path stores all label paths in the XML documents and their identifiers. The table Element stores information about the element nodes, where the information about each node consists of the identifier of the document containing the node, the identifier of the label path ending at the node, the offsets of the start and end positions of the node within the document, and the order of the node among its siblings. The combination of start position and end position is used to identify a node in place of a node identifier. The table Text stores information about the values of element nodes, where the column value stores the text value. The table Attribute is identical to the table Text except that the column value stores the values of attributes value instead of elements. These two tables can be stored physically as one table.

In order to evaluate LPEs, XRel needs only the tables Path and Element. It first finds the label paths matching the query's path expression from the Path table. The matching is done using the SQL string match operator LIKE. All label paths in the Path table must be scanned in this case because an index like the B+-tree cannot be used to search for a partially matching label path. Then, XRel joins the set of matching label paths with the table Element via the column label_path_id to obtain the result nodes. Fig. 3 shows the SQL statement generated for evaluating the LPE //article//author/first. In the LIKE phrase, the symbol '#' is a label delimiter, and the symbol '%' is a wildcard matching one or more characters. The table Text and Attribute can be subsequently used to retrieve the values of the selected nodes if needed.

3.2.2. XParent

XParent (Jiang et al., 2002a,b) is similar to XRel, but uses a slightly different table schema for a different node identification mechanism, i.e., the node identifier (*node_id*) instead of the interval (*start_position, end_position*).

```
SELECT distinct e1.document_id, e1.start_position, e1.end_position
FROM   Path p1, Element e1
WHERE  p1.label_path LIKE ``#%/article#%/ author#/first#"
AND    e1.label_path_id = p1.label_path_id;
```

Fig. 3. XRel SQL statement for the partial match query //article//author/first.

```
SELECT distinct e1.document_id, e1.node_id
FROM   LabelPath lp1, Element e1
WHERE  e1.label_path_id = lp1.label_path_id
AND    lp1.label_path LIKE ``.%/article.%/author./first.";
```

Fig. 4. XParent SQL statement for the partial match query //article//author/first.

> **LabelPath** (label_path_id, length, label_path)
> **Element** (document_id, label_path_id, node_id, sibling_order)
> **Data** (document_id, label_path_id, node_id, sibling_order, value)
> **DataPath** (parent_node_id, child_node_id)
> **Ancestor** (node_id, ancestor_node_id, offset_to_ancestor)

The table LabelPath is equivalent to the table Path in XRel. The table Element stores information about the element nodes and attribute nodes and, thus, is equivalent to the union of the table Element and the table Attribute (without the column value) in XRel. The table Data stores information about element values and, thus, is equivalent to the table Text in XRel. The table DataPath (a.k.a. Parent table) keeps parent–child relationship between nodes. Alternatively, the table Ancestor may be used to keep ancestor–descendent relationship between nodes.

In query processing, XParent evaluates an LPE in the same way as XRel, using the tables Labelpath and Element. Fig. 4 shows the SQL statement for the LPE //article//author/first. The SQL statement is very similar to that in XRel except using node_id instead of start_position and end_position (and using '.' instead of '#' in the LIKE phrase). The table Data can be subsequently used to retrieve the values of the selected nodes if needed.[3]

4. XIR-Linear storage structures

XIR-Linear uses three tables and an inverted index to store information about XML document structure:

[3] The tables DataPath and Ancestor are used for BPE's, which are not discussed in this paper.

LabelPath (label_path_id, label_path)
Element (document_id, label_path_id, node_id, sibling_order)
Data (document_id, label_path_id, node_id, sibling_order, value)
Inverted index on label_path of the table LabelPath.

Fig. 5 shows the LabelPath table and the inverted index for the example XML tree in Fig. 2. The table LabelPath represents the *schema-level information* and stores all the distinct label paths occurring in XML documents and their path identifiers (label_path_pid). We add the labels prefixed with '$' and '&' to denote the first label and the last label of each label path. The first label is to match the root label of the document, and the last label is to match the last label in a partial match query. The details on their use in query processing will appear in Section 5.

The LabelPath inverted index is created on the label_path column of the LabelPath table. Here, we consider label paths as text documents and labels in these label paths as keywords. Like the traditional inverted index (Salton and McGill, 1983), the LabelPath inverted index is made of the pairs of a keyword (i.e., a label) and a posting list. Each posting in a posting list has the following fields: label_path_id, occurrence_count, offsets, label_path_length, where label_path_id is the identifier of the label path in which the label occurs, occurrence_count is the number of occurrences of the label within the label path, offsets is the set of the positions of the label

label_path_id	label_path
1	$issue.issue.&issue
2	$issue.issue.editor.&editor
3	$issue.issue.editor.first.&first
4	$issue.issue.editor.last.&last
5	$issue.issue.articles.&articles
6	$issue.issue.articles.article.&article
7	$issue.issue.articles.article.@category.&@category
8	$issue.issue.articles.article.title.&title
9	$issue.issue.articles.article.author.&author
10	$issue.issue.articles.article.author.first.&first
11	$issue.issue.articles.article.author.last.&last
12	$issue.issue.articles.article.keyword.&keyword

a

keyword	posting list
$issue	: <1, 1, {1}, 3> <2, 1, {1}, 4> <3, 1, {1}, 5> ...
issue	: <1, 1, {2}, 3> <2, 1, {2}, 4> <3, 1, {2}, 5> ...
&issue	: <1, 1, {3}, 3>
article	: <6, 1, {4}, 5> <7, 1, {4}, 6> <8, 1, {4}, 6> ...
&article	: <6, 1, {5}, 5>
articles	: <5, 1, {3}, 4> <6, 1, {3}, 5> <7, 1, {3}, 6> ...
&articles	: <5, 1, {4}, 4>
editor	: <2, 1, {3}, 4> <3, 1, {3}, 5> <4, 1, {3}, 5>
&editor	: <2, 1, {4}, 4>
author	: <9, 1, {5}, 6> <10, 1, {5}, 7> <11, 1, {5}, 7>
&author	: <9, 1, {6}, 6>
first	: <3, 1, {4}, 5> <10, 1, {6}, 7>
&first	: <10, 1, {7}, 7>
last	: <4, 1, {4}, 5> <11, 1, {6}, 7>
&last	: <11, 1, {7}, 7>
title	: <8, 1, {5}, 6>
&title	: <8, 1, {6}, 6>
keyword	: <12, 1, {5}, 6>
&keyword	: <12, 1, {6}, 6>
@category	: <7, 1, {5}, 6>
&@category	: <7, 1, {6}, 6>

b

Fig. 5. An example (a) LabelPath table and (b) inverted index.

document_id	label_path_id	node_id	sibling_order
1	1	1	1
1	2	2	1
1	3	3	1
1	4	5	2
1	2	7	2
1	3	8	1
1	4	10	2
1	5	12	3
1	6	13	1
1	7	14	1
1	8	16	2
1	9	18	3
...	...	...	...
1	12	35	4

a

document_id	label_path_id	node_id	sibling_order	value
1	3	4	1	Michael
1	4	6	1	Flanklin
1	3	9	1	Jane
1	4	11	1	Poe
1	7	15	1	R
1	8	17	1	XML schema
1	10	20	1	David
1	11	22	1	Curry
1	12	24	1	XML
1	7	27	1	T
1	8	29	1	OODB
1	10	32	1	John
1	11	34	1	Smith
1	12	36	1	DB

b

Fig. 6. An example (a) Element table and (b) Data table for the XML document of Fig. 2.

```
Algorithm XIR_Linear_LPE_evaluation
Input: LPE P, LabelPath inverted index, Element table
Output: set of nodes obtained from the LPE
begin
1. Convert the input LPE P to an IR expression E using the syntactic
   mapping rule LPE-to-IRExp (Rule 1).
2. Find the set of label_path_ids (pidSet) using the LabelPath inverted index
   given the IR expression E.
3. Find the set of nodes(Nset) through an equi-join between
   pidSet and the Element table.
4. Return Nset.
end
```

Fig. 7. XIR-Linear LPE evaluation algorithm.

from the beginning of the label path, and `label_path_length` is the number of labels in the label path. For instance, in the posting of the label `section` in a label path `$chapter.chapter.section.section.section.paragraph.&paragraph`, the `occurrence_count` of `section` is 3, the `offsets` of `section` is {3, 4, 5}, and the `label_path_length` is 7.

The tables `Element` and `Data` represent the instance-level information and are identical to those of XParent. They are used to identify the nodes in the XML documents that belong to the label path selected from `LabelPath` table representing the schema-level information.[4] Fig. 6 shows an example of the `Element` table and `Data` table for the XML tree in Fig. 2.

5. XIR-Linear query processing algorithms

Fig. 7 shows the algorithm for evaluating an LPE based on the XIR-Linear storage structures described in Section 4. It first finds matching label paths in the `LabelPath` table using the `LabelPath` inverted index, and then, performs an equi-join between the set of the label paths found and the `Element` table via the column `label_path_id`. It then returns the matching nodes as the query result. The table `Data` can be subsequently used to retrieve the values of the selected nodes if needed.

Formally, an LPE is evaluated as

$$\Pi_{node\_id}(\sigma_{MATCH(label\_path,LPE)} \\ LabelPath \bowtie_{label\_path\_id=label\_path\_id} Element) \qquad (1)$$

Since the selection $\sigma_{MATCH(label\_path,LPE)}LabelPath$ is implemented as a text search on the `label_path` column, XIR-Linear should first convert an LPE to a keyword-based text search condition (we call it *information retrieval expression (IRExp)*). The following rule specifies how the conversion is done.

Rule 1. [*LPE-to-IRExp*] An LPE $o_1 l_1 o_2 l_2 \cdots o_p l_p$, where $o_i \in \{`/', `//'\}$ for $i = 1,2,\ldots,p$, is mapped to an IRExp using the following rule:

$$o_1 l_1 \Rightarrow \begin{cases} l_1 & \text{if } o_1 = `//' \\ \$l_1 \text{ near}(1)\ l_1 & \text{if } o_1 = `/' \end{cases}$$

$$l_i o_{i+1} l_{i+1} \Rightarrow \begin{cases} l_i \text{ near}(\infty) l_{i+1} & \text{if } o_{i+1} = `//' \\ l_i \text{ near}(1)\ l_{i+1} & \text{if } o_{i+1} = `/' \end{cases}$$
$$\text{for } i = 1,2,\ldots,p-1$$

$$l_p \Rightarrow l_p \text{near}(1)\&l_p$$

where near(w) is the proximity operator, which retrieves the documents in which the two operand keywords appear within w words apart.

Note that l_1 and l_p are respectively the root (i.e., first) node and the leaf (i.e., last) node of the LPE. For example, an LPE `/issue/articles//author` is converted to an IRExp `&issue near(1) issue near(1) articles near(∞) author near(1) &author`. Note `$issue` indicates that `issue` is the root of the XML tree of the document.

Example 1. Consider the LPE `//article//author/first`. Using the rule LPE-to-IRExp, XIR-Linear converts this LPE to the IRExp `article near(∞) author near(1) first near(1) &first`. Then, searching the `LabelPath` inverted index in Fig. 5(b) returns the *pidSet* {10}, and joining this set with the `Element` table returns the set of nodes {19, 31}. Fig. 8 shows the SQL statement generated for evaluating the LPE. The `MATCH` in the `WHERE` clause

```
SELECT  DISTINCT e1.document_id, e1.node_id
FROM    LabelPath p1, Element e1
WHERE   p1.pid = e1.pid
AND     MATCH(p1.label_path, `article' NEAR(MAXINT) `author'
                               NEAR(1) `first' NEAR(1) `&first');
```

Fig. 8. XIR-Linear SQL statement for the LPE `//article/author/first`.

[4] These two tables of XParent correspond to three tables `Element`, `Text`, and `Attribute` of XRel. We prefer those of XParent due to the join efficiency for BPE (Jiang et al., 2002a,b) of using node identifiers (`node_id`) over using start_position and end_position in XRel.

Performance-related features	XRel	XParent	XIR-Linear
label path match method	string match	string match	inverted index search
element identification method	interval	node id	method used in XRel or XParent
factors affecting label path match time	the number of label paths in *Path* table	the number of label paths in *LabelPath* table	the sum of the lengths of the posting lists for the labels in the LPE

Fig. 9. Comparison of XRel, XParent, and XIR-Linear.

makes use of the `LabelPath` inverted index. The symbol `MAXINT` is a system-defined maximum integer.

The query processing algorithms of XRel, XParent, and XIR-Linear share the same outline, but have quite different implementations leading to their performance differences. Fig. 9 shows a summary of comparing the three methods. XIR-Linear's performance advantage over XRel and XParent comes from using inverted index search instead of string match for finding label paths matching the LPE. For XRel or XParent, the label path match time is proportional to the number of label paths stored in the `Path` or `LabelPath` table. In contrast, for XIR-Linear, the time is determined by the number of labels in the LPE and the lengths of the posting lists associated with these labels in the index. The length of the posting list for each label is proportional to the number of label paths containing the label. This number is only a small portion of the total number of label paths stored in the table `LabelPath`. Thus, the label path match time is expected to be far shorter for XIR-Linear.

6. Performance evaluation

In this section we compare the performance of XIR-Linear with those of XRel and XParent, with particular attention to the efficiency of query processing.

6.1. Experimental setup

6.1.1. Databases

We have collected 10008 XML documents from the Internet using two web crawlers: Teleport Pro Version 1.29.1959 (Teleport, 2004) and ReGet Deluxe 3.3 Beta (build 173) (ReGet, 2004). Note that we have not used a synthetic data set because these data are confined within a particular domain and, consequently, do not have sufficiently heterogeneous structures.

Using the collected XML documents, we have constructed five sets of data files of different sizes. Each set contains approximately 5000, 10,000, 20,000, 40,000, and 80,000 distinct label paths. The last set has 1,460,000 nodes. A larger set contains all label paths in a smaller set, i.e., is a superset of smaller sets. Each set has been loaded into three databases, each containing tables used by XRel, XParent, and XIR-Linear methods. The total number of databases thus generated is 15.

For XRel and XParent, we have used the database schema and indexes as they were used in the original designs (Jiang et al., 2002a,b). For XIR-Linear, we have loaded the data files into the `LabelPath`, `Element`, and `Data` tables, created B+-tree indexes on the columns `document_id` and `label_path_id` of each table as in XRel or XParent, and created an inverted index on the `label_path` column of the `Labelpath` table.

6.1.2. Queries

Table 1 shows two groups organized as sets of XPath LPE queries: one is on `issue` documents; the other is on `movie` documents. The former has far more

Table 1
Queries

Group	Label	XPath query	Selectivity
Group 1	LPE1	/issue/editor	10^{-7}–10^{-6}
(LPEs on	LPE2	//issue//first	10^{-4}–10^{-3}
`issue`	LPE3	//issue//author/first	10^{-5}–10^{-4}
documents)	LPE4	//issue//article//author/first	10^{-5}–10^{-4}
Group 2	LPE5	/movie/cast	10^{-7}–10^{-6}
(LPEs on	LPE6	//movie//first	10^{-6}–10^{-5}
`movie`	LPE7	//movie//actor//first	10^{-6}–10^{-5}
documents)	LPE8	//movie/cast//actor//first	10^{-6}–10^{-5}

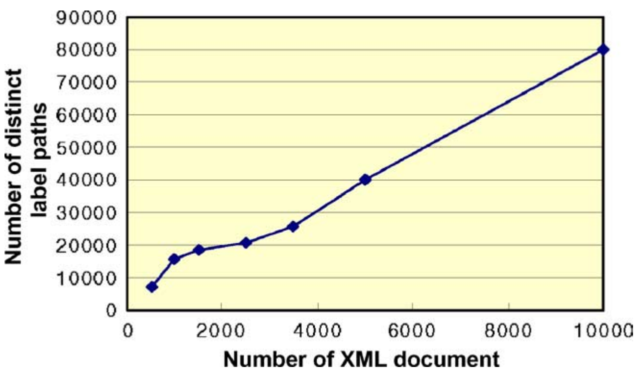
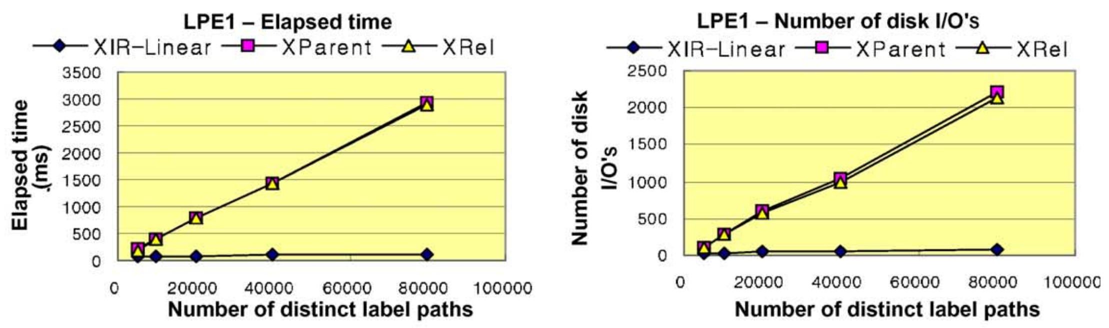
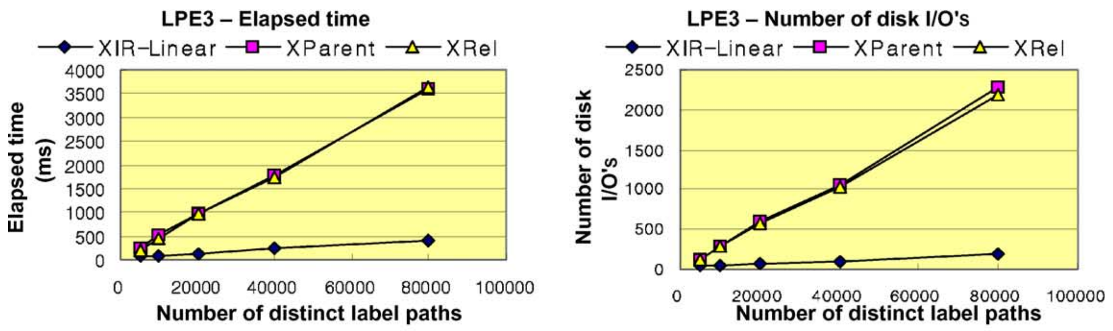


Fig. 10. The number of distinct label paths as the number of XML documents increases.

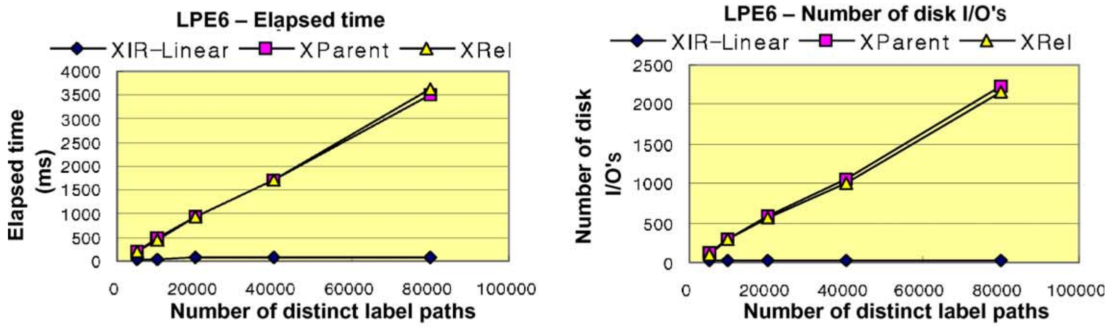
document instances than the latter. LPEs within each group have different numbers of labels and/or different combinations of '/' and '//'. The `selectivity` field in Table 1 is the ratio between the number of nodes resulting from the LPE and the total number of nodes in the database.



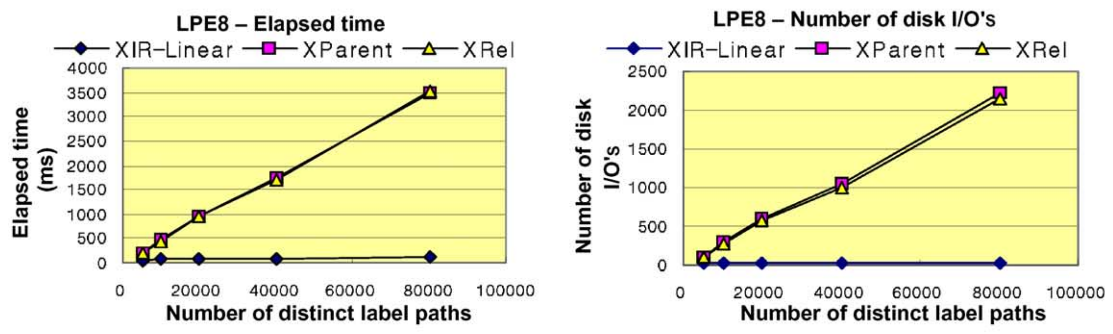
a



b



c



d

Fig. 11. Query costs of XRel, XParent, XIR-Linear (buffer size = 200 pages). (a) LPE1 (Group 1); (b) LPE3 (Group 1); (c) LPE6 (Group 2); (d) LPE8 (Group 2).

6.1.3. Computing environment

We have conducted the experiments using the Odysseus object-relational database management system[5] (Odysseus, 2004), which provides operations needed by a text search engine, on SUN Ultra 60 workstation with 512 Mbyte RAM. In order to eliminate the unpredictable buffering effect in the operating system, we have used a raw disk device to bypass the OS buffer. Additionally, we have flushed the DBMS buffer after each query execution so that the execution does not affect later ones. The cost metrics used are the elapsed time and the number of disk I/O's.

6.2. Experimental results

Since the crawlers collect *arbitrary* documents from the Internet, new label paths are added as new documents are added by crawling. We have extracted the number of distinct label paths from the XML documents collected. Fig. 10 shows the result. The database size is 301 Mbytes for XIR-Linear, 271 Mbytes for XRel, and 248 Mbytes for XParent when the number of XML documents is 10,000 (and the number of distinct label paths is 80,000). The database size for XIR-Linear is slightly larger due to inclusion of the inverted index.

Fig. 11 shows the results for some of the LPEs in Table 1. Results from the other LPEs show the same trend. We see that, for the database used, XIR-Linear is more efficient than both XRel and Xparent by a factor of 3 to 30 in elapsed time and by a factor of 3 to 74 in the number of disk I/O's. Note that the performance gap widens as the database size grows since the costs increase linearly for XRel and XParent while sublinearly—almost constant—for XIR-Linear. This confirms the significant performance advantage of XIR-Linear in a large-scale database environment.

7. Conclusions

We have proposed a novel approach, called XIR-Linear, for evaluating linear path expressions on a large number of heterogeneous XML documents. In this approach, the label paths occurring in XML documents are treated as texts, and an inverted index is created on them. This inverted index supports much faster partial match than XRel's or XParent's string match when evaluating a linear path expression.

We have presented the storage structures of XIR-Linear, including the inverted index as well as the tables storing all the label paths and nodes of the XML documents. Based on these structures, we have presented the query processing algorithm for a linear path expression. Then, we have compared the performance of XIR-Linear with those of XRel and XParent through experiments using real XML documents collected from the Internet. The results show that XIR-Linear outperforms both XRel and XParent by an order of magnitude with the performance gap widening as the database size grows.

As a future work, we are currently developing techniques for extending our work to evaluation of branching path expressions (BPEs) for more complex partial match queries.

[5] Developed at the Advanced Information Technology Research Center of KAIST.

References

Aboulnaga, A., Alameldeen, A., Naughton, J., 2001. Estimating the selectivity of XML path expressions for internet scale applications. In: Proceedings of the 27th International Conference on Very Large Data Bases (VLDB). pp. 591–600.

Al-Khalifa, S., Jagadish, H., Koudas, N., Patel, J., 2002. Structural joins: a primitive for efficient XML query pattern matching. In: Proceedings of the 18th International Conference on Data Engineering (ICDE). pp. 141–152.

Altinel, M., Franklin, M., 2000. Efficient filtering of XML documents for selective dissemination of information. In: Proceedings of the 26th International Conference on Very Large Data Bases (VLDB). pp. 53–64.

Amer-Yahia, S., Cho, S., Lakshmanan, L., Srivastava, D., 2001. Minimization of tree pattern queries. In: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data. pp. 497–508.

Bremer, J., Gertz, M., 2002. XQuery/IR: integrating XML document and data retrieval. In: Proceedings of the fifth International Workshop on the Web and Databases (WebDB 2002). pp. 1–6.

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

Clark, J., DeRose, S., 1999. XML Path Language (XPath), W3C Recommendation. Available from http://www.w3.org/TR/xpath.

Cooper, B., Sample, N., Franklin, M., Hjaltason, G., Shadmon, M., 2001. A fast index for semistructured data. In: Proceedings of the 27th International Conference on Very Large Data Bases (VLDB). pp. 341–350.

Florescu, D., Kossmann, D., Manolescu, I., 2000. Integrating keyword search into XML query processing. In: Proceedings of the 9th WWW Conference/Computer Networks. pp. 119–135.

Goldman, R., Widom, J., 1997. DataGuides: enabling query formulation and optimization in semistructured databases. In: Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB). pp. 436–445.

Guo, L., Shao, F., Botev, C., Shanmugasundaram, J., 2003. XRANK: ranked keyword search over XML documents. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. pp. 16–27.

Halverson, A., Burger, J., Galanis, L., Kini, A., Krishnamurthy, R., Rao, A., Tian, F., Viglas, S., Wang, Y., Naughton, J., DeWitt, D., 2003. Mixed mode XML query processing. In: Proceedings of the 29th International Conference on Very Large Data Bases (VLDB). pp. 225–236.

Ives, Z., Levy, A., Weld, D., 2000. Efficient evaluation of regular path expressions on streaming XML data. Technical Report UW-CSE-2000-05-02, University of Washington.

Jiang, H., Lu, H., Wang, W., Yu, J., 2002a. Path materialization revisited: an efficient storage model for XML data. In: Proceedings of the 13th Australasian Database Conference (ADC). pp. 85–94.

Jiang, H., Lu, H., Wang, W., Yu, J., 2002b. XParent: an efficient RDBMS-based XML database system. In: Proceedings of the 18th International Conference on Data Engineering (ICDE). pp. 335–336.

Jiang, H., Lu, H., Wang, W., Ooi, B., 2003a. XR-Tree: indexing XML data for efficient structural joins. In: Proceedings of the 19th International Conference on Data Engineering (ICDE). pp. 253–264.

Jiang, H., Wang, W., Lu, H., Yu, J., 2003b. Holistic twig joins on indexed XML documents. In: Proceedings of the 29th International Conference on Very Large Data Bases (VLDB). pp. 273–284.

Li, Q., Moon, B., 2001. Indexing and querying XML data for regular path expressions. In: Proceedings of the 27th International Conference on Very Large Data Bases (VLDB). pp. 361–370.

Mandreoli, F., Martoglia, R., Tiberio, P., 2002. Searching similar (sub)sentences for example-based machine translation. In: Proceedings of SEBD'02.

McHugh, J., Widom, J., 1999. Query optimization for XML. In: Proceedings of the 25th International Conference on Very Large Data Bases (VLDB). pp. 315–326.

Naughton, J. et al., 2001. The Niagara internet query system. IEEE Data Engineering Bulletin 24 (2), 27–33.

Odysseus, 2004. Odysseus Object-Relational Database Management System. Available from: http://aitrc.kaist.ac.kr/english/index.html.

Petrou, C., Hadjiefthymiades, S., Martakos, D., 1999. An XML-based, 3-tier scheme for integrating heterogeneous information sources to the WWW. In: Proceedings of the 10th International Workshop on Database and Expert Systems Applications, pp. 706–710.

Ramanan, P., 2003. Covering indexes for XML queries: bisimulation–simulation = negation. In: Proceedings of the 29th International Conference on Very Large Data Bases (VLDB). pp. 165–176.

ReGet, 2004. ReGet Deluxe 3.3 Beta (build 173). Available from: http://deluxe.reget.com/en/.

Salton, G., McGill, M., 1983. Introduction to modern information retrieval. McGraw-Hill, New York.

Tatarinov, I., Viglas, S., Beyer, K., Shanmugasundaram, J., Shekita, E., Zhang, C., 2002. Storing and querying ordered XML using a relational database system. In: Proceedings of 2002 ACM SIGMOD International Conference on Management of Data. pp. 204–215.

Teleport, 2004. Teleport Pro Version 1.29, http://www.tenmax.com/teleport/pro/home.htm.

XML, 2004. eXtensible Markup Language (XML). Available from: http://www.w3.org/XML/.

Xyleme, 2004. Xyleme. Available from: http://www.xyleme.com/en/.

Yoshikawa, M., Amagasa, T., Shimura, T., Uemura, S., 2001. XRel: a path-based approach to storage and retrieval of XML documents using relational databases. ACM Transactions on Internet Technology (TOIT) 1 (1), 110–141.

Zhang, C., Naughton, J., DeWitt, D., Luo, Q., Lohmann, G., 2001. On supporting containment queries in relational database management systems. In: Proceedings of 2001 ACM SIGMOD International Conference on Management of Data. pp. 425–436.