Computer Standards & Interfaces 49 (2017) 34–43

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Indexing techniques for processing generalized XML documents

Ghassan Z. Qadah

Computer Science & Engineering Department, American University of Sharjah, P.O. Box 26666, Sharjah, United Arab Emirates

ARTICLE INFO

Article history:
Received 26 November 2015
Received in revised form 19 June 2016
Accepted 1 July 2016
Available online 2 July 2016

Keywords:
Algorithms
Containment queries
Database system
Extensible Markup Language
Query processing
XLink

ABSTRACT

The Extensible Markup Language (XML) data model has recently gained huge popularity because of its ability to represent a wide variety of structured (relational) and semi-structured (document) data. Several query languages have been proposed for the XML model, the most-widely used one is the XQuery. An important component of an XQuery is its XPath expression which retrieves a set of XML documents to be manipulated by the associated XQuery. An XPath expression can be of several types, among which are the containment queries. Traditional research of processing containment queries has concentrated on data retrieval from independent XML documents; not much research has been directed towards interlinked XML documents. This paper reviews this area of research and shows the adequacy and correctness of one of the reviewed algorithms when applied to independent XML documents. However, the direct application of this algorithm to process queries against interlinked XML documents is shown to generate incorrect results. To remedy such a situation, two new algorithms and the associated indexing structures are developed and shown to perform correctly in processing both independent and/or inter-linked XML documents. In addition, one of the new algorithms is shown to minimize the storage requirement of the intermediate lists generated throughout its execution and therefore improving further the algorithm's space and time performance.

1. Introduction

A database is a huge repository of interrelated data elements [13]. Traditional databases are structured using the relational model [9]. In this model, data is organized into relations viewed as two-dimensional tables. The column headers of a table represent the attributes of the relation, whereas, the table's rows are the relation's tuples. Relational databases are accessed using a very high-level language called Structured Query Language (SQL) [13]. A statement of this type, referred to as a query, specifies the data to be retrieved from the database. To process a query, a software system, the database management system (DBMS), generates a number of execution plans, selects the one with the least cost, and then executes it to retrieve the requested data. In a relational database, a given query plan consists of a sequence of operations called the relational algebra operations (select, project, join, etc.). The execution of these operations; especially projection, join and transitive closure; against very large collection of data is generally slow. Many serial and parallel algorithms have been proposed to speed-up the execution of these operations. Some of these algorithms and their performance evaluation can be found in [1,12,18–20,22–23]. It is worth to note here that the relational model has many advantages including its simplicity, rigorous mathematical foundation, and easiness to program and use. However, it has also many limitations, including its poor modeling capability, especially for semi-structured and unstructured data.

To overcome the relational model drawbacks, new and powerful data models, with semantics richer than those of the relational one, have been introduced. Among those models are deductive [14], object-oriented [3] and XML [5]. The XML (Extensible Markup Language) model, organizes data, into collections of nested elements and character sets (strings), referred to as documents. The XML data model has gained huge popularity because of its ability to model a wide variety of structured (relational) and semi-structured (documents) data as well as its use in integrating heterogeneous data sources (traditional relational databases, data files, email messages, web pages, etc.) for displaying data on a variety of devices, including personal computers, personal digital assistants (PDAs) and smart mobile phones [16,17].

Several Query languages have been proposed for the XML data model [4,6]; the most-widely used one is XQuery [4]. An important component of the XQuery is the XPath expression which defines the set of XML documents to be manipulated by the associated XQuery. An XPath expression can be of several types, among which are the containment [24] ones. A containment query retrieves the XML elements based on the containment of these elements within some other ones. Two types of containment queries do exist, namely direct and indirect containments, both of which will be presented in details in Section 2.2.

E-mail address: gqadah@aus.edu.

http://dx.doi.org/10.1016/j.csi.2016.07.002
0920-5489/

Traditional containment query processing research [2,7,21,24] has concentrated on developing data retrievals algorithms from independent XML documents. In this paper, we review this area of research and show the adequacy and correctness of one of the reviewed algorithms when applied to independent XML documents. However, the direct application of this algorithm to process queries against interlinked XML documents is shown to generate incorrect results. In this paper, two new algorithms and the associated index structures are introduced and shown to perform correctly in processing both independent and/or interlinked XML documents. In addition, one of the new algorithms is shown to minimize the storage requirement of the intermediate lists generated throughout the execution of such an algorithm and therefore improving further the it's space and time performance.

The rest of this paper is organized as follows. Section 2 presents some background material concerning the XML database model and its query types, whereas; Section 3 reviews the processing of the containment queries against independent XML documents. Section 4 presents two new algorithms that handle the processing of XML independent and inter-linked documents. Finally, Section 5 presents a summary, some concluding remarks and future work.

2. Background

2.1. The Extensible Markup Language (XML) data model

The relational data model [9] is simple and easy to use; however, its ability to model data with complex structures is very limited. To overcome this limitation, the Extensible Markup Language (XML) model [5] has been introduced. This model, as shown in Fig. 1.a, organizes data into collections of nested elements and character sets (strings); these collections are referred to as documents. For example, Fig. 1.a presents a simple XML document that contains data about a specific car. "<car>, </car>", "<vin>, </vin>" and "<owners> </owners>" of Fig. 1.a are examples of the XML elements, whereas, "John", "Debra" and "999" are examples of the character sets/strings. An element within an XML document can be referred to by its tag; the element "<car>, </car>", for example, is referred to by car. Moreover, an XML element may contain other elements and/or strings. Element car, for example, contains, as shown in Fig. 1.a, one **owners** element which in turns contains two owner elements. Elements in the XML model may also be nested to any depth. For example, the "car" element, as shown in Fig. 1.a, contains two owner elements, each owner element contains its name and ssn.

An XML document, as shown in Fig. 1.b, can also be viewed as a rooted, ordered tree. The nodes in this tree represent the elements and strings of the XML document. The root node of the tree, node car of Fig. 1.b, represents the outermost element of the XML document, namely, the car element of Fig. 1.a. The leaf nodes of the tree, on the other hand, represent the document's string data. The other elements of the XML document, as shown in Fig. 1.a, are represented by the internal (non-leaf) nodes of the tree. A node in the tree is labeled by the tag of the element or the string it represents. An edge from one node in the tree to one of its child nodes represents a direct element-subelement (parent-child or direct-containment) relationship between the two nodes. On the other hand, a path between two nodes in the tree, going from top to bottom, represents an ancestor-descendant (indirect containment) relationship between these two nodes. For example and as shown in Fig. 1.b, the node/element **owners** directly contains two owner elements, whereas, it indirectly contains two name and two ssn elements.

It is important to note here that the fact that the XML model has a general tree structure explains its power in representing complex data structures as opposed to the limited representation power of the relational model. However, this improvement comes with a price, basically, an increased complexity when processing the XML data.

2.2. The XML database and queries

In general, an XML database is a huge collection (forest) of trees; each tree is similar to the one presented in Fig. 1.b. To query such a database, several languages have been proposed, namely, Quilt [6], XML-QL [11] and XQuery [4]. XQuery, being developed by a W3C working group, has emerged as the front-runner. An XQuery contains two types of components, namely, the XPath expressions [8], which locate elements in the XML database, and the XML functions that are used to manipulate the located elements. It is interesting to note here that the XQuery standard has over 800 built-in functions and provides a mechanism to build up new XQuery functions. A simple example of an XQuery, which output the number of **owners** elements in the car XML database, is presented next.

```
xquery version "1.0"
let $owners = doc("carsdb.xml")/car/owners
return count($owners)
```

where "carsdb.xml" is the XML database containing all of the car documents. On the other hand, doc("carsdb.xml"), calls an XQuery built-in function named doc and passes to it the name of the XML database to connect to and open. Whereas, the expression "$owners = doc ("cars.xml")/car/owners" is an XPath expression that locates all owners elements that are the children of car (see Fig. 1) and store these elements in the XQuery variable $owners. Finally, the expression "return count

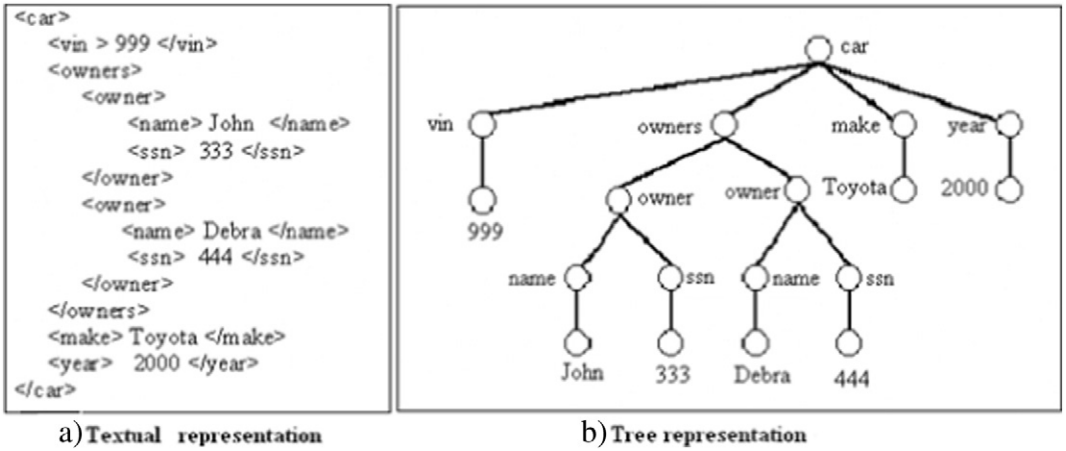

Fig. 1. Different representations of an XML document.