Egyptian Informatics Journal (2012) **13**, 209–216

Cairo University

Egyptian Informatics Journal

www.elsevier.com/locate/eij
www.sciencedirect.com

ORIGNAL ARTICLE

Efficient evaluation of reachability query for directed acyclic XML graph based on a prime number labelling schema

Awny Sayed [a,*], **Mohammed Kayed** [b], **Mayyada Hammoshi** [c]

[a] *Faculty of Sciences, Minia University, Egypt*
[b] *Faculty of Sciences, Beni-Suef University, Egypt*
[c] *College of Applied Sciences, Ibri, Oman*

Received 29 June 2011; revised 17 September 2012; accepted 8 October 2012
Available online 9 November 2012

KEYWORDS

XML;
Labelling schema;
Reachability queries;
Directed cycles graph

Abstract Many schema labelling approaches have been designed to facilitate querying of XML documents. The proposed algorithms are based on the fact that ancestor–descendant relationships among nodes can be quickly determined. Schema labelling is a family of technologies widely used in indexing tree, graph, or structured XML graph, in which a unique identifier is assigned to each node in the tree/graph. The generated identifier is then used in indexing as a reference to the actual node so that structural relationship among the nodes can be quickly captured. In this paper, we extend the prime number schema labelling algorithm for labelling DAG XML graph. Our main contribution is scaling down the original XML graph size substantially based on the Strongly Connected Component (SCC) principles. Labelling each node in DAG with an integer that is the arithmetical multiplication of the prime number associating with the node and its parent label. The schema does not depend on spanning tree. Thus, subsumption hierarchies represented in a DAG can be efficiently explored by checking the divisibility among the labels. Also, it inherits dynamic update ability and compact size features from its predecessors. Our theoretical analysis and the experimental results showed that the generated labelled schema is an efficient and a scalable one for processing reachability queries on large XML graphs.

1. Introduction

Recent interests on XML, biological databases, social network analysis, the Semantic Web, Web ontology, and many other emerged applications have sparked renewing the interests of graph-structured databases (or simply *graphs*) and the related problems (e.g., query processing and optimization [1]). Reachability queries have many emerged applications in graph structured databases. For example, ancestor–descendant (“//”) relationship in a large XML graph can be implemented as

* Corresponding author. Mobile: +20 968 98838296.
E-mail address: awny.ibr@cas.edu.om (A. Sayed).

Peer review under responsibility of Faculty of Computers and Information, Cairo University.

ELSEVIER Production and hosting by Elsevier

1110-8665 © 2012 Faculty of Computers and Information, Cairo University. Production and hosting by Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.eij.2012.10.002

reachability queries problem. There are two alternative approaches for evaluating reachability queries. The running time of the first approach which uses the principle of graph traversal is O($|G|$), where $|G|$ is the size of the graph G. In the second approach, the reachability queries can be evaluated by precompiling the Transitive Closure (TC) of the underlying graph. The running time is quadratic to the graph size in the worst case. It is clear that these approaches are not qualified if the underlying graph is a large one with many cycles, like the XML graph. In this paper, we focus on minimizing the underlying XML graph of many cycles by taking links (ID/IDREF, XLink, and XPointer) into account during the parsing process. So, a Directed Acyclic Graph (DAG) is generated as a result of this minimization. The principle of Strongly Connected Component (SCC) is used. Then, we assign a label to each vertex in DAG. This label is based on the Prime Number characteristics. The proposed labels will be used to efficiently determine the ancestor–descendant relationship ''or reachability queries'', ''ancestor's or self'', ''descendant-or self'', ''sibling'' path expressions.

Moreover, one major category of schema labelling of DAG is a spanning tree. *First*, we find the spanning tree and label each node in the tree according to tree's edges. *Second*, additional labels are proposed to record the relationships represented through non-tree edges. Many labelling methods are presented to solve this problem such as interval-based [2] and prefix-based [3] labelling. Labels generated by such methods still have some disadvantages. First, the evaluation of the relationship implied by non-tree edges cannot take the advantage of deterministic tree labels characteristics. Second, non-tree edges need additional techniques for query processing. Also, intervals-based approaches suffer from the updates problem. Two-Hop [4] and bit-vector [5] have no concern about the spanning tree.

Recently, two more approaches are proposed based on a prime numbering [5] and [6]. Each tree node in the two approaches is given a unique prime number, and the node is assigned by a label which is the multiplication of the node parent's label and the node prime number. These approaches still suffer from the updating problem. To the best of our knowledge, no further works are proposed to extend the idea of prime numbering to work with DAG XML graph.

The main components (parts) of our labelling approach can be summarized as follows:

- Break down the underlying large xml graph which has many cycles into DAG with a small number of vertexes and edges.
- Assign each vertex in DAG a prime number with two parts; the first part contains self-label and the second part contains label of the vertex parent.
- Create two database tables; the first table is used to store information about SCCs, while the second one is used to store information about the labels.
- Build a B-tree index for each table to speed up query evaluation.
- Compare the proposed schema with PLSD-FULL schema [7] based on the efficiency and the storage requirements.

The rest of the paper is organized as follows. Section 2 reviews related works. The SCCs principles and the proposed prime number labelling schema of DAG are discussed in Section 3. Size estimation and optimization techniques are given in Section 4. Section 5 illustrates the experimental results. Finally, we conclude and give future work in Section 6.

2. Related work

Numbering and labelling schemes are essentially used to avoid exhaustive traversing of documents for query processing. Many of the proposed labels focus on the simplified case of tree-like (rather than graph-like) data and simple path expressions (like, movie/director/name, where "/" is the parent–child axis). Queries contain branches have to be resolved into multiple sub-queries; each one is corresponding to a single branch in the original structure. The result of these sub-queries has then to be combined by expensive join operations to produce the final answer. These approaches are inefficient in handling ancestors-descendants queries with wildcard ("//") over arbitrary XML graphs with long paths.

2.1. Tree-centric nature

We discuss the proposals that facilitate numbering and labelling schemes for the efficient evaluation of ancestors-descendants queries. XML schema labelling has been proposed in [4,8,9]. By comparing labels assigned to the nodes of the XML tree, it is possible to determine the relationship between any two nodes. The authors in [10,11] proposed dynamic labelling schemes, where nodes inherit their parent labels as prefixes to their own labels. The reachability queries can simply be determined by examining whether the prefix relationship exists in the labels of the two nodes. In [12], a nested loop-join algorithm has been proposed which requires B+ tree indexes on the input element sets. The proposal in [13] also relies on B+ trees, however, additionally it requires that element sets are sorted. Then, those elements that do not participate in the join are eliminated. The authors in [10,14] described a labelling scheme for XML trees that supports efficient evaluation of ancestor queries as well as efficient insertion of new nodes. In [15,16], a tree labelling scheme based on a two level partition of the tree has been presented.

2.2. Prefix schema

The label of a node is a concatenation of its parents label and its own label. For any two given nodes u and v; u is ancestor of v *iff label* (u) is a prefix of *label* (v). There are two prefix schemes: an integer based [10] and a binary string based [11]. *Prime Number schema*, Wu et al. [5] proposed a novel approach to label XML trees with prime numbers. The label of the node is the product of its parent label and its own label. For any two nodes u and v; u is an ancestor of v *iff label* (u)% label (v) = 0; where % is the modulus operator.

2.3. Graph-centric nature

In the meantime, several approaches have been proposed to deal with graph-like XML-documents. For example, APEX index [17] uses data mining algorithms to summarize paths that appear frequently in the query workload. Instead of keeping all paths starting from the root node, it maintains only paths of length two. Therefore, it performs poorly for ancestor–descendant and descendant-or-self queries (e.g. ''director//

title''). Many other approaches rely on a structural summary along with a stored mapping from these summary nodes to the data nodes. Such an index is used to evaluate path expressions directly by pruning the search space. Data Guide [18] is restricted to a simple label path and is not useful in complex path queries with several regular expressions. Index Fabric [19] is conceptually similar to the Data Guide in that it keeps all label paths starting from the root element. It encodes each label path to each XML element with a data value as a string and inserts the encoded label path and data value into an efficient index for strings. The index block and XML data are both stored in a relational database systems. Index Fabric losses all parent–child relationships, thus, is not efficient for processing partial matching queries [19]. Similar to Index Fabric, the F + B Index [20] optimizes a set of branching queries. It is based on the Forward and Backward index (F&B index [21]) and suffers from the same problems as index Fabric. Index schemes like 1-index [22], $A(k)$-index [23], and $D(k)$-index [24] are based on the concepts of similarity and bi-similarity of nodes. These indexes are suited for intra-document links (of type ID and IDREF(S)). But they ignore inter-document links (links by XLink [25] and XPointer [26]). Moreover, much indexing techniques has been proposed for optimizing reachability queries on tree [5,27,28], DAG graph [7,29–32] and arbitrary graph [4,14,33,34]. The performance of such indexes is improved on the graphs with certain structural characteristics.

3. Labelling directed acyclic XML graph

Before discussing the notations and our schema labelling method in Section 3.2, we show how to compress a large XML graph into a small DAG using the idea of strongly connected components in Section 3.1.

3.1. Strongly Connected Components (SCCs)

The problem of determining strongly connected components of an input graph is a ''classical'' one. A common solution is based on two *depth-first traversing algorithm* [35]. To detect the SCCs, the underlying graph is traversed twice. In the first traversing, a depth-first search, all the edges are visited to construct a *depth first* spanning forest. Once the root (topmost node) of the strongly connected component is found, all its descendants that are not elements of previously found components are marked as elements of these components. The second traversing is implemented by a ''stack'' that holds each node in a depth first order (computed in the first step). Before the root of a component is pushed from the stack, all nodes down the root are removed from the stack. These nodes form the component in question.

Fig. 1a describes the algorithm that is used to compute the strongly connected component of the input graph $G = (V, E)$. It consists of a recursive procedure *visit* and the main procedure (SCC) that traverses the graph in a depth-first fashion and calls the method visit. The first node which is identified as a component of the SCC is called the root of the SCC. The *MIN operation* in the visit procedure (line 5) compares the nodes using the order in which *visit( )* has entered them. The *Comp operation* at line 10 is used to distinguish among nodes belonging to the same component as well as nodes belonging to other components. The time complexity for the procedure SCC is $O(V + E)$, where V is the set of nodes in the underlying graph and E is the set of edges. The size of the Strongly Connected Component is defined as the number of nodes it contains. Any node of the underlying graph that is not contained in cycles forms an SCC of size one. From Fig. 1b it is clear that (title, movie, director) will construct one SCC.

3.2. Notations and labelling schema for DAG

A directed graph G can be represented as $G = (V, E)$, where V represents the number of the nodes and E represents the number of edges. For any two pair of nodes u and u' in G, the sequence of nodes $\langle v_0, v_1, \ldots, v_k \rangle$ is called a path, if $u = v_0$, $u' = v_k$, and $(v_{i-1}, v_i) \in$ E for $i = 1, 2, \ldots, k + 1$. The underlying graph is a directed acyclic graph (DAG) if there is no path which starts at node u and ends at the same node. Fig. 1 shows an XML graph representation from a Movie database.

```
Procedure visit (v)
 1. Root (v) = v; Comp (v) = Null
 2. push (v, stack)
 3. for each (node u such that (u, v) ∈ E )
 4.    if u is not visited then visit (u);
 5.    if Comp (u) = Null then Root (v) = MIN (Root (v), Root (u));
 6.    if Root (v) = v then
 7.       Create a new component SCC
 8.       repeat
 9.          u = POP (stack)
 10.         Comp (u) = SCC
 11.         Insert u into Component SCC
 12.      until u = v
 13.   endif
 14. endfor

Procedure SCC (V)  // main program
 1. stack = φ
 2. for each node v ∈V
 3.    if v is not already visited then visit (v) endif
 4. endfor
```

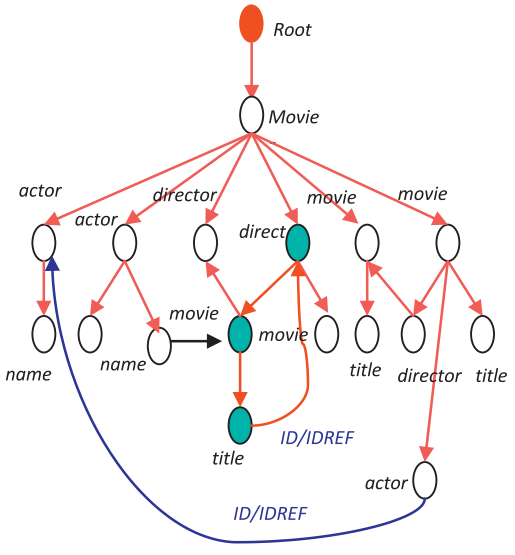

Figure 1 (a) SCC algorithm and (b) movies database sample (MDB).

Definition 1 (*SCC*). Let $G = (V, E)$ is a directed graph (digraph), a strongly connected component of G is a maximal set of vertices $C \subseteq V$ such that for all $u, v \in C$, both $u \rightsquigarrow v$ and $v \rightsquigarrow u$; that is both u and v are reachable from each other. In other words, two vertices of directed graph are in the same component if and only if they are reachable from each other.

Definition 2 (*Divisible property*). If an integer A has a prime factor which is not a prime factor of another integer B, then B is not divisible by A. This property can be applied on graph nodes, such that, for each two nodes u and v in DAG with labels $L(u)$ and $L(v)$, the node u is *ancestor* of the node v if and only if $L(u)$ *is dividable by* $L(v)$, otherwise, u is not an ancestor of v or there is no reachability.

Definition 3. Prime Number Labelling Schema for DAGGiven a DAG $G = (V, E)$, a prime number labelling schema for DAG is to assign to each vertex $v \in V$ a prime number P and associate a label $L(v)$ to the vertex v as: $L(v) = P(v)$, where $P(v)$ is the product of the prime numbers of all ancestors.

Definition 4 (*Graph Decomposition*). A graph G is decomposed into graphs in H if the set of edges E(G) can be partitioned into subsets such that each subset induces a graph in H. For simplicity, we say that G has an H-decomposition.

In Fig. 2, each vertex is given a unique prime number and the label of the product of all its parents (the label for the root node is 2). This means each label is a product of two factors: First factor is the number the labelling assigned to each vertex based on the prime labelling schema. The second part is the number that is inherited from the label of the parent. One advantage of this labelling methodology is for dynamic updates. Such that a new vertex is inserted; it is easy to give a prime number that has not been given before as the self-labelling for inserted node. As result there are no needs for re-labelling.

Theorem 1. Let $G = (V, \mathrm{E})$ be a directed acyclic graph. For any two given nodes $(u, v) \in V$ and two labels $L(u)$ and $L(v)$ for the two nodes u and v. if $L(u)$ *mod* $L(v) = 0$, then node u is ancestor of node v or there is a path between the two nodes.

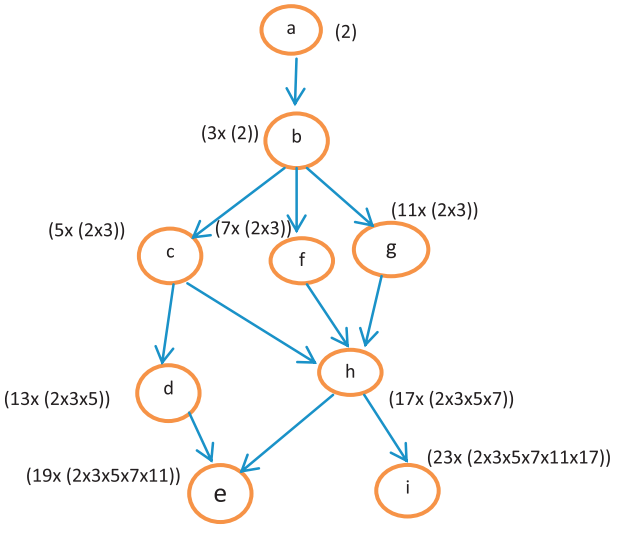

Figure 2 A prime number labelling schema for DAG graph.

Proof. This means that the label $L(v)$ assigned to the node v is a factor of the label $L(u)$ assigned to node u. Now assume that u is not an ancestor of v. Let p_v be the unique prime number assigned to node v. As $L(u)$ *mod* $L(v) = 0$, we must have $L(u)$ *mod* $p_v = 0$. As p_v is unique to v, and it was not associated with u during the initialization step, p_v must have propagated from one of the nodes v_1 which is a child of u. Continuing the same argument for v_1 and v, p_v must have propagated from one of the children v_2 of v_1. Continuing the same argument we reach at the conclusion that there is a path from u to v as without v the number p_v could not have propagated to u which is a contradiction. Hence, it is proved. □

Theorem 2. If node u is an ancestor of node v, then $L(u)$ *mod* $L(v) = 0$.

Proof. We show by induction that the label $L(v)$ associated with node v would be a factor of only the labels associated with all the nodes which are ancestors of v. □

According to Definition 1, any parent k of node v will enter L only after v enters L. Now for the edge (k, v) we will compute $N(k) = L.C.M\ (N(k), N(v))$. We have $L(Parent) = L.C.M\ (N(Parent), N(Child))$ i.e. $L(Parent)$ *mod* $L(Child) = 0$. Hence, $L(k)$ *mod* $L(v) = 0$.

Hypothesis. Let P be a set of ancestors of v. Then all nodes in P will contain the unique prime number associated with v. *Iteration step:* Now any node m which is a parent of any node p in P must have $L(m)$ *mod* $L(p) = 0$. Also from the hypothesis $N(p)$ *mod* $L(v) = 0$, So, $L(m)$ *mod* $L(v) = 0$. Therefore the result is true by induction.

4. Optimization techniques

In this section, we discuss the size requirements of our proposed labelling schema. This is because there is a direct impact between the storage size and the performance of the XML query processing. Moreover, our experimental results show that the idea of the *SCC* reduced the size of the original graph more than 30%. In the prime number labelling scheme, we carry out a depth-first traversal of the XML tree and assign to each node a prime number.

4.1. Maximum size of prime Label

The maximum number of bits required for a label is determined by the total number of the nodes in the XML file. Given an XML file with N nodes, we use θ_N to denote the maximal prime number that has been used to label the nodes. If the maximum level in the corresponding XML graph is D then the maximum number of bits required by the node labels at each level is given by $D\,log(\theta_N)$.

We assume that the bit length of the product of two numbers is the sum of the bit lengths of the two numbers. From the characteristics of prime numbers, we know that for an integer N, the number of prime numbers that is smaller than or equal to N is

$N(1/\log(N))$. Hence, the Nth prime number approximately is $N\log(N)$ and the number of bits needed to represent the Nth prime number is $\log(N\log(N))$. Note that the error ratio for using $\log(N\log(N))$ to predict the length of the binary representation of the Nth prime number is the logarithm of the difference between $N\log(N)$ and the Nth actual prime number. Therefore, although there is fluctuation in the difference between the actual prime number and the estimated prime number, the error ratio is small. Fig. 3 shows the difference between the length for the binary representation of the first 10,000 actual prime numbers and the estimated prime numbers. In the worst case the maximum size of prime number label given to a node is

$$L_{\max} = D \log\left(\left(\sum_{i=0}^{D} F^i\right) \log\left(\sum_{i=0}^{D} F^i\right)\right)$$

where (F = fan-out), D = depth of the XML files. It is clear from the above equation that it depend on the depth of the XML files. The authors in [5,36] proved that the depth of the real XML data is low with relatively fan-out, they perform a statistical analysis on 200,000 XML documents, and discovers that 99% of these documents have less than 8 levels of nesting's. Which also leads to the maximum number of levels in underling XML DAG graph will not increase about 8 levels. This leads to a small size required for storing XML graph.

4.2. Least common multiple

In this section, we will introduce an optimization technique to minimize the size of the label given to each node; our technique based on the Least Common Multiple (LCM). From Fig. 2, we notice that, there is a redundancy in the second part of the labels (ancestors-label). Given a series S of prime numbers are available $(2,3,5,7,\ldots)$, we start our numbering by assigning each node in DAG a distinct prime number. Now for each of the parent nodes we assign numbers according to the following rule.

- If parent node P has many children, say C1, C2 and the numbers associated with them are N(C1), N(C2), respectively, then the label associated with P is given by: $L(P)$ = **LCM** (N(C1), N(C2)).
- If parent node P has one child C1 with associated number N(C1), then the label associated with the parent P is $L(P) = N$(C1).

The Least Common Multiple of two numbers a and b, namely, ***LCM*** (a, b), is the smallest number m for which there is a positive integers n_a and n_b such that $n_a \times a = n_b \times b = m$. For example, the Common Multiples of *3* and *4* are 0, 12, 24, etc. The Least Common Multiple of two numbers is the smallest number (not zero) that is a multiple of both. For example, Multiples of *3* are 0, 3, 6, 9, *12*, 15, 18, 21, *24*, ... Multiples of *4* are 0, 4, 8, *12*, 16, 20, *24*, 28, etc. Then LCM of the two numbers *3* and *4* is *12, 24*, etc.

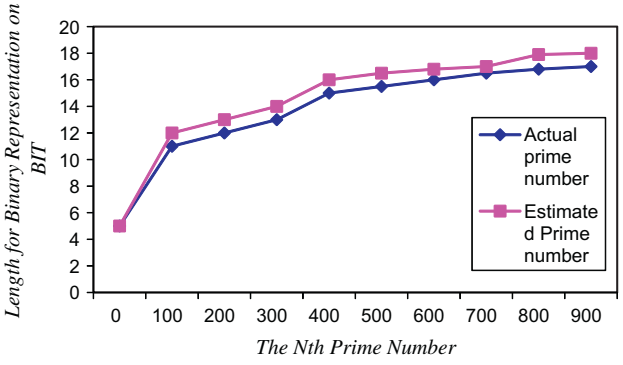

Figure 3 Estimated and actual prime number.

5. Experimental results

5.1. Experimental platform

The experiments are performed under two environments. The first experiment is performed on a Pentium IV-2 GHz platform with windows-XP and 788 MB of RAM. Oracle-OracHom 9.2 is used as a database server. The second experiment is performed on a Pentium 1V-3 GHz platform with Linux Suse 9.1 and 3 GB of RAM. IBM DB2 8.1 is used as a database server and a single hard disk with 120 GB. All strategies of our labelling schema are implemented as a database application (set of tables), using a java-based application to store the information into tables.

We compare our labelling schema with PLSD-FULL [7]; the comparison constructed on the space requirements and response times. Moreover, in the experiment the labels of the XML are stored in a relational database with table structure (*self-label*, *label*, *parent-label*, *uri*) which is more similar to PLSD-Full structure. To speed up query evaluation we build B-tree on self-labels, though indexes are necessary for ancestors-label and parents-label.

5.2. Data sets

As a real-life example of XML data with links, the Internet Movie Database (IMDB) [37] is used. The general characteristics of this data are as follows. The center file of the IMDB has a list of movies, each with a unique identifier. The actors of those movies are listed with their roles in a distinct file. All directors are listed in an independent file, with a number of important producers, writers, and cinematographers. The IMDB is used because it was identified as a highly cyclic database likely to stress the path-indexing algorithms.

Now, the generation of linked XML documents from this data. A small subset of movies and all people (actor, directors, etc.) associated with these movies is randomly chosen. One XML document for each *movie* is generated and an *XLink* to the *actors* and the *director* for the underlying movie is added. The portion of the used database is organized around *movie* elements and elements for classes of people who appear in movie credits, for example, *actor*, *director*, *composer*, etc., as well as a wide variety of information about movies. Cyclicity arises since each movie element is serviced as ID references and has pointers to individuals who acted in the movie, and each element representing an individual pointer to the movies in which she or he acted. This datasets consists of 40,211 nodes and 44,349 edges, among which 2718 are of type IDREF.

5.3. Space requirements

At first, the concepts are studied by using a small fragment of the movies database from the generated set as described above. This small fragment has *1235 nodes*; *1411* edges, *53 Xlinks, and*

Table 1 Data set description.

	# Vertices	# Edges	# Global links	# Internal links	# Cycles	Max. no. of nodes in cycle
XML graph	1235	1411	46	130	–	–
DAG	782	862	25	55	24	15

Table 2 Data set description with different nodes.

# Nodes	# Global links	# Internal links	# Cycles	# Nodes in cycle	Labels size	
					Our proposed schema	PLSD-FULL
2500	94	141	30	15	40 Bit	49 Bit
3200	110	20	95	15	46 Bit	55 Bit
5000	137	196	132	187	52 Bit	70 Bit
6240	163	210	140	152	55 Bit	85 Bit
7100	213	392	174	195	64 Bit	97 Bit
8312	287	401	213	286	69 Bit	110 Bit
9530	392	470	242	364	83 Bit	127 Bit

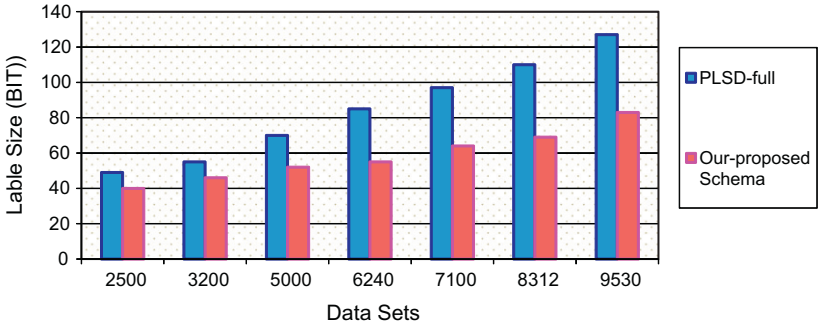

Figure 4 Space requirements.

Table 3 Test queries.

Query	Query in english	Path expression	Our-proposed schema		PLDS-full	
			Time (s)	#Result	Time (s)	# Result
Q1	Find all the descendants of a movie	movie//*	0.54	408	1.2	1064
Q2	Find all the descendants of actor	actor//*	0.61	651	1.4	2011
Q3	Find all the descendants of title with its name	title[“Animal Instinct”]//*	0.2	1	0.3	1
Q4	Find all the descendants of a director by its name	director [”Zemeckis”]//*	0.2	1		1
Q5	Find all the descendants of casting node	casting//*	0.45	209	1.1	879
Q6	Find all the descendants of identifier 8	8//*	0.1	1	0.1	1
Q7	Find all the descendants of identifier 100	100//*	0.11	1	0.16	1
Q8	Find all the descendants of identifier 300	300//*	0.2	1	0.28	1
Q9	Find all the descendants of identifier 1000	1000//*	0.2	1	0.2	1
Q10	Find all the descendants of identifier 4000	4000//*	0.31	1	0.4	1

123 IDREFS. Then, we apply the SCC principle; the cycles from the original XML graph are removed. The underlying small fragment has *24 cycles*. The maximum number of nodes in one cycle is *15 nodes*. The result is a DAG with *782 nodes* and *862 edges*. Table 1 describe this information.

This means that the optimization techniques based on the principle of the SCC, reduce of the size of DAG graph more than 30% compared with the original XML graph. Reduce the number of nodes will reduce the size of the labels later on. To evaluate the space requirements for our proposed labelling schema we increase regularly the size of the underlying XML fragments (see Table 2).

From Table 2 and Fig. 4we noticed our proposed labelling schema able to achieve up to 27% reduction in the maximum label size compared with PLSD-FULL schema. The main reason is that in PLSD-Full used three external tables to store additional information besides spanning tree; each table needs one independent index. The second reason is that our idea

based on creating labels of the fly while processing the XML files. However, PLSD-Full wait to create additional tables for non-spanning trees.

5.4. Response times

In this set of experimental, we used the subset of Movies Databases which described in Table 2 to study the query performance of our proposed labelling schema against PLSD-full. Ten path expression queries are submitted to the database. Table 3 shows these 10 path expressions. The first five path expressions test the reachability by node names and the second five path expressions test the reachability by node identifiers. The last two columns explain the execution time for each path expressions using our proposed labelling schema and PLSD-Full schema. The table shows that about "30%" of the reachability queries can be determined in a linear time by looking only at SCC tables. Moreover, It is observed that proposed labelling schema provides significantly better performance than the PLSD-Full. It is more than an order of magnitude better than PLSD-Full schema when testing reachability queries.

It is also observed that the execution time needed to test the reachability between two nodes depends on the distance between these nodes. For example, the time needed to execute the path query "200//12100" is 3.1 s, whereas, the time needed to execute the path query "8//25" is 0.01 s.

6. Conclusion and future work

Labelling schema is a family of technologies widely used in indexing tree or graph structured XML graph which assign a unique identifier to each node in the tree or the graph. Our main objective for designing labelling schemas for DAG XML graph is to allow fast reachability test between any two given nodes. This labelling schema is based on the idea of prime numbers and strongly connected to the components principle. Moreover, operation on DAG such as ancestors, descendants, and siblings could also be easily covered. The optimization techniques reduced space requirements and time consumption. Our labelling schema could be applied for any arbitrary XML graph. Our experimental results show that; the proposed labelling schema for DAG XML needs less space and less construction time compared with PLSD-Full schema. The main reason is that no additional information is required to be stored for non-spanning tree edges and that the utilization of elementary arithmetic operations avoid time-consuming database operations. Moreover, for complex XML collection with many links this type of labelling is not efficient.

References

[1] McHug J, Wisdom J. Query optimization for XML. In: Proceedings of the 25th international conference of very large data bases (VLDB), Edinburgh, Scotland; 1999.

[2] Santoro N, Katib R. Labelling and implicit routing in network. Comput J 1985;28(1):5–8.

[3] O.C.L. Center. Dewey decimal classification; 2009. <http://www.oclc.org/dewey>.

[4] Cohen, Halperin Eran, Kaplan Harin, Zwick Uri. Reachability and distance queries via 2-hop labels. In: Proceedings thirteenth annual ACM-SIAM symposium on discrete algorithms. ACM Press; 2002. p. 937–46.

[5] Wu X, Lee M, Hsu W. A prime number labeling scheme for dynamic ordered XML trees. In: Proceedings of the 20th international conference on data engineering (ICDE), Boston, USA; 2004.

[6] Sayed Awny. A prime number labelling scheme for reachability queries over complex XML collection. In: The 4th Indian international conference on artificial intelligence (IICAI-09); 2009.

[7] Wu Gang, Zhang Kuo, Liu Can, Li Juan-Zi. Adapting prime number labelling scheme for directed acyclic graphs.DASFAA; 2006. p. 787–96.

[8] Li Q, Moon B. Indexing and querying XML data for regular path expressions. In: 27th International conference on very large data bases (VLDB); 2001. p. 361–70.

[9] Yoshikawa M, Amagasa T. XRel: a path-index based approach to storage and retrivel XML documents using relational databases. ACM Transactions on Internet Technology (TOIT); 2001.

[10] Cohen E et al. Labeling dynamic XML trees. In: Symposium on principle of databases (POSD); 2002. p. 271–81.

[11] Tatarinov SD, Zhang C. Storing and querying ordered XML using a relational database system. In: ACM SIGMOD international conference on management of data; 2002. p. 204–15.

[12] Zhang C, Naughton JF, DeWitt DJ, Luo Q, Lohman G. On supporting containment queries in relational database mangement system. In: ACM SIGMOD international conference on management of data; 2001.

[13] Chein Shu-Yaho et al. Efficient structural joins on indexed XML documents. In: 28th international conference on very large data bases (VLDB); 2002.

[14] Kaplan H, Milo Tova, Shabo Ronen. A comparison of labeling schemes for ancestor queries. In: 13th ACM-SIAM symposium on discrete algorithms (SODA); 2002. p. 954–963.

[15] Kaplan H, Milo T. Short and simple lables schemes for small distances and other functions. In: 7th international workshop on algorithms and data structures (WADS); 2001. p. 246–57.

[16] Abiteboul S. Compact labeling schemes for ancestor's queries. In: 12th ACM-SIAM symposium on discrete algorithms (SODA); 2001. p. 547–56.

[17] Chung C-W, Min J-K, Shim K. APEX: an adaptive path index for XML data. In: ACM SIGMD 2002, USA; 2002.

[18] Goldman R, Widom J. DataGuides: enabling query formulation and optimization in semistructured databases. In: Jarke M, Carey MJ, Dittrich KR, Lochovsky FH, Loucopoulos P, Jeusfeld MA, editors. VLDB'97, Proceedings of 23rd international conference on very large data bases, August 25–29, 1997, Athens, Greece, Morgan Kaufmann; 1997. p. 436–45.

[19] Cooper Brian F, Sample Neal, Franklin Michael J, Hjaltason Gísli R, Shadmon Moshe. A fast index for semistructured data; VLDB 2001. In: Proceedings of 27th international conference on very large data bases, September 11–14, 2001, Roma, Italy. Morgan Kaufmann 2001, ISBN 1-55860-804-4; 2001.

[20] Kaushik R et al. Covering indexes for branching path queries. In: ACM SIGMOD international conference on management of data; 2002. p. 133–44.

[21] Abiteboul S, Bunmen P, Suciu D. Data on the web: from relations to semistructured data and XML. Los Atlos, CA 94022, USA: Morgan Kaufmann Publishers; 1999.

[22] Milo T, Suciu D. Index Structures for path expressions. In: 7th International conference on database theory (ICDT); 1999. p. 277–95.

[23] Kaushik R et al. Exploiting local similarity for indexing paths in graph-structured data. In: 18th International conference on data engineering (ICDE); 2002.

[24] Qun C et al. $D(K)$-Index: an adaptive structural summares for graph-based data. In: ACM SIGMOD international conference on mangement of data; 2003. p. 134–44.

[25] XML Linking Language(XLink) Version 1.0, W3C Recommendation (27 June 2001), see <http://www.W3.org/TR/xlink>.

[26] XML Pointer Language (XPointer), W3C Working Draft (16 August 2002), see <http://www.w3.org/TR/xptr>.
[27] Jiang H, Luz H, Wang W, Chin Ooi B. XR-tree: indexing XML data for efficient structural join. ICDE; 2003.
[28] Dietz PF. Maintaining order in a linked list. In: STOC; 1982. p. 122–7.
[29] Chen L, Gupta A, Kurul ME. Efficient algorithms for pattern matching on directed acyclic graphs. In: ICDE; 2005. p. 384–5.
[30] Chen L, Gupta A, Kurul ME. Stack-based algorithms for pattern matching on dags. In: VLDB; 2005. p. 493–504.
[31] Jin R, Xiang Y, Ruan N, Wang H. Efficiently answering reachability queries on very large directed graphs. In: SIGMOD; 2008. p. 595–608.
[32] An Dong Chan, Park Seog. Group-based prime number labelling scheme for XML data. In: 10th IEEE international conference on computer and information technology, CIT 2010, Bradford, West Yorkshire, UK, June 29–July 1; 2010.
[33] Schenkel R et al. HOPI: an efficient connection index for complex XML document collections. In: 9th International conference on extending database technology (EDBT); 2004. p. 237–55.
[34] Sayed A, Unland R. Index-support on XML documents containing links. In: IEEE midwest symposium on circuits and, system; 2003.
[35] Nuutila Esko, Soisalon-Soininen. Efficient transitive closure computation. Technical, report TKO-B113; 1993.
[36] Yoshikawa M, Amagasa T. XRel: a path-based approach to storage and retrieval of XML documents using relational databases. In: ACM transactions on internet technology (TOIT); 2001.
[37] The Mondial Database. <http://dbis.informatik.uni-goettingen.de/Mondial/>.