applied sciences

MDPI

Article

Efficiently Answering Reachability Queries for Tree-Structured Data in Repetitive Prime Number Labeling Schemes

Jinhyun Ahn [1], Taewhi Lee [2] and Dong-Hyuk Im [3,*]

[1] Department of Management Information Systems, Jeju National University, Jeju 63243, Korea; jha@jejunu.ac.kr
[2] Smart Data Research Group, Electronics and Telecommunications Research Institute, Daejeon 34130, Korea; taewhi@etri.re.kr
[3] Department of Computer and Information Engineering, Hoseo University, Asan 31499, Korea
* Correspondence: dhim@hoseo.edu

Received: 29 March 2018; Accepted: 12 May 2018; Published: 15 May 2018

Abstract: Reachability queries plays a crucial role in accessing relationships between nodes in tree-structured data. Previous studies have proposed prime number labeling schemes that answer reachability queries using arithmetic operations. However, the prime numbers in these schemes can become very large when a tree contains a considerable number of nodes; thus, it is not scalable. Recently, a repetitive prime number labeling scheme that reduces space requirements was proposed. Unfortunately, it suffers from slow query processing, owing to the complexity of its reachability test. In this paper, we propose a more efficient method for answering reachability queries in a repetitive prime number labeling scheme. The results of experiments using real-world XML datasets show that our approach reduces reachability query processing times.

Keywords: reachability; prime number labeling; tree; XML

1. Introduction

A vast amount of tree-structured data on diverse domains is available in eXtensible Markup Language (XML) files on the Web, such as SwissProt (protein sequence database), DBLP (computer science bibliography), and Treebank (tagged sentences). A query that determines whether there exists a path between the two nodes of a given pair (source and target) is an important one for trees. It can also be regarded as a reachability or ancestor–descendant query. A straightforward manner to address it is tree traversal, starting from the source node and then visiting intermediate nodes until arriving at the target node. However, it is not feasible to use this method in the case of large trees.

To avoid visiting intermediate nodes, labeling schemes have been proposed [1]. Existing works are classified into containment-based [2,3], prefix-based [4,5], and prime number labeling schemes. In this paper, we focus on a prime number labeling scheme that does not require all the nodes to be re-labeled when some nodes are updated. An earlier study on a prime number labeling scheme was reported in [6]. In this scheme (hereafter, PRM), each node is assigned a globally unique prime number, called a *self label*. The node label is then recursively computed by multiplying its *self label* and its parent node's label. The pair of two nodes is reachable if the target node label is divisible by the source node label. An approach has been proposed to utilize the MapReduce framework in order to do prime number labeling of massive XML data [7]. The schemes' shortcomings are apparent when the number of nodes becomes very large—the size of *self labels* increases as well. In addition, they did not consider the performance of answering reachability queries.

Attempts to reuse *self labels* were made. In [8,9], a repetitive prime number labeling scheme (hereafter, REP) was described that reuses prime numbers inherited from parents. In [10], the order of ancestors' *self labels* is encoded based on the Chinese Remainder Theorem (hereafter, CRT). A drawback of these approaches is that their inefficient method for performing reachability tests significantly reduces their usability in the case of large datasets. In this paper, we propose an efficient method for answering reachability queries in REP.

2. Repetitive Prime Number Labeling Scheme

In this section, REP is explained, and then, we explain the drawbacks related to its inefficient method for performing reachability tests. Let v and w be nodes in a tree. If an edge (v, w) exists, then v is the parent of w. We denote by $v = pa(w)$ the parent node of w. Furthermore, w is reachable from v if there exists a sequence of nodes starting at v and ending at w, where consecutive node pairs in the sequence are in edges. In REP, each node is assigned a prime number, called a *self label*.

Definition 1. ***Self Label****: $self(v)$ is the self label of a node v such that $self(v)$ is the $s(v)$-th prime number, where*

$$s(v) = \begin{cases} 1 & \text{if } v \text{ is root} \\ s(pa(v)) + order(v) & \text{otherwise} \end{cases}$$

where $order(v)$ is the sibling order starting from 0. The 1-th prime number is 2, which means that $self(v)$ is 2 if v is a root.

Note that a *self label* in PRM [6] is a globally unique prime number, whereas in REP, *self labels* are locally unique prime numbers within sibling nodes. We now provide the definition of the label of a node.

Definition 2. ***Node Label****: $L(v)$ is the label of a node v, which is a composite number computed as*

$$L(v) = \begin{cases} 2 & \text{if } v \text{ is root} \\ L(pa(v)) \times self(v) & \text{otherwise} \end{cases}$$

Lemma 1. *Given a node v, $self(v)$ is equal to $lpf(L(v))$, where $lpf(\mathrm{n})$ is the largest prime factor of* n.

(B) in Figure 1 depicts an illustrative example labeled by Definition 2. For example, $L(I)$ is computed by multiplying $L(F) = 2 \times 3 \times 3$ by $self(I) = 5$. A tree-labeling algorithm for this scheme is outlined in Algorithm 1.

Algorithm 1 Labeling

```
1: procedure AssignLabels(T,p) ▷ T is a tree and p is a parent node (the root for the first-time call)
2:     for the i-th child node c of p in T do
3:         self(c) ← the i-th next prime number from self(p)        ▷ Definition 1
4:         L(c) ← L(p) × self(c)                                    ▷ Definition 2
5:         AssignLabels(T,c)
6:     end for
7: end procedure
```

In [8], a prime factorization method to answer a reachability query in REP was described. Both labels of the given two nodes must be prime factored; subsequently, prefix matching is performed against the two prime factor sequences. For example, referring to (B) in Figure 1, H is not reachable

from D as $2 \times 2 \times 3$ is not prefix-matched in $2 \times 3 \times 3 \times 3$. On the other hand, H is reachable from E as 2×3 is a prefix. We argue that prime factorization is an expensive task, which led us to devise a more efficient method of answering reachability queries, as explained in the next section.

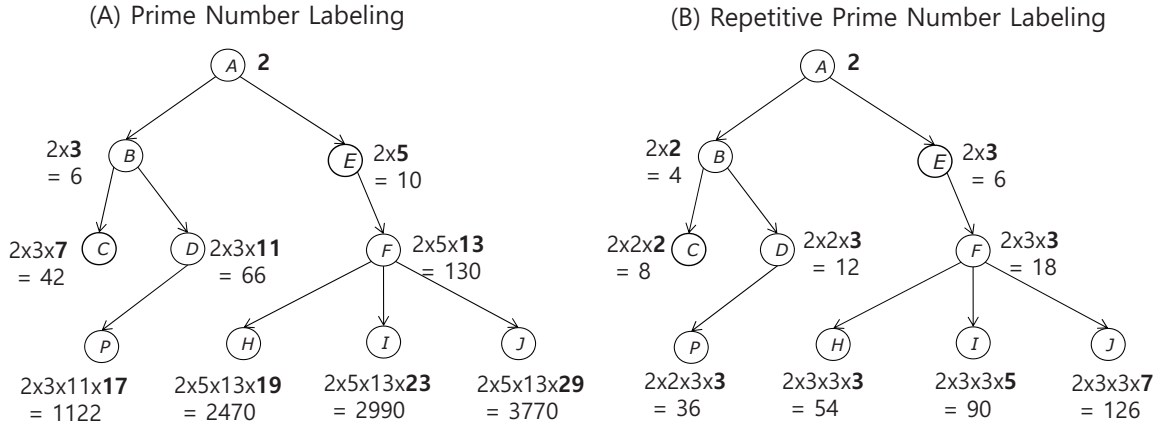


Figure 1. (**A**) is produced by `PRM` and (**B**) by `REP`. *self labels* are in bold. Note that a node label that is in fact to be stored is a single composite number computed by a multiplication of its prime numbers sequence. It can be easily seen that node labels produced by `PRM` are much bigger than those by `REP`.

3. Improving on Answering Reachability Queries

We observed that if a label is not divisible by the other label, it is not reachable. Referring to (B) in Figure 1, for example, $L(H)$ is not divisible by $L(D)$ and H is not a descendant of D. Even if these nodes have the same *self label* 3, they differ in terms of their ancestor nodes' *self labels*, as in the case of $self(B) = 2$ and $self(F) = 3$. This observation allows us to devise a simpler method of making decisions for unreachable pairs, which is proved in Theorem 1.

Theorem 1. *If* $L(v) \mod L(w)$ *is not* 0, *then* v *is not a descendant of* w.

Proof. In a proof by contrapositive, first we assume that v is a descendant of w. By Definition 2, because the label is computed recursively, $L(v)$ contains the label of its parent node $L(pa(v))$ as a factor, $L(pa(v))$ contains $L(pa(pa(v)))$, and so on. This indicates that $L(v)$ contains $L(w)$ as a factor, that is, $L(v)$ is divisible by $L(w)$. □

However, the inverse of Theorem 1 does not always hold, i.e., if $L(v) \mod L(w)$ is 0, then v is a descendant of w. Consider the two node pairs (E, H) and (E, P) in (B) of Figure 1. $L(H)$ is divisible by $L(E)$ and H is a descendant of E. This is trivial, because a child node label must have its parent node label as a factor. On the other hand, $L(P)$ is also divisible by $L(E)$, but P is indeed not a descendant of E. This occurs because both have the same *self labels* and their ancestor nodes also have the same *self labels* (i.e., $self(B) = self(A) = 2$). Definition 2 leads us to consider the quotient of $L(P) \div L(E) = 2 \times 3$ and $self(E) = 3$. If P is a descendant of E, every factor of the quotient must be larger than or equal to $self(E)$, because the *self label* of a child node is larger than or equal to any *self labels* of its parent nodes. This is not true in this case, which means that P is not a descendant of E. We formally define this observation in Theorem 2.

Theorem 2. *Let* $spf(\mathrm{n})$ *be the smallest prime factor of a number* n. *Let* q *be the quotient of* $L(v) \div L(w)$. *Assume that* $L(v) \mod L(w)$ *is* 0. *If* $self(w) \leq spf(\mathrm{q})$, *then* v *is a descendant of* w; *otherwise,* v *is not a descendant of* w.

Proof. In a direct proof, because $L(v) \mod L(w)$ is 0, we have

$$L(v) = L(w) \times \mathrm{q} \tag{1}$$

There exist two cases for q. Assume that q is a prime number. Consider the case ($self(w) \leq spf(\mathrm{q})$). It is trivial that $spf(\mathrm{q}) = \mathrm{q}$ and q must be $self(v)$ by the antecedent and Property 1. Therefore, Equation (1) can be rewritten as $L(v) = L(w) \times self(v)$. By Definition 2, we have $w = pa(v)$, which proves the consequent. Now, consider the case ($self(w) > spf(\mathrm{q})$). Because $self(w)$ is the largest factor of $L(w)$ (by Property 1) and is larger than q, $self(w)$ is the largest factor of $L(v)$, which means that $self(v) = self(w)$. We now rewrite Equation (1) as $L(v) = (L(w)/self(v) \times \mathrm{q}) \times self(v)$. By Definition 2, there exists a node $h = pa(v)$ such that $L(h) = L(w)/self(v) \times \mathrm{q}$. Consider Theorem 1. Because $L(h)$ mod $L(w)$ is not 0, $h = pa(v)$ is not a descendant of w. Therefore, v is not a descendant of w, which proves the consequent.

Assume the second case for Equation (1), where q is a composite number. We consider $pa(v)$ and then replace $L(w)$ in Equation (1) as follows:

$$L(v) = L(pa(v)) \times lpf(\mathrm{q}) \tag{2}$$

where

$$L(pa(v)) = L(w) \times \mathrm{q}' \tag{3}$$

such that $\mathrm{q} = \mathrm{q}' \times lpf(\mathrm{q})$. If q' is still a composite number, $L(w)$ in Equation (3) can further be rewritten as $L(pa(v)) = L(pa(pa(v))) \times lpf(\mathrm{q}')$, where $L(pa(pa(v))) = L(w) \times \mathrm{q}''$, such that $\mathrm{q}' = \mathrm{q}'' \times lpf(\mathrm{q}')$. This process can be continued until we have a node label $L(pa(...pa(v))) = L(w) \times spf(\mathrm{q})$. Now, assume a node $r = pa(...pa(v))$. Applying the same proof as above for the case where q is a prime, we are able to determine the ancestor-descendant relationships between r and w. If r is determined to be a descendant of w, v must be a descendant of w, because v is a descendant of r. If r is determined not to be a descendant of w, v must not be a descendant of w because every descendant of r is not a descendant of w. □

We devised an algorithm based on the theorems to test the reachability of two ordered nodes, which is outlined in Algorithm 2. In implementation level, we used BigInteger class in Java to handle very large integers. An iterative algorithm was implemented to realize line 7–12. The algorithm begins with the first prime factor $x = 2$ of Lw. We check if x is larger than $spf(\mathrm{q})$. If so, the algorithm terminates and returns FALSE (line 9). Otherwise, we compute Lw^1 by iteratively dividing Lw with x. In other words, Lw^1 is obtained by removing all prime factors x in Lw (e.g., if $Lw = 2 \times 2 \times 3$ and $x = 2$ then $Lw^1 = 3$). We move x to the next prime factor (e.g., 3) and repeat the above process until x is smaller than Lw^k in k-th iteration. It returns TRUE (line 11) if the iteration finishes.

Algorithm 2 Querying

```
1: procedure IsReachable(Lw, Lv)                    ▷ Lw and Lv are node labels
2:     r ← Lv mod Lw
3:     if r ≠ 0 then                                ▷ Theorem 1
4:         return FALSE
5:     end if
6:     q ← Lv ÷ Lw
7:     self_Lw ← the largest prime factor of Lw
8:     if self_Lw > spf(q) then                     ▷ Theorem 2
9:         return FALSE
10:    else
11:        return TRUE
12:    end if
13: end procedure
```

4. Evaluation

Since in our study we focused on a prime number-based labeling scheme, containment-based and prefix-based approaches are beyond the scope of this paper. Experiments were conducted to compare our approach (REP+) with PRM [6], REP [8], and CRT [10]. Note that REP+ and REP employ the same labeling schema, called REP, but different reachability test methods. The Sections 4.1 and 4.2 are intended to convince that REP is the state-of-the-art approach in the area of prime number labeling of trees. Our contribution is to improve on reachability query processing of REP which is demonstrated in Section 4.3. The experiments were performed on a machine with a 2.6 GHz CPU and 128 GB RAM. XML datasets were obtained from [11], with varied numbers of nodes, fanouts, and depths (see Table 1). The experimental results were averaged over five runs.

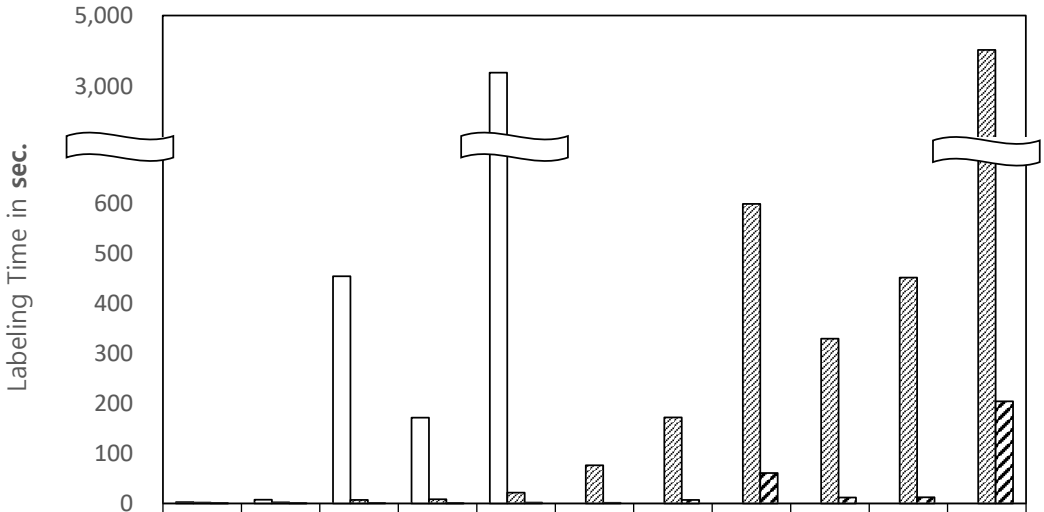
Table 1. Experimental datasets.

ID	Name	Nodes	Avg. Fanout (max)	Avg. Depth (max)
1	SigmodRecord	20,334	1.8 (89)	5.4 (6)
2	mondial-3.0	64,994	5.2 (1913)	3.1 (5)
3	partsupp	96,488	2.0 (16,001)	2.7 (3)
4	uwm	192,421	3.4 (4225)	3.4 (5)
5	orders	300,616	2.0 (30,001)	2.8 (3)
6	nasa	1,598,934	3.4 (4871)	5.3 (8)
7	lineitem	2,048,247	2.0 (120,351)	2.9 (3)
8	dblp	6,504,386	2.0 (328,858)	3.0 (6)
9	Treebank_e	7,337,533	3.0 (112,804)	7.3 (36)
11	psd7003	39,118,126	1.3 (262,526)	5.8 (7)

4.1. Labeling Time

Figure 2 depicts the labeling time for each dataset. Because a substantial amount of time was spent by CRT on smaller datasets, we omit its results for larger datasets. In all cases, REP+, and REP, outperformed PRM and CRT. Notable results were observed for large datasets, such as Treebank_e, SwissProt, and psd7003. As the time expended by PRM increased exponentially, the time expended by REP+ increased gradually. This occurred because PRM assigned larger prime numbers, and thus, additional time was required to determine the next prime number. REP+ uses much smaller prime

numbers because it reuses them. CRT also reuses prime numbers; however, it considers every *self label* of ancestor nodes, which significantly reduces its speed.



	1	2	3	4	5	6	7	8	9	10	11
CRT	2.8	7.4	454	172	3,381						
PRM	1.8	2.4	7.0	8.5	22	77	172	599	329	452	4,028
REP+, REP	0.1	0.2	0.9	0.4	2	2	7	61	12	12	204

Figure 2. Labeling time (X-axis indicates datasets in Table 1).

The performance results for dblp, which are poorer than the results for the larger dataset Treebank_e, are noteworthy. We observe that dblp has the largest max fanout, which means that there exists a node with a large number of child nodes. In REP+, the existence of a large number of sibling nodes is likely to increase the size of the prime numbers.

4.2. Space Requirements

Figure 3 shows a comparison of the space requirements of the approaches. For all datasets, REP+ requires less space than PRM. CRT produced better results than REP+ for some datasets; however, this is not a concern, because the labeling process is too slow to take advantage of the lower space requirements. It may be interesting to analyze datasets such as SigmodRecord, mondial-3.0, and nasa. We observed that these datasets have larger average depths and smaller max fanouts than the others (see Table 1). Because REP+ reuses the prime numbers of the parent nodes, a smaller fanout of a certain node could lead to its grandchild nodes beginning with smaller *self labels*. Meanwhile, in REP+, a taller tree (e.g., Treebank_e) does not necessarily mean that leaf nodes have larger prime numbers. Assume an extreme case of a leftist tree. REP+ uses only a prime number 2, while PRM uses as many prime numbers as the number of nodes. For datasets such as partsupp, orders, lineitem, and dblp, REP+ provides a saving in space of only 10% as compared to PRM. These datasets show large max fanouts.

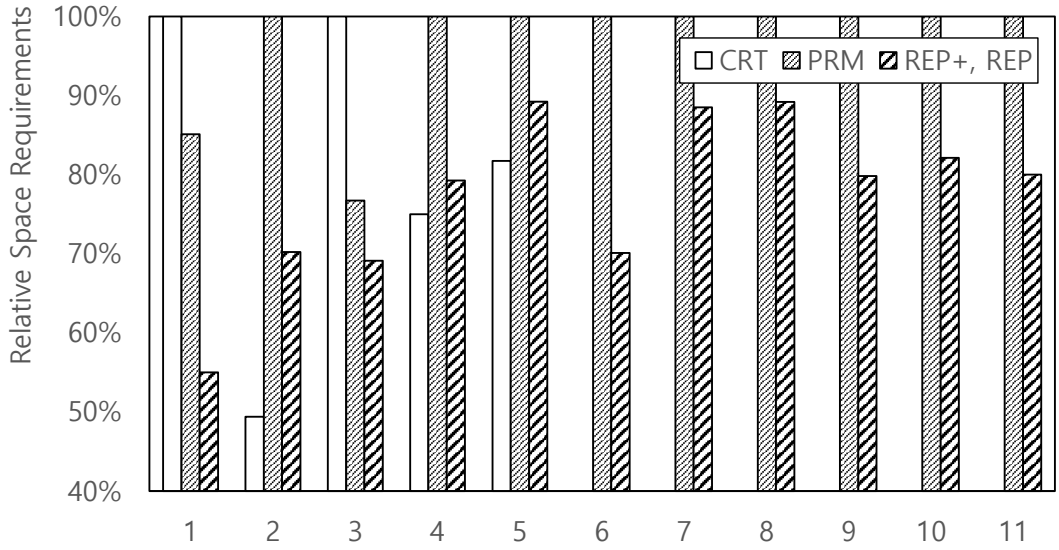


Figure 3. Space requirements (X-axis indicates datasets in Table 1).

4.3. Query Processing Time

We randomly selected 1000 TRUE (reachable) and FALSE (non-reachable) node pairs from each dataset. The query processing time was measured to be the total time required to determine the reachability of 1000 node pairs.

For TRUE pairs (Figure 4), `CRT` showed the worst performance, as it requires prime factorization and a comparison of all remainders. The other approaches showed a similar performance. However, for the larger datasets, such as SwissProt and psd7003, `REP+` showed the best performance. The reason is that, as already shown in the previous section, the space requirements of `REP+` are the smallest, which renders the node labels smaller than those of other labeling schemes.

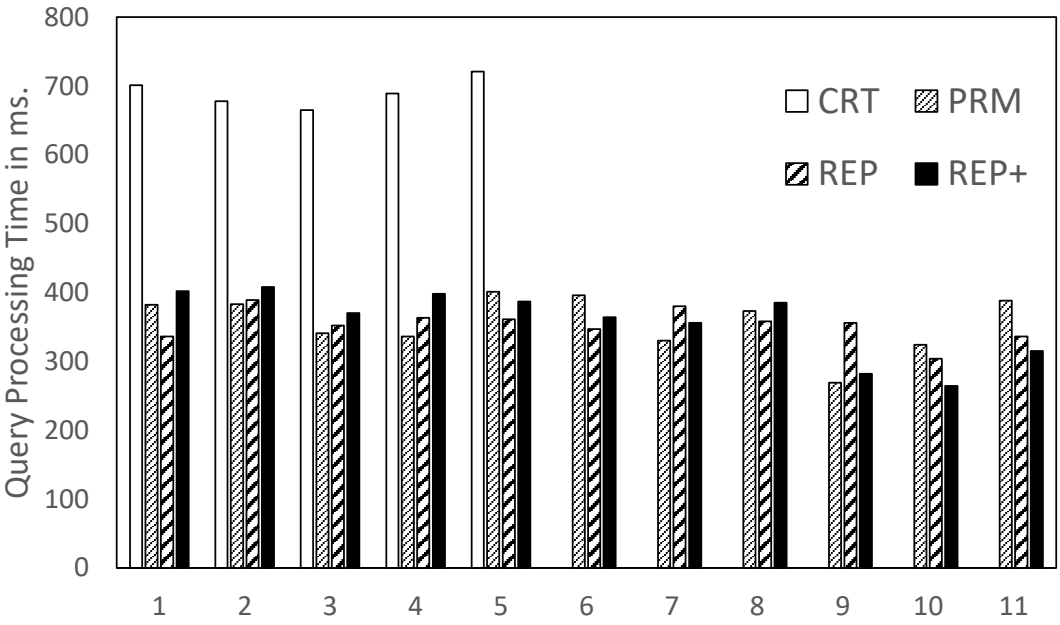


Figure 4. Query processing time for TRUE pairs (X-axis indicates datasets in Table 1).

For FALSE pairs (Figure 5), `REP` showed the worst performance. The prime factorization and prefix matching of prime number sequences required by `REP` reduces its speed. Note that there is a

big performance difference between REP and REP+, which employ the same labeling scheme. Consider line 2 in Algorithm 2. In REP+, if a label is not divisible by the other label, this signifies a FALSE pair. No additional operations are required for FALSE pairs, which allows REP+ to be much faster than REP. That is, REP+ immediately terminates when the remainder is zero. It makes REP+ to be more efficient than REP in terms of CPU time. Meanwhile, CRT is faster than REP because CRT stores *parent labels* and *self labels* separately, whereas REP stores only one composite number. Thus, CRT often has smaller number labels than REP, which reduces query processing time.

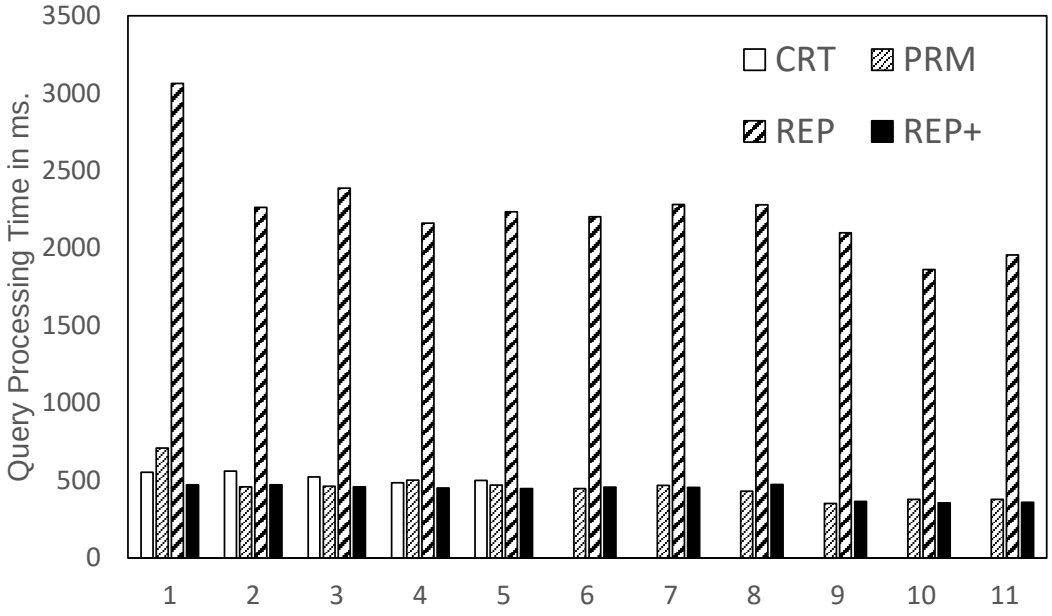


Figure 5. Query processing time for FALSE pairs (X-axis indicates datasets in Table 1).

Considering both TRUE and FALSE pairs, PRM is the only method that is comparable to REP+. PRM requires a single division operation for both TRUE and FALSE pairs, and is the fastest of the prime number labeling approaches thus far. Nevertheless, it was slower than REP+ for the larger datasets. This is because REP+ has smaller labels than PRM, which can be seen in its space requirements. This leads us to regard REP+ as a scalable approach. Moreover, since REP+ also outperforms PRM in terms of labeling time and space requirements, REP+ should be considered the most efficient approach overall.

5. Conclusions

We proposed an efficient method of answering reachability queries in a state-of-the-art prime number labeling scheme for XML data. Our contribution is that we devised a novel algorithm to determine the structural relationships between two nodes. We showed experimentally that our approach is faster than existing approaches. However, the proposed method failed to show better performance for TRUE pairs. It might be interesting to devise a faster way to determine reachability of TRUE pairs. We plan to design and implement an efficient mechanism to address update issues when there are insertions and deletions of nodes in a tree. In addition, we will devise a distributed algorithm to label XML datasets.

Author Contributions: D.-H.I. conceived the problem and supervised the whole research work; T.L. figured out some points that helped J.A. write the theorems; T.L. revised the theorems; J.A. implemented the algorithm and performed the experiments; J.A., T.L. and D.-H.I. wrote the paper.

Funding: This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2017R1C1B1003600) and Institute for Information & Communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. R0113-15-0005, Development of a Unified Data Engineering Technology for Largescale Transaction Processing and Real-Time Complex Analytics).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Klaib, A.; Lu, J. Investigation into Indexing XML Data Techniques. In Proceedings of the International Conference on Internet Computing and Big Data, Kuala Lumpur, Malaysia, 17–19 November 2014.
2. Xu, L.; Bao, Z.; Ling, T.W. A dynamic labeling scheme using vectors. In *Database and Expert Systems Applications*; Springer: Berlin, Germany, 2007.
3. Li, C.; Ling, T.W. QED: A novel quaternary encoding to completely avoid re-labeling in XML updates. In Proceedings of the 14th ACM International Conference on Information and Knowledge Management, Bremen, Germany, 31 October–5 November 2005.
4. Xu, L.; Ling, T.W.; Wu, H.; Bao, Z. DDE: From dewey to a fully dynamic XML labeling scheme. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, Providence, RI, USA, 29 June–2 July 2009.
5. Tatarinov, I.; Viglas, S.D.; Beyer, K.; Shanmugasundaram, J.; Shekita, E.; Zhang, C. Storing and querying ordered XML using a relational database system. In Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, WI, USA, 3–6 June 2002.
6. Wu, X.; Lee, M.L.; Hsu, W. A prime number labeling scheme for dynamic ordered XML trees. In Proceedings of the 20th International Conference on Data Engineering, Boston, MA, USA, 2 April 2004.
7. Ahn, J.; Im, D.H.; Lee, T.; Kim, H.G. Parallel Prime Number Labeling of Large XML Data Using MapReduce. In Proceedings of the 2016 6th International Conference on IT Convergence and Security (ICITCS), Prague, Czech Republic, 26–29 September 2016; pp. 1–2, doi:10.1109/ICITCS.2016.7740360. [CrossRef]
8. Sun, D.H.; Hwang, S.C. A Labeling Methods for Keyword Search over Large XML Documents. *J. KIISE (Korean Inst. Inf. Sci. Eng.)* **2014**, *41*, 699–706. [CrossRef]
9. Ahn, J.; Im, D.H.; Lee, T.; Kim, H.G. A dynamic and parallel approach for repetitive prime labeling of XML with MapReduce. *J. Supercomput.* **2017**, *73*, 810–836, doi:10.1007/s11227-016-1803-y. [CrossRef]
10. Morozov, S.; Saiedian, H.; Wang, H. Reusable Prime Number Labeling Scheme for Hierarchical Data Representation in Relational Databases. *J. Comput. Inf. Technol.* **2014**, *22*, 31–44. [CrossRef]
11. University of Washington. XMLData Repository. Available online: http://aiweb.cs.washington.edu/research/projects/xmltk/xmldata/ (accessed on 17 March 2018).