Journal of Theoretical and Applied Information Technology
15th February 2017. Vol.95. No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OPTIMIZATION TECHNIQUE FOR PRIME NUMBER LABELING OF DIRECTED ACYCLIC GRAPHS

[1]JINHYUN AHN, [2]DONG-HYUK IM, [3]TAEWHI LEE AND [1,4]HONG-GEE KIM

[1]Biomedical Knowledge Engineering Lab. and Dental Research Institute, Seoul National University

[2]Department of Computer and Information Engineering, Hoseo University

[3]BigData Intelligence Research Department, Electronics and Telecommunications Research Institute

[4]Institute of Human-Environment Interface, Seoul National University

E-mail: jhahncs@snu.ac.kr, dhim@hoseo.edu, taewhi@etri.re.kr, hgkim@snu.ac.kr

ABSTRACT

Directed Acyclic Graphs (DAGs) are widely used data structures. Labeling schemes have been proposed to accelerate reachability query processing over DAGs. Among such schemas, prime number labeling has attracted researchers because of its simplicity: a division operation is sufficient for answering reachability queries. However, the limitation of existing prime number labeling algorithms is that they generate huge label sizes. Minimizing the label size is an important issue because query-processing performance depends on it. In this paper, we propose techniques for generating prime number labels that are much smaller than those made by existing algorithms. In order to reduce the overall label size, some heuristics are suggested for sorting the vertices to which prime numbers are assigned. Experiments over real-world and synthetic DAG datasets show that the proposed approach produces smaller labels than existing approaches, and the labeling time is also reduced.

Keywords: *Directed Acyclic Graph, DAG, Prime Number Labeling, Label Assignment Order, Label Size*

1. INTRODUCTION

Directed acyclic graphs (DAGs) are popular data structures for representing knowledge. The real-world knowledge modeled by DAGs can be found in social networks, biological networks, traffic networks, and software version management. For example, protein-protein interactions, which are important in biology, can be modeled by regarding proteins as vertices and interactions as edges. One might be interested in mining the relationship between two given proteins in terms of protein-protein interactions. In the context of DAGs, such relationship can be identified by checking whether a vertex is reachable from another vertex. Note that reachability queries over a graph and its corresponding DAG are equivalent because a graph can be transformed directly into DAG by condensing the cycles [9]. This allows us to focus on DAGs in this paper without loss of generality.

Reachability queries over DAGs can be answered by computing transitive closures. In order to avoid such expensive computations, labeling schemes have been proposed. Among these, prime number labeling for DAGs [10] has attracted researchers because of its simplicity. First, each vertex v is assigned a unique prime number denoted as the self-label of v. The label of v is defined as the product of all its direct parents' labels and the self-label. Divisibility checking is used to determine the reachability between labels of any two vertices. Therefore, the performance of reachability query processing is dependent on the label size (i.e., how large the integer number is). Fortunately, the prime number labeling scheme is invariant to an order of uniquely assigning self-labels to each vertex. This means that the label size varies according to the self-label assignment orders.

Fig. 1 shows two existing methods for self-label assignment for the same DAG, where the self-labels correspond to the first number inside the rectangles. It can be seen that the label size by DSC (ordered by the number of unique descendants) [14] is much smaller than that by Topological Sort (TOP) [9]. The details are discussed in Section 4. Therefore, it is important to devise a method for determining the optimal order of assigning self-labels to vertices.

Various self-label assignment orders have been proposed, such as Bread-First Search (BFS), Depth-First Search (DFS), TOP [9], and DSC [14].

We observe that label size can be reduced further. We propose an approach to reduce the label size further in two ways. First, smaller self-labels need to be assigned to a vertex that has many ancestors. Second, larger self-labels need to be assigned to a vertex that has few descendants. Our approaches exploit these two aspects.

The rest of this paper is organized as follows. Related works are introduced in Section 2. Section 3 summarizes the prime number labeling scheme proposed in [9]. Previous labeling algorithms are explained in Section 4. The proposed approaches are explained in Section 5. The performance study is presented in Section 6. The paper is concluded in Section 7.

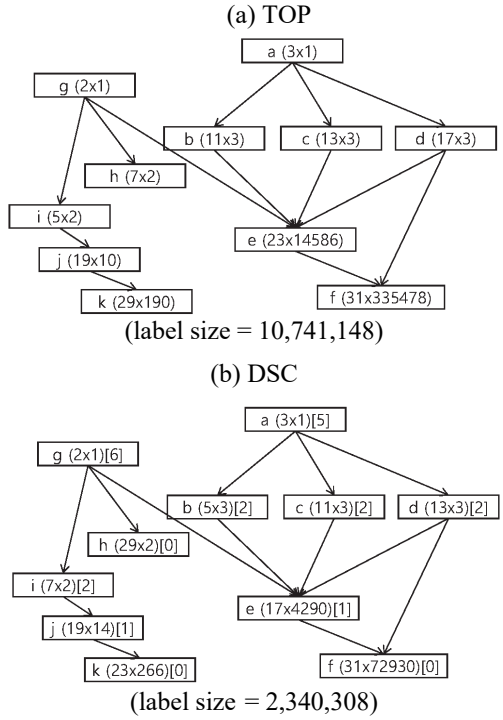

Fig. 1. For the same DAG, different label sizes are obtained by different methods for self-label assignments. (a) Labeled by TOP. The rectangle represents vertex $v(self(v) \times Lp(v))$. The notations are based on Definition 1 and Definition 2. (b) Labeled by DSC. The rectangle represents vertex $v(self(v) \times Lp(v))[weight]$, where $weight$ is the weight of v defined by a method; in the case of DSC, the weight is the number of unique descendant nodes of v.

2. RELATED WORK

Early works on labeling schemes have been conducted on tree data structure, which is a special case of graph data structure [2,3]. Existing tree labeling schemes can be classified by the definition of label, including interval-based, prefix-based, and prime number labeling scheme.

In the interval-based labeling scheme [1,4,7,11,12], each node is labeled with (start, end, level). "start/end" signifies an offset that covers all its descendants. "level" is the number of nodes up to the root. Reachability can be answered by checking whether an interval is included in the other interval.

In the prefix-based labeling scheme [5,6,8, 13], the label of each node is composed of a prefix and postfix label. The prefix label is the same as the parent node's label. The postfix label becomes the sibling order. Prefix matching is used to check the reachability of two nodes. In the prime number labeling scheme [9,10], the label of each node is an integer. A unique prime number is assigned to each node, which is called "self-label." The label of each node is the product of the self-label and the labels of all direct parents. Among these schemes, we focus on the prime number labeling scheme, which is summarized in Section 3.

[14] proposed a prime number labeling scheme for ontology. Several heuristics have been demonstrated for reducing label size in the context of ontology. [9] adapted the prime number labeling scheme to DAGs. They also proposed optimization techniques. TOP is used to sort nodes and assign self-labels to also reduce label size. Unfortunately, existing works that intend to reduce label size are still inefficient, which motivates us to devise more sophisticated approaches. We show why existing works are inappropriate in Sections 4 and 5.

3. PRIME NUMBER LABELING SCHEME

In this section, we summarize the prime number labeling scheme for DAGs proposed in [9]. DAG is a pair $G = (V, E)$ comprised of a set V of vertices and a set E of edges. An edge is an ordered pair of vertices. Each vertex is assigned a unique prime number called "self-label," as stated in Definition 1.

Definition 1. Self-label: $self(v)$ is a prime number for a vertex v that is unique in V. In other words, there exist no distinct vertices v_i and v_j such that $self(v_i) = self(v_j)$.

As stated in Definition 1, the only criterion for determining self-labels is uniqueness. This

means that a vertex can have any prime number be a self-label if the number has not been used by any other vertex. This motivates us to devise a novel method for assigning self-labels such that the label size is minimized, as explained in the Sections 4 and 5.

In the original version of the prime number labeling scheme [10], the label of each vertex is computed by multiplying all the direct parents' labels and the self-label. In [9], a more efficient scheme was proposed. The label of each vertex is computed by multiplying the least common multiple of all the direct parents' labels and the self-label, as stated in Definition 2.

Definition 2. Label: Label $L(v)$ for vertex v is an integer computed as follows:

$$L(v) = self(v) \cdot Lp(v)$$

$$Lp(v) = \begin{cases} 1 & \text{if } v \text{ has no parents} \\ lcm(L(p_1),\ldots,L(p_n)) & \text{for all direct parents } p_i \text{ of } v \end{cases}$$

, where $lcm(x_1, x_2, \ldots, x_n)$ is the least common multiple of n integers $x_1, x_2, \ldots, x_n$.

The reachability test of two given vertices can be performed by a division operation, as stated in Definition 3.

Definition 3. Reachability Test: Given two ordered vertices, s and d, we say that d is reachable from s if $L(d)$ is divisible by $L(s)$; otherwise, it is not reachable.

4. EXISTING LABELING ALGORITHMS

In this section, several existing prime number labeling algorithms are discussed. The prime number labeling of DAGs is formally stated in Definition 4.

Definition 4. Prime Number Labeling: Given DAG $G(V, E)$, a vertex label injective function $L : V \rightarrow \mathrm{N}$ is output, where N is a set of positive integers such that $L(v_i)$ is divisible by $L(v_j)$ if and only if there exists a path from v_j to v_i. In particular, we might want to minimize the label size of G defined as $\sum L(v_i)$.

Label size is closely related to the time for determining the reachability relationship between two vertices. The larger the labels that two vertices have, the slower it determines reachability. Thus, it is important to obtain as smaller a label size as possible. An observation is that the prime number labeling scheme does not require a particular vertex to have a certain self-label. Diverse label assignments are possible. Existing label assignment ordering includes BFS, DFS, TOP, and DSC.

In the approaches based on BFS and DFS, sequentially increasing prime numbers are assigned to each vertex in the order visited by these traversals. The common problem of DFS and BFS is that they cannot manage vertices with fewer ancestors. Imagine that we do a DFS on the DAG in Fig. 1a. Consider a vertex a that is at the top of the hierarchy. Assume that g has been visited first by a DFS. Then, a has to be visited last, and thereby be assigned a large self-label. It is trivial that this make the overall label size larger because there are many descendants from a.

To resolve the issue, TOP is utilized [9]. It first visits vertices with fewer ancestors. Fig. 1a shows that the self-labels of g and a are now 2 and 3, respectively. This allows obtaining a smaller label size than BFS and DFS. Unfortunately, TOP is inefficient when there is a vertex with few descendants and ancestors. For example, see vertex h in Fig. 1a, where $self(h) = 7$. Because h has no descendants, it would be better if a large prime number be assigned. This can be achieved by DSC, which considers the number of unique descendants [14]. In Fig. 1b, see $self(h) = 29$.

5. PROPOSED APPROACH

Our approaches consider the statistics of individual vertices in order to determine the label assignment order by which self-labels are assigned to each vertex. The algorithm is stated in Algorithm 1. The input is DAG G, and it returns a vertex label function L as described in Definition 4. An empty max-heap H is created to maintain a list of vertices sorted by its weight (line 1). It then iterates over vertices to calculate its weight and puts them into H (lines 2 to 5). The first prime number is 2 (line 6). Sequentially increasing prime numbers are assigned to each vertex in the order obtained from H (lines 7 to 11). Finally, the label of each vertex is computed according to Definition 2 (lines 12 to 14).

Several variations for the weight of a vertex (line 3 in Algorithm 1) are demonstrated below. All the weights are defined over a single vertex. We are motivated by the heuristics proposed in [14], such as leaf class with the most ancestors

and class with the most descendants. The difference is that they consider the number of *unique* ancestors/descendants, whereas our definition is based on the number of *accumulated* ancestors/descendants.

Algorithm 1. Prime Number Labeling Algorithm

Input: DAG $G(V, E)$

Output: Vertex label map L

Procedure:

1: $H \leftarrow$ empty max-heap
2: FOR EACH vertex v_i in V
3: compute the weight of v_i
4: put v_i to H
5: END FOR
6: $p \leftarrow 2$
7: WHILE H is not empty
8: $v_i \leftarrow$ obtain a vertex from H
9: $self(v_i) \leftarrow p$
10: $p \leftarrow$ the next prime number after p
11: END WHILE
12: FOR EACH vertex v_i in V
13: $L(v_i) \leftarrow$ according to Definition 2
14: END FOR
15: RETURN L

Our approach is more appropriate to the prime number labeling scheme described in Definition 2. It can be seen that all the ancestors' labels are recursively multiplied. For example, as shown in Fig. 1a, in order to compute $L(e)$, considering $L(a) = 3$ three times is required. Based on this rationale, we propose the following weights:

Definition 5. AAN: The number of *accumulated* ancestors $AAN(v)$ for vertex v is calculated recursively as follows:

$$AAN(v) = \begin{cases} 0 & \text{if } v \text{ has no parents} \\ \sum_{v_p \in Pa(v)} AAN(v_p) + 1 & \text{otherwise} \end{cases}$$

, where $Pa(v)$ is a set of direct parents of v.

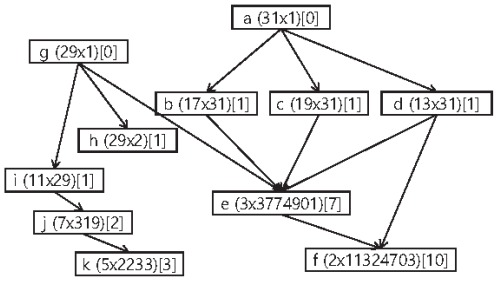

Fig. 2. ANN (label size = 33,990,072)

$AAN(v)$ reflects the number of times that $L(e)$ has to be multiplied in order to obtain $L(v)$, where $e \neq v$. According to Definition 2, making $L(v)$ smaller by assigning self-labels that are as small as possible (i.e., making $self(v)$ smaller) to v with many ancestors (i.e., $Lp(v)$ is likely to be large) is expected. Unfortunately, this method could produce a very large label size. Fig. 2 shows that vertices a and g have no ancestors, and therefore they are assigned the large self-labels 31 and 29. Such large self-labels are inherited by many descendants, thereby producing large labels overall. Indeed, compared with Fig. 1, the label size by AAN is even larger than previous works. This does not mean that AAN is useless. Experiment results show that AAN is useful when combined with other metrics (i.e., ADS), which will be clear in Section 6.

Definition 6. ADS: The number of *accumulated* descendants $ADS(v)$ of a vertex v is calculated recursively as follows:

$$ADS(v) = \begin{cases} 0 & \text{if } v \text{ has no children} \\ \sum_{v_c \in Ch(v)} ADS(v_c) + 1 & \text{otherwise} \end{cases}$$

, where $Ch(v)$ is a set of direct children of v.

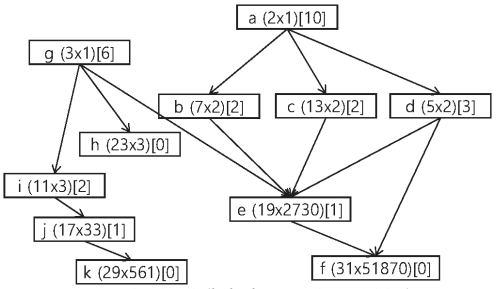

Fig. 3. ADS (label size=1,676,827)

$ADS(v)$ reflects the number of times that $L(v)$ is used to compute the other vertices' label. ADS is intended to be used such that self-labels that are as small as possible are assigned to vertices

with many descendants. By doing so, it can be expected for vertices with less descendants to have large self-labels, which means that most large self-labels are inherited by fewer vertices. Fig. 3 shows that vertex a has the largest number of descendants, which is 10. This means that $L(a)$ is the label used the most by the other vertices. We can reduce the label size by assigning the smallest self-label to a. On the other hand, we have $self(j) = 17$, which is a large self-label. This is because $L(j)$ is used only once to compute $L(k)$.

Using ADS, we can reduce the label size dramatically. However, we observe that ADS is not optimal in some cases. For example, $L(h)$ has one ancestor and $L(f)$ has several ancestors while both have no descendant. The observation leads us to devise AAN_ADS.

Definition 7. AAN_ADS: A linear combination of AAN and ADS, denoted as $AAN\_ADS(v)$, is defined as $(1-\alpha)\cdot AAN(v) + \alpha \cdot ADS(v)$, such that $0 < \alpha < 1$

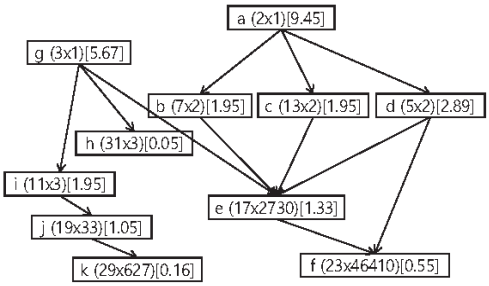

Fig. 4. AAN_ADS(label size = 1,132,831)

Fig. 4 is obtained by AAN_ADS when $\alpha = 0.946$. The difference from ADS is the assignment against vertices f and h. The largest self-label (31) is assigned to f in the case of ADS, whereas it is assigned to h in AAN_ADS. Because f and h have the same descendants, ADS cannot determine the order of f and h. Note that f has more ancestors than h, which leads us to expect that $Lp(f) > Lp(h)$. In this case, the reduced label size can be achieved by assigning larger self-labels to h instead of f. This observation leads us to consider both AAN and ADS in one hand.

6. PERFORMANCE EVALUATION

6.1 Experiment Settings

The proposed approaches are evaluated through a comparison with several existing labeling algorithms. The existing approaches include TOP [9] and DSC (ordered by the number of unique descendants) [14]. Our approaches include ADS and AAN_ADS. Results of AAN are omitted here because the label size by AAN was too large to be compared with directly other approaches (e.g. the label size in Fig. 2 and Fig 3.). We compare these systems in terms of label size and labeling time. The labeling time is averaged over five runs. In these experiments, Java was used to implement all the systems. All the experiments were conducted on a machine with 2.4 GHz CPU and 40 GB RAM.

6.2 Real-world Datasets

The real-world graph datasets were downloaded from diverse sources. Cycles were removed to obtain DAG datasets by condensing vertices in cycles into a single vertex. The dataset statistics are listed in Table 1 in Appendix. The diameter is the average number of edges from a vertex with no incoming edges to another vertex with no outgoing edges. The degree of a vertex indicates the number of incoming and outgoing edges. The top-vertices ratio is the proportion of vertices with no ancestors to the entire vertices.

6.3 Performance on Real-world Datasets

Table 2 lists the label sizes by each approach. The label size is represented as the logarithm of the label size to base 2. The label size by AAN_ADS is obtained by running several trials with varying α. Approximately 90 values ranging from 0.0 to 1.0 were tried in the experiments. In this setting, AAN_ADS could achieve the smallest label size against all the datasets.

Table 2. Label sizes (base-2 logarithm)

	TOP	DSC	ADS	AAN_ADS
D1	286	211	211	**205**
D2	277	252	252	**250**
D3	141	138	138	**136**
D4	530	426	426	**425**
D5	80	34	34	**28**
D6	1608	1229	1230	**1225**
D7	475	322	322	**321**
D8	2318	1561	1553	**1550**
D9	444	376	376	**372**

D10	1516	1098	1097	**1088**

The best performance by ADS and AAN_ADS with respect to TOP is obtained against D8: -765 and -768, respectively. Overall, D8 has large max. ancestors, max. diameters, and max. degree. In such a complex DAG, TOP fails to measure the number of descendant vertices affected by a vertex. For example, as shown in Fig. 1a, TOP wastes small self-labels on vertices with no descendants (vertex h in this case). Similarly, DSC is inefficient because it cannot measure the number of ancestors that exist. When DAG has a complex structure, there can be many cases similar to this situation. The experiments on synthetic datasets described in the next section provide the details for how graph structure affects label size.

Label sizes by varying α of AAN_ADS are depicted in Fig. 5 and 6. It can be seen that the label size is reduced to some degree by increasing the value. This does not necessarily indicate that the label size is minimized when α is equal to 1.0. Rather, the label size is minimized when the value is less than 1.0 and larger than 0.9. According to Definition 7, increasing α gives more weight to ADS. This can be understood in the same context as the results listed in in Table 2, where ADS is slightly worse than AAN_ADS and the worst is AAN (omitted here). The nature of the prime number labeling scheme, where the numbers are inherited by descendants in one direction, explains why ADS is a more significant feature for reducing label size.

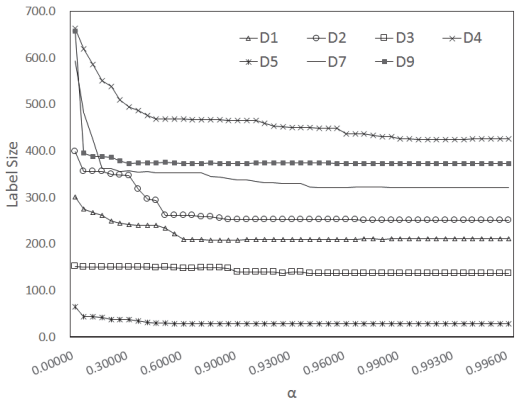

Fig. 5. Label sizes (base-2 logarithm) of small datasets by AAN_ADS with varying α. Values greater than 0.996 are omitted to make the graphs clear.

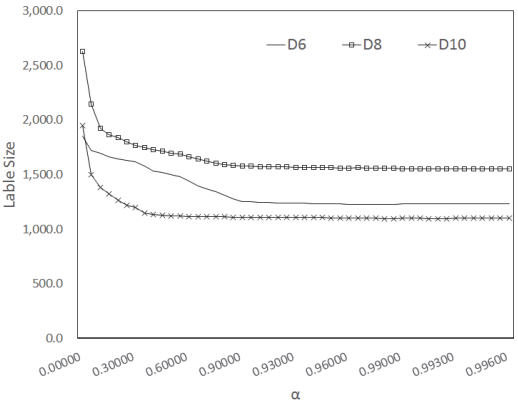

Fig. 6. Label sizes (base-2 logarithm) of large datasets by AAN_ADS with varying α. Values greater than 0.996 are omitted to make the graphs clear.

The labeling time is listed in Table 3. Our approaches (ADS and AAN_ADS) show the best performance for all datasets. The reduced label size accounts for the labeling time. A smaller label size indicates that smaller numbers have been multiplied to compute the labels of each vertex. Our approaches help manage smaller numbers, thereby decreasing labeling time.

Table 3. Labeling time in ms. for real-world datasets

	TOP	DSC	ADS	AAN_ADS
D1	0.8	0.8	0.6	**0.5**
D2	1.6	1.6	**1.0**	1.1
D3	2.4	2.0	**1.4**	1.6
D4	13.0	8.6	7.0	**6.8**
D5	502.8	**252.6**	252.8	252.9
D6	171.6	108.2	107.4	**107.0**
D7	236.2	145.6	**142.6**	143.5
D8	2,901.4	2,148.0	**2,133.6**	2,138.1
D9	5,933.0	3,892.0	3,903.4	**3,872.8**
D10	6,892.6	4,879.2	**4,776.8**	4,780.9

6.4 Performance on Synthetic Datasets

We generated two groups of synthetic DAG datasets grouped by the number of vertices, each of which contains 43 datasets. Using the datasets, we are able to evaluate the proposed approach in diverse settings that reflect real-world diversity. The overall statistics are listed in Table 4.

Table 4. Synthetic DAG datasets

ID	Vertices	Edges	Avg. Ancestors	Avg. diameter	Max degree
S1	912 ~ 1,000	1,264 ~ 4,950	0.4 ~ 5.5	2.4 ~ 4.6	2.7 ~ 9.9
S2	9,115 ~ 10,000	12,179 ~ 49,050	0.39 ~ 5.0	2.4 ~ 6.4	2.6 ~ 9.8

The labeling time for S1 and S2 are shown in Fig. 7 and 8. The labeling time for AAN_ADS is selected from the experiments when the best value of α is chosen. Overall, ADS and AAN_ADS are better. DSC is slow because it requires counting unique descendants, whereas ADS simply counts descendants without determining the vertices that have been counted. Because the labeling task is a preprocessing step not often performed, we would like to focus more on label size. We argue that even if the proposed approach is little bit slow, it is accepted that if we can take much advantage of label size.

Fig. 9 in Appendix depicts the label size reduced by AAN_ADS with respect to ADS. This is obtained by subtracting label sizes by AAN_ADS with ADS. We take the base-2 logarithm to be plotted in the graphs. In all cases, AAN_ADS outperforms ADS. Overall, the more average diameter and degree these datasets have, the more label sizes AAN_ADS can reduce. The reason is that complex graph structures can easily be formed by the existence of many edges, where AAN_ADS is expected to work better. When the average diameter is large (larger than 4.3 and 6.2 for S1 and S2, respectively), the performance of AAN_ADS no longer increases. Changing the self-label assignment order has slight influence on reducing label size when there are too many vertices between two vertices. In a sense, because it is unavoidable for the label size to become very large when the diameter or degree is large, reducing the label size in logarithmic scale is difficult.

7. CONCLUSION

In order to improve the performance of reachability query processing against prime number labelled DAGs, it is crucial to reduce the label size. To address the issue, we propose vertex ordering techniques for the prime number labeling of DAGs in order to reduce label size. Diverse statistics of vertices were combined to sort vertices and then assign self-labels in the order. The proposed approaches were evaluated against both real-world and synthetic DAG datasets and compared with existing labeling algorithms. The experiment results showed that label size was dramatically reduced and labeling time was also reduced. We also found that our approaches are more suitable for complex DAGs. The limitation of the proposed approach is that the labeling algorithm cannot deal with massive DAGs if not loaded into memory. In future works, we plan to devise a MapReduce framework-based algorithm for managing large DAGs.

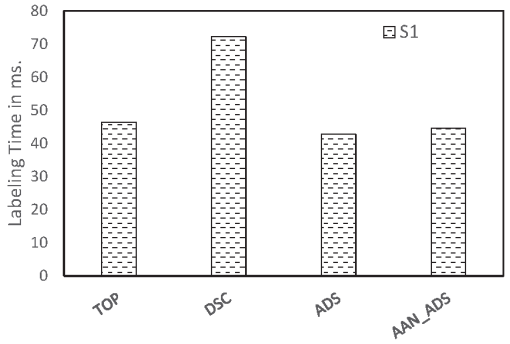

Fig. 7. Labeling time in ms. for synthetic datasets (S1)

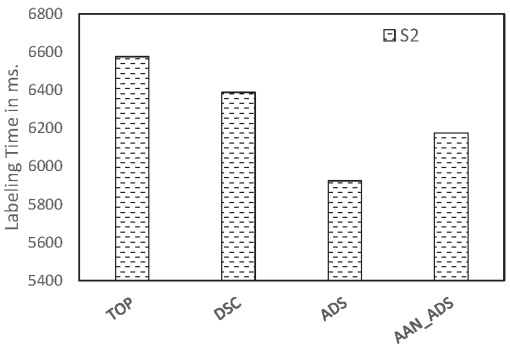

Fig. 8. Labeling time in ms. for synthetic datasets (S2)

ACKNOWLEDGEMENT

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. R0101-16-0054, WiseKB: Big data based self-evolving knowledge base and reasoning platform) and ETRI R&D Program ("Development of Big Data Platform for Dual Mode Batch-Query Analytics, 16ZS1410") funded by the Government of Korea.

REFRENCES:

[1] Christophides, V., Karvounarakis, G., Plexousakis, D., Scholl, M., Tourtounis, S.: Optimizing taxonomic semantic web queries using labeling schemes. Web Semantics: Science, Services and Agents on the World Wide Web 1(2), 207-228 (2004)

[2] Clark, J., DeRose, S., et al.: XML path language (xpath), 1999 (2004)

[3] Klaib, A., Joan, L.: Investigation into indexing XML data techniques (2014)

[4] Li, C., Ling, T.W.: QED: a novel quaternary encoding to completely avoid re-labeling in XML updates. In: CIKM. ACM (2005)

[5] Lin, R.R., Chang, Y.H., Chao, K.M.: A compact and efficient labeling scheme for XML documents. In: Database Systems for Advanced Applications, pp. 269-283. Springer (2013)

[6] Lu, J., Meng, X., Ling, T.W.: Indexing and querying xml using extended Dewey labeling scheme. Data & Knowledge Engineering 70(1), 35-59 (2011)

[7] Subramaniam, S., Haw, S.C., Soon, L.K.: Relab: A subtree based labeling scheme for efficient xml query processing. In: Telecommunication Technologies (ISTT), 2014 IEEE 2nd International Symposium on, pp. 121-125. IEEE (2014)

[8] 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, pp. 204-215. ACM (2002)

[9] Wu, G., Zhang, K., Liu, C., Li, J.: Adapting prime number labeling scheme for directed acyclic graphs. In: Database Systems for Advanced Applications, pp. 787-796. Springer (2006)

[10] Wu, X., Lee, M.L., Hsu, W.: A prime number labeling scheme for dynamic ordered XML trees. In: ICDE (2004)

[11] Xu, L., Bao, Z., Ling, T.W.: A dynamic labeling scheme using vectors. In: Database and Expert Systems Applications. Springer (2007)

[12] Xu, L., Ling, T.W., Wu, H.: Labeling dynamic XML documents: an order-centric approach. Knowledge and Data Engineering, IEEE Transactions on 24(1), 100-113 (2012)

[13] Xu, L., Ling, T.W., Wu, H., Bao, Z.: DDE: from Dewey to a fully dynamic XML labeling scheme. In: SIGMOD. ACM (2009)

[14] Preuveneers, D., Berbers, Y.: Prime numbers considered useful: ontology encoding for efficient subsumption testing. Tech. Rep. CW464. Department of Computer Science, Katholieke Universiteit Leuven, Belgium (October 2006).

APPENDIX

Table 1 DAG datasets

ID	Name	Vertices	Edges	Max. Diameter (avg.)	Max. Degree (avg.)	Top-Vertices ratio
D1	BPMN ontology[1]	183	338	8 (3.1)	59 (3.7)	27.3%
D2	Wildlife ontology[2]	242	435	3 (1.9)	85 (3.6)	73.1%
D3	geospecies[3]	459	735	4 (1.6)	118 (3.2)	47.4%
D4	Taxon concept[4]	746	1,477	7 (1.9)	253 (4.0)	53.2%
D5	Wikidata property[5]	2,076	2,183	2 (2.0)	2,042 (2.1)	0.2%
D6	copyrighttermbank[6]	3,582	7,018	4 (1.6)	774 (3.9)	64.5%
D7	British geological survey geochronology[7]	3,820	8,494	4 (2.0)	722 (4.4)	70.5%
D8	DBPedia[8]	18,603	28,739	9 (3.2)	2,681 (3.1)	80.9%
D9	Genage models[9]	22,119	40,143	3 (1.7)	3,657 (3.6)	83.5%
D10	lifescience[10]	29,250	48,832	8 (1.8)	2,614 (3.3)	87.7%

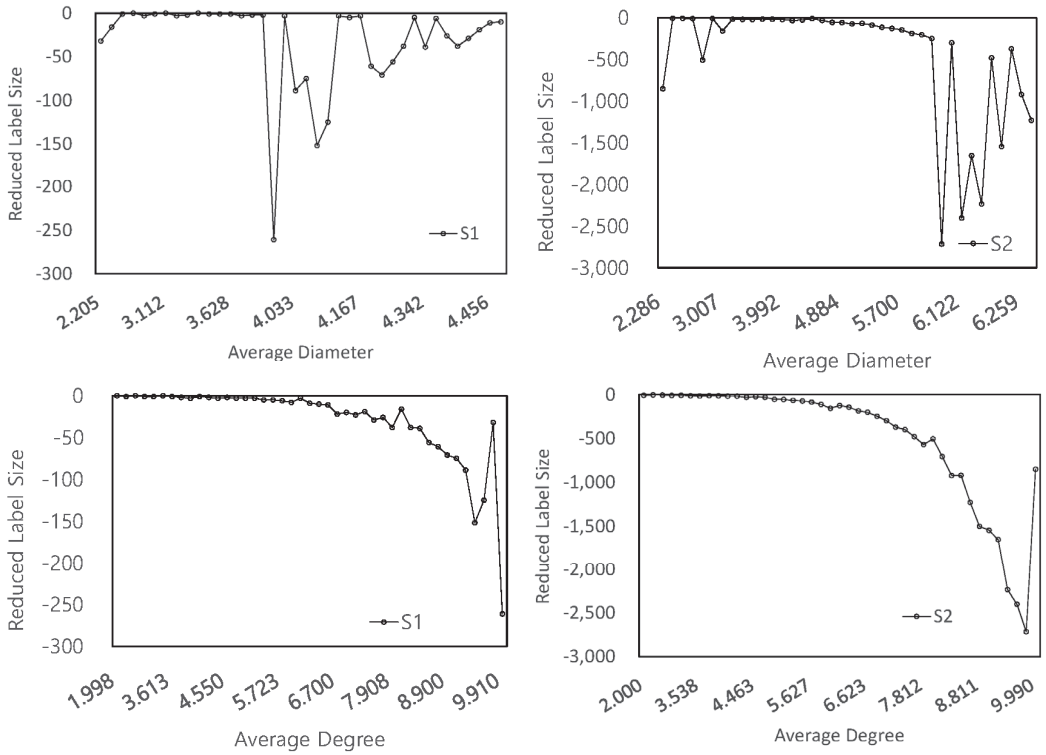

Fig. 9. Label size of AAN_ADS relative to ADS against synthetic datasets

[1] https://dkm.fbk.eu/bpmn-ontology
[2] http://www.bbc.co.uk/ontologies/wo
[3] https://bioportal.bioontology.org/ontologies/GEOSPECIES
[4] http://www.taxonconcept.org/ontologies/
[5] http://tools.wmflabs.org/wikidata-exports/
[6] https://datahub.io/dataset/copyrighttermbank
[7] http://data.bgs.ac.uk/downloads
[8] http://wiki.dbpedia.org/services-resources/ontology
[9] http://download.openbiocloud.org/release/3/genage/
[10] http://download.openbiocloud.org/release/3/lsr/