Optimization Techniques for 2-hop Labeling of Dynamic Directed Acyclic Graphs

Jinhyun Ahn

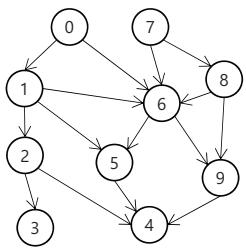
Biomedical Knowledge Engineering Laboratory and Dental Research Institute
Seoul National University
jhahncs@snu.ac.kr

Abstract. Graph databases are increasingly used for RDF(Resource Description Framework) data management. Mining reachability relationships between resources is one of important building blocks for graph databases. Reachability relationships in a graph and its corresponding directed acyclic graph(DAG) are equivalent when we focus on reachability alone, which allows to focus on DAGs in this thesis. Diverse labeling schemes have been proposed to efficiently determine the reachability of DAGs. We focus on a state-of-the-art 2-hop labeling scheme that is based on a permutation of vertices to achieve a linear index size and reduce on-line searches that are required when the reachability cannot be answered by 2-hop labels only. We observed that the approach can be improved in three different ways; 1) *space-efficiency* —guarantee the minimized index size without randomness 2) *update-efficiency* —update labels efficiently when graphs changes 3) *parallelization* —labeling should be cluster-based, and solved in a distributed fashion. In these regards, this PhD thesis proposes optimization techniques that address these issues. In this paper in particular, a way of reducing the 2-hop label size is proposed with preliminary results on real-world DAG datasets. In addition, we will discuss the feasibilities of the other issues based on our on-going works.

1 Problem statement

A 2-hop labeling scheme of a DAG is to label each vertex v with a pair ($L_{out}(v)$, $L_{in}(v)$), where $L_{out}(v)$ is a set of vertices that v can reach and $L_{in}(v)$ is a set of vertices reachable from v [4]. In this thesis, we focus on a state-of-the-art variation of 2-hop labeling [14], where a permutation of vertices is used to allow $L_{out}(v)$ and $L_{in}(v)$ to keep at most k reachable vertices, which probabilistically guarantees reduction in on-line Depth-First-Search(DFS), a condition required when the reachability cannot be answered by these labels only. We briefly review the labeling scheme as follows.

2-hop Labeling based on Independent Permutation (IP) Let $G(V, E)$ be a DAG with V as a set of vertices and E as a set of edges pairing two vertices. $u \rightsquigarrow v$ is used to denote that u can reach v, and $u \not\rightsquigarrow v$, otherwise. A permutation of V is a bijection denoted as $\sigma : V \longrightarrow V$. Given G and a positive integer k, IP



v	$\sigma_1(v)$	$L_{out}(v)$	$L_{in}(v)$
0	7	0,2,3	7
1	4	0,2,3	4,7
2	9	3,8,9	4,7,9
3	3	3	3,4,7
4	8	8	0,1,2
5	0	0,8	0,1,4
6	5	0,2,5	1,4,5
7	6	0,1,2	6
8	1	0,1,2	1,6
9	2	2,8	1,2,4

$size\left(I^3_{\sigma_1}(V)\right) = 162$

v	$\sigma_2(v)$	$L_{out}(v)$	$L_{in}(v)$
0	0	0,1,2	0
1	7	1,2,3	0,7
2	1	1,2,8	0,1,7
3	8	8	0,1,7
4	2	2	0,1,2
5	6	2,6	0,3,5
6	3	2,3,4	0,3,5
7	5	2,3,4	5
8	9	2,3,4	5,9
9	4	2,4	0,3,4

$size\left(I^3_{\sigma_2}(V)\right) = 139$

Fig. 1: The 2-hop index size of the same graph varies according to σ_1^3 and σ_2^3.

randomly generates σ and then outputs the 2-hop index $I^k_\sigma : V \longrightarrow H$, where H is a set of pairs $(L_{out}(v), L_{in}(v))$ defined as:

$$L_{out}(v) = min_k\{\sigma(u) | v \rightsquigarrow u\}$$

$$L_{in}(v) = min_k\{\sigma(u) | u \rightsquigarrow v\}$$

, where $min_k\{A\}$ is a sub-set of A, such that $A_i < A_j$ if $i < j$ and $|min_k\{A\}| \leq k$. In other words, $min_k\{A\}$ contains the k smallest integers in A.

In the original version of IP, σ is randomly generated using the Knuth shuffle algorithm. We argue that the randomized way of generating a permutation is unable to ensure that the minimized size of index is obtained. Therefore, we define the first problem that generates a *space-efficient* permutation defined in Definition 1.

Definition 1. Space-efficient Permutation Given $G(V, E)$ and a positive integer k, output σ such that

$$\underset{\sigma}{\text{argmin}}\ size(I^k_\sigma(V)) := \frac{1}{|V|} \sum_{v \in V} space(L_{out}(v)) + space(L_{in}(v))$$

, where $space(L)$ is the space requirements for a set L of integers, which can be defined according to the application requirements.

An illustrative example that shows the effect of σ on the index size is depicted in Fig. 1, where we assume $space(L) = \sum_{w \in L} w$ to see the index size difference for the very small DAG.

Linked Open Data[1](LOD) evolves over time [9]. When a DAG G_t changes into G_{t+1}, the straightforward way is to construct the 2-hop index against G_{t+1} from scratch. As this is not feasible for massive DAGs in LOD, we define a second problem that generates an *update-efficient* permutation by which 1) a smaller number of existing labels are modified and 2) a smaller index size is maintained, as defined in Definition 2. Since there is a trade-off between these criteria, weights

[1] http://linkeddata.org

are used according to requirements. The first factor of the equation in Definition 2 represents the number of vertices in the current version V_{v+1} whose labels are not the same as in the previous version in $I^k_\sigma(v)$.

Definition 2. Update-efficient Permutation Assume we already have $G_t(V_t, E_t)$ and its 2-hop index $I^k_\sigma(V_t)$. Given $G_{t+1}(V_{t+1}, E_{t+1})$, generate a new permutation σ_{t+1} such that

$$\underset{\sigma_{t+1}}{\operatorname{argmin}}\ \alpha \times |\{v \in V_t \bigcap V_{t+1} | I^k_{\sigma_t}(v) \neq I^k_{\sigma_{t+1}}(v)\}| + \beta \times size(I^k_{\sigma_{t+1}}(V_{t+1}))$$

, where $\alpha \geq 0$, $\beta \geq 0$.

Meanwhile, existing 2-hop labeling algorithms running on a single machine are not efficient enough to address the above problems for massive DAGs. Our third problem is thus to devise a *cluster-based* 2-hop labeling algorithm that supports Definition 1 and 2.

2 Relevancy

Reachability relationships in a graph and its corresponding DAG are equivalent, since a graph can be transformed into a DAG by condensing every strongly connected component(SCC) into a single vertex and retaining edges between vertices in SCC and the other vertices that are connected to SCC [13]. In terms of reachability relationships, LOD can be viewed as a real-world DAG dataset. A vast amount of knowledge is available in LOD, including social networks, biological networks, traffic networks, and software version management. Protein-protein interaction networks, for example, can be represented in DAGs. One may be interested in mining interactions (i.e., reachability relationships) between two proteins (i.e., vertices) [5]. Regarding RDF triple stores, reachability relationship identification helps process SPARQL queries efficiently [6].

3 Related Work

There are extensive studies on labeling of DAGs for reachability queries processing [17]. The straightforward way is to pre-compute the edge transitive closures(TC) [11]. Even if this method provides an answer to a reachability query in $O(1)$ time, the index could be huge for even small dense graphs. On-line traversal, such as DFS and Bread-First Search(BFS), do not require any index but lead to slow query processing. Various approaches have been made to address trade-offs between the index size and the speed of query processing. Some authors have adapted the prime number labeling scheme to DAGs [15]. However, prime numbers quickly grow to large values. Tree Cover [1] labels a vertex v with a compressed TC of the subtree rooted at v. [12] present GRIPP(GRaph Indexing based on Pre- and Postorder numbering) that utilizes spanning trees

based on an interval label scheme. GRAIL [16] is based on randomized multiple interval labeling. FERRARI [10] labels each vertex with a mixture of exact and approximate reachability intervals, where subsets of intervals are merged into approximate intervals. 2-hop labeling [4] labels each vertex v with a pair $(L_{out}(v), L_{in}(v))$.

Recently, a state-of-the-art variation of 2-hop labeling has been proposed in [14] that reports outstanding performance compared to existing approaches. A permutation of vertices is first generated randomly. MinHash is adapted to construct 2-hop labels; at most k reachable vertices are only maintained in 2-hop labels. For pairs that cannot be determined by k reachable vertices only, an on-line search is performed. Some heuristics are also presented to minimize the case that requires exhaustive on-line DFS searches.

4 Research questions

The research objective is to optimize the IP labeling [14] in terms of index size, updating cost, and scalability. We want to answer the following research questions (where difficulties regarding the questions are also discussed).

1) Is it possible to deterministically choose a permutation σ of V that best minimizes the index size? Because the number of candidate permutations is $n!$, a brute-force algorithm is not feasible. One may choose a permutation by sorting V using a topological sort. Smaller numbers are assigned to vertices having more reachable vertices. It can be expected that this method allows $L_{in}(v)$ to have smaller numbers. However, conversely, $L_{out}(v)$ would have large numbers, offsetting smaller size of $L_{in}(v)$. A more sophisticated technique is required.

2) When a new vertex v_{new} is added to a DAG $G_t(V_t, E_t)$ that has already been labeled by σ_t, which number should be assigned to $\sigma_{t+1}(v_{new})$? The simplest way is to make $\sigma_{t+1}(v_{new}) = max(\sigma_t(V_t)) + 1$. However, with this approach, the minimized index size is not always ensured.

3) In order to do 2-hop labeling in a cluster, when vertices are distributed, how can we know reachable vertices from a vertex? Given a set M of machines, we may distribute a subset V_i of V to a machine M_i and then perform labeling independently. The challenge is to obtain $L_{in}(v)$ and $L_{out}(v)$ for a vertex $v \in V_i$ residing in M_i whereas some $o \in L_{in}(v) \bigcup L_{out}(v)$ have been distributed to the other machines M_j, such that $i \neq j$.

5 Hypotheses

Several hypotheses regarding research questions are stated as follows.

- If smaller numbers are assigned to $\sigma(v)$ because v has more edges, then the 2-hop index size is reduced.

- The reachability query processing performance increases as index size is reduced.

Table 1: Real-world DAG datasets

Name	Vertices	Edges	Average Degree (max.)	Median Degree
arxiv	6,000	66,707	22.2 (700)	14
go	6,793	13,361	3.9 (71)	3
pubmed	9,000	40,028	8.9 (432)	4
citeseer	340,945	312,282	1.8 (55,758)	1
citpatents	3,774,768	16,518,947	8.8 (793)	6
go-uniprot	6,967,383	34,769,339	10.0 (1,186,280)	4
yago	16,375,031	25,908,132	3.2 (732,895)	1

– If 2-hop labeling is carried out in a cluster, then the index construction and update would be faster than existing single machine based algorithms are.

6 Approach

To prove the first hypotheses, we first compute the degree of each of the vertices in V and then sort V in decreasing order by degree. The order is regarded as the desired permutation $\sigma(V)$ as follows.

$$\sigma(v) = i$$

, where v is the i-th element in V^{degree} such that V^{degree} is a sorted set of V, sorted by decreasing degree of v.

In other words, the more degrees v has, the smaller the number assigned to $\sigma(v)$. This is derived by an expectation that if v has many edges, v is likely to become a reachable vertex from another vertex. The overall 2-hop index size could be reduced by making $\sigma(v)$ smaller. Note that the current approach does not take into account the parameter k. We plan to introduce the parameter k to guarantee a reduced index size for arbitrary k.

Regarding a cluster-based algorithm, we are motivated by the approach in [3] that proposed cluster based labeling algorithms for trees. In that work, Mapper performs incomplete labeling and then Reducer completes the labels by referring to the offset table shared by all machine. The offset table is constructed based on information collected from each Mapper, which contains information needed to complete the labels in Reducers. We plan to adapt the idea in a manner that effectively deals with large DAGs in a cluster.

7 Preliminary results

We implemented the first proposed approach based on the source codes provided by the authors of IP[2]. The proposed approach (denoted as IP_adv) is evaluated compared with the original approach (IP) and baseline (IP_fix). IP generates a

[2] http://www1.se.cuhk.edu.hk/~hwei/source/IP.zip

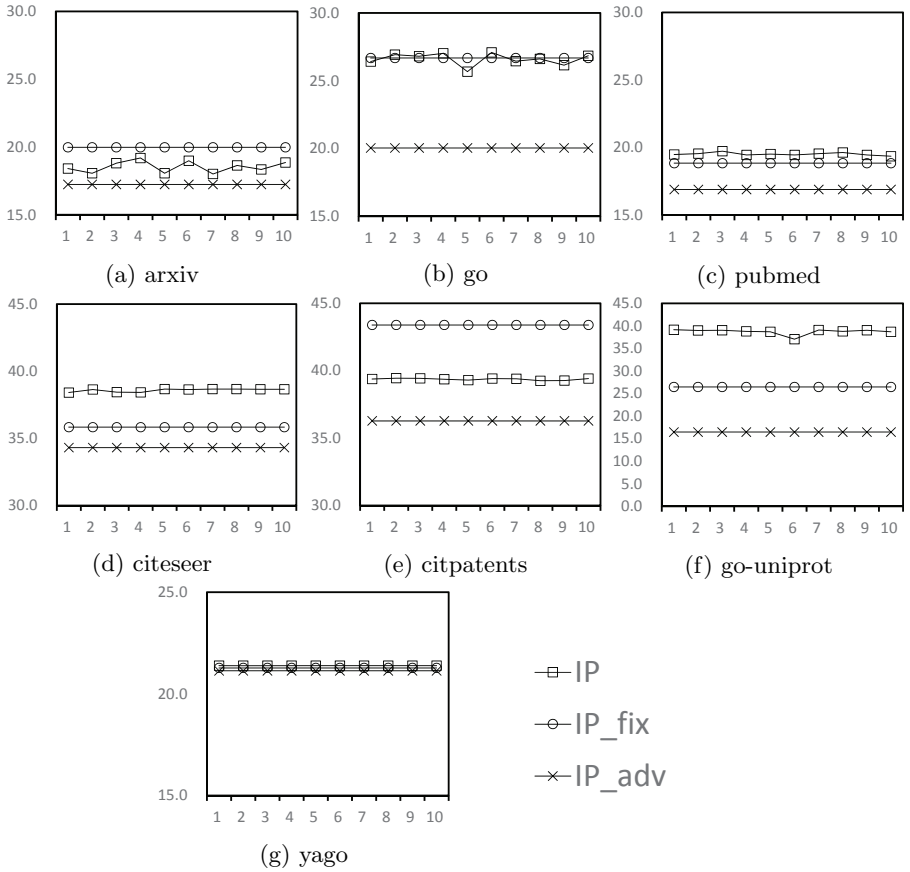


Fig. 2: 2-hop index sizes over 10 runs. The Y-axis indicates the index size, where the space requirements is $space(L) = \sum_{w \in L} len(w)$ such that $len(w)$ is the number of digits in w. The X-axis indicates runs. k is set to 5.

random permutation, which means that a different index is constructed for each run. IP_fix is based on the identity permutation such that $\sigma(v) = j$, where v is the j-th element in V. All the experiments were conducted on a machine with a 2.4 GHz CPU and 40 GB of RAM. We downloaded the real-world DAG datasets[3]. Statistics of the datasets are listed in Table 1.

The index sizes for each dataset are plotted in Fig. 2. The index size by IP_adv and IP_fix are compared with the index sizes generated by 10 runs of IP. For each run, only IP showed the different index size due to its random nature. IP_adv showed the best performance for all datasets. For small datasets with large degrees such as `arxiv` and `pubmed`, relatively small improvements are observed. For large datasets, we observed relatively large improvements, except

[3] `https://code.google.com/archive/p/ferrari-index/downloads`

for citeseer and yago. It can be seen in Table 1 that these datasets have relatively small median degrees. Since IP_adv is based on degrees, too small or too large degrees would not significantly contribute to a reduction in the index size. In future works, we plan to utilize more graph metrics to work better against such datasets.

8 Evaluation plan

We plan to collect LOD datasets and then transform them into DAGs by condensing SCC. Synthetic DAG datasets will also be considered, varying graph metrics such as in/out-degree, diameter, and vertices with no ancestors or descendants. In particular, several releases of DBpedia datasets[4] will be collected in order to evaluate the performance of updating the index. To examine the pros and cons of our approach in greater details, we will compare it to a number of notable existing approaches such as [10, 16, 18]. Regarding our cluster-based algorithm, as there exist few works on cluster-based 2-hop labeling, we will modify existing cluster-based tree labeling algorithms [3] and consider these as the baseline.

9 Reflections

It remains a challenge to apply graph reachability indexing techniques to very large sets of LOD. In this paper, we showed that simple graph metrics can be exploited to reduce the index size. By improving the proposed approach further, it will become more applicable to LOD. Another characteristic of LOD is that it changes over time. We have some experience on scalable RDF change detection [2, 8, 7] that outputs added and removed triples for two given RDF dataset versions. By utilizing this system, it would be possible to update only a portion of an existing 2-hop index while avoiding re-labeling. To perform these algorithms efficiently in a scalable manner, we can utilize distributed computing frameworks such as MapReduce and Spark.

Acknowledgement

I would like to acknowledge my advisors, Hong-Gee Kim (Seoul National University) and Dong-Hyuk Im (Hoseo University). 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).

[4] http://wiki.dbpedia.org/services-resources/datasets/previous-releases

References

1. Agrawal, R., Borgida, A., Jagadish, H.V.: Efficient management of transitive relationships in large data and knowledge bases, vol. 18. ACM (1989)
2. Ahn, J., Im, D.H., Eom, J.H., Zong, N., Kim, H.G.: G-Diff: A Grouping Algorithm for RDF Change Detection on MapReduce. In: Semantic Technology, pp. 230–235. Springer (2014)
3. Choi, H., Lee, K.H., Lee, Y.J.: Parallel labeling of massive xml data with mapreduce. The Journal of Supercomputing 67(2), 408–437 (2014)
4. Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queries via 2-hop labels. SIAM Journal on Computing 32(5), 1338–1355 (2003)
5. Gabr, H., Kahveci, T.: Signal reachability facilitates characterization of probabilistic signaling networks. BMC Bioinformatics 16(Suppl 17), S6 (2015)
6. Gubichev, A., Bedathur, S.J., Seufert, S.: Sparqling kleene: fast property paths in rdf-3x. In: First International Workshop on Graph Data Management Experiences and Systems. p. 14. ACM (2013)
7. Im, D.H., Lee, S.W., Kim, H.J.: Backward inference and pruning for RDF change detection using RDBMS. Journal of Information Science pp. 238–255 (2012)
8. Im, D.H., Lee, S.W., Kim, H.J.: A version management framework for RDF triple stores. International Journal of Software Engineering and Knowledge Engineering 22(01), 85–106 (2012)
9. Käfer, T., Umbrich, J., Hogan, A., Polleres, A.: Towards a dynamic linked data observatory. LDOW at WWW (2012)
10. Seufert, S., Anand, A., Bedathur, S., Weikum, G.: Ferrari: Flexible and efficient reachability range assignment for graph indexing. In: Data Engineering (ICDE), 2013 IEEE 29th International Conference on. pp. 1009–1020. IEEE (2013)
11. Simon, K.: An improved algorithm for transitive closure on acyclic digraphs. Theoretical Computer Science 58(1), 325–346 (1988)
12. Trißl, S., Leser, U.: Fast and practical indexing and querying of very large graphs. In: Proceedings of the 2007 ACM SIGMOD international conference on Management of data. pp. 845–856. ACM (2007)
13. Veloso, R.R., Cerf, L., Meira Jr, W., Zaki, M.J.: Reachability queries in very large graphs: A fast refined online search approach. In: EDBT. pp. 511–522. Citeseer (2014)
14. Wei, H., Yu, J.X., Lu, C., Jin, R.: Reachability querying: An independent permutation labeling approach. Proceedings of the VLDB Endowment 7(12), 1191–1202 (2014)
15. 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)
16. Yıldırım, H., Chaoji, V., Zaki, M.J.: GRAIL: a scalable index for reachability queries in very large graphs. The VLDB JournalThe International Journal on Very Large Data Bases 21(4), 509–534 (2012)
17. Yu, J.X., Cheng, J.: Graph reachability queries: A survey. In: Managing and Mining Graph Data, pp. 181–215. Springer (2010)
18. Zhu, A.D., Lin, W., Wang, S., Xiao, X.: Reachability queries on large dynamic graphs: a total order approach. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data. pp. 1323–1334. ACM (2014)