Recurrent Path Index for Efficient Graph Traversal

1st Kazuma Kusu
· *Graduate School of Culture and Information Science, Doshisha University*, Kyoto, Japan
· *Research Fellow of the Japan Society for the Promotion of Science*
ORCID: 0000-0002-3379-8720

2nd Kenji Hatano
· *Faculty of Culture and Information Science Doshisha University*, Kyoto, Japan
ORCID: 0000-0001-7079-8247

***Abstract*—Graph databases (GDB) enable us to conduct a query for searching and analyzing graph data efficiently. However, such a query has to extract sub-graphs in the beginning, so this process is high cost due to the NP-complete problem. GDBs find out sub-graphs specified in a query by graph traversal that is a process following edges from a node. Moreover, it enables them to traverse an edge at a constant cost, but graph traversal involving some edges is affected by database volume due to the increase of candidate that it has to traverse edges. To improve the performance of graph traversal more efficiently, it is necessary to reduce the number of times for graph traversal on conducting a query. In this study, we focus on traversing some edges having the same relationship recurrently. Therefore, we propose a new graph index for enabling to traverse the same type edges efficiently to improve the performance of sub-graph searching.**

***Index Terms*—Graph database; Graph index; Recurrent path**

I. Introduction

In the decade, a lot of GDBs appeared in order not only to efficiently extract sub-graphs from a huge graph but also to analyze it. GDBs The property graph is one of the graph data model [1], whose main characteristic is that nodes and edges have labels and its attribute.

Each node of conventional native GDBs have pointers to its adjacent nodes, that characteristic is called a *index-free adjacency*. This characteristic enables conventional native GDBs to traverse an edge from a certain node by constant cost without being affected by the size of a whole graph. However, the cost of traversing an edge will be affected by the size of a graph database when there are many the number of times for traversing edges in a query.

In this study, therefore, we aim to reduce the number of times for traversing edges by a new graph index, which has harmonious with *index-free adjacency* in native GDBs.

II. Related Works

GDBs are largely divided into native and non-native GDB [1]. We describe the difference of native or non-native GDB, in brief, it is whether a storage engine of GDB has a characteristic called *index-free adjacency*. On GDB applied *index-free adjacency*, each node has pointers to adjacent nodes, so the GDBs enable them to traverse an edge from a certain node at a constant cost. Neo4j[1], which is one of GDB having *index-free adjacency*, can traverse edges the most efficient as shown some experiments. Neo4j, which is one of the GDBs, can traverse edges efficient compared with other GDBs as shown in some experiments [2], [3]. Moreover, most GDBs store data in a property graph model that consists of labeled and attributed nodes and edges [4]. A lot of query languages for GDB have appeared [5]–[7] because the property graph model becomes a de facto standard in GDB.

However, it's not enough only *index-free adjacency* to traverse edges efficiently on a huge and complex graph data like social networks because the number of times of traversing edges become too much.

III. Our Approach

It is effective for the performance of graph traversal to reduce the number of times for traversing edges. For instance, there are *recurrent path*s to analyze a graph representing a social network [8]. Here, we define a *recurrent path* as a set of edges having the same relationship and connected via some nodes. Moreover, social networks tend to become the huge size of a graph because it contains various labels of node and edge and that graph is high connectivity.

A. A Management of recurrent paths

Our purpose is to reduce the number of times for traversing edges for improving the performance of graph traversal even if a huge graph. To succeed in our purpose, an index structure has to enable to skip a few nodes that these cannot avoid being traversed on the nature of *index-free adjacency*. In this paper, hence, we propose a graph index based on *index-free adjacency* that the unit of our graph index is a *recurrent path*.

In this study, therefore, we propose an approach for reducing the number of times for traversing edges by our graph index for *recurrent path*s as shown in Fig.1. The steps of construction for our approach is the following:

1) Collecting *recurrent path*s from each node in graph data with a graph query.
2) Creating extra nodes for managing *recurrent path*s of each node.

This study was partially supported by Grant-in-Aid for the Japan Society for Promotion of Science (JSPS) Fellows.

[1]Neo4j, Inc.: "Neo4j's Home Page," https://neo4j.com/ (reviewed on 20th November, 2019.)

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

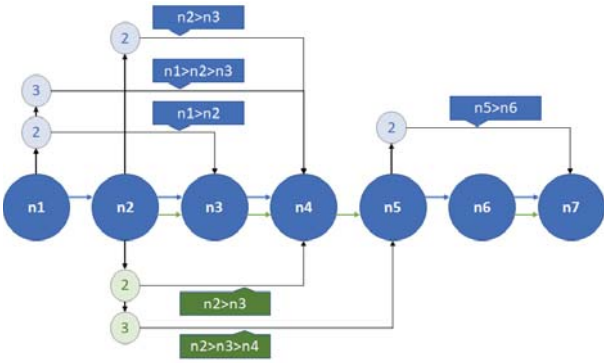

Fig. 1. *Recurrent Path* Management

3) Creating extra edges for managing a shortcut path from each node.
4) Putting the detail of *recurrent path* as an attribute into extra edges.

In the step 2), extra nodes play an important role for managing *recurrent path*s between data nodes (n1, n2, ..., n7 as shown in Fig. 1. Moreover, there are graph queries that require the path of graph traversal such as 'from n1 to n4 via n2 and n3.' Hence, it is necessary that edges of *recurrent path*s have the route from a start node to an end node in the step 3).

B. Traversing Recurrent Paths

We describe how to utilize *recurrent path*s on conducting a query in this section. We can represent a pattern of *recurrent path* by query language named Cypher as follows:

```
()-[:RELTYPE*1..5]->()
```

In above, `:RELTYPE` is a relationship of edge, `()` are unspecified nodes, and `-[]->` is a directed edge. This pattern represents traversing an edge having a relationship named `RELTYPE` for one to five times. Our approach enables a GDB to reduce the number of times for checking whether a node has edges specified in a query, so it is possible to improve the performance for traversing edges.

The following pattern is to traverse edges having a relationship named `RELTYPE` for three to five.

```
()-[:RELTYPE*3..5]->()
```

In this case, conventional GDBs have inefficient because conventional GDBs need to traverse edges from one by one. Our approach enables to skip the first node to third node via a recurrent path.

IV. Experiments

In this paper, we conduct an experiment for evaluating the performance of executing queries with a benchmark for GDBs. Our computing machine of this experiment is configured with the OS is CentOS 7.6.1801 (x86_64), the CPU is Intel Xeon Silver 4114 CPU @ 2.20GHz, and RAM is 256 GB.

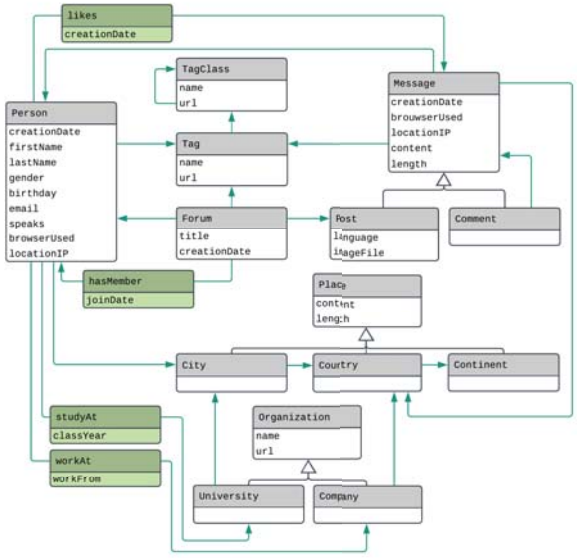

Fig. 2. The model of LDBC SNB dataset

A. Dataset and Experimental Method

In this study, this experiment needs huge and complex graph data and its queries for evaluating the performance of conducting a query and its scalability. There is a benchmark, which is called Social Network Benchmark (SNB), developed by Linked Data Benchmark Council[2] (LDBC) for evaluating the performances of GDBs under any data volumes. As shown in Fig. 2, LDBC SNB dataset has the shape of a social network such as Facebook and Twitter. Moreover, LDBC SNB has a scale factor (SF) for adjusting the data volume so that it enables us to evaluate the scalability of GDB. Furthermore, it has two sets of queries: Interactive Workloads [8] and Business Intelligence Workload [9], for searching and analyzing subgraph. Interactive workloads consist of two types: Interactive Short Workloads (IS), which are simple queries consisted of a few graph traversals, and Interactive Complex Workloads (IC), which are complex queries consisted of many graph traversals, conditions for attributes of nodes and edges, and aggregate functions. Business Intelligence Workloads is a set of transactions about graph analysis for extracting a helpful intelligence of business on social networking services.

In this experiment, we use IC Workloads of LDBC SNB because almost all queries contain a graph traversal of *recurrent path*s. Moreover, we generate a dataset of LDBC SNB that the value of SF is 0.1 with a program for generating LDBC SNB dataset[3]. Furthermore, we construct a graph index for *recurrent path*s on generated datasets. Finally, we conduct IC Workloads implemented by Java language, and we compare

[2]LDBC: "LDBC's Home Page," http://ldbcouncil.org/ (reviewed on 20th November, 2019.)

[3]LDBC: "ldbc/ldbc_snb_datagen.git," https://github.com/ldbc/ldbc_snb_datagen (reviewed on 20th November, 2019.)

TABLE I
PROPERTIES OF IC WORKLOADS

workload	#execution	length of *recurrent path*
IC3	15	1 - 2
IC5	15	1 - 2
IC6	15	1 - 2
IC10	15	2
IC11	15	1 - 2

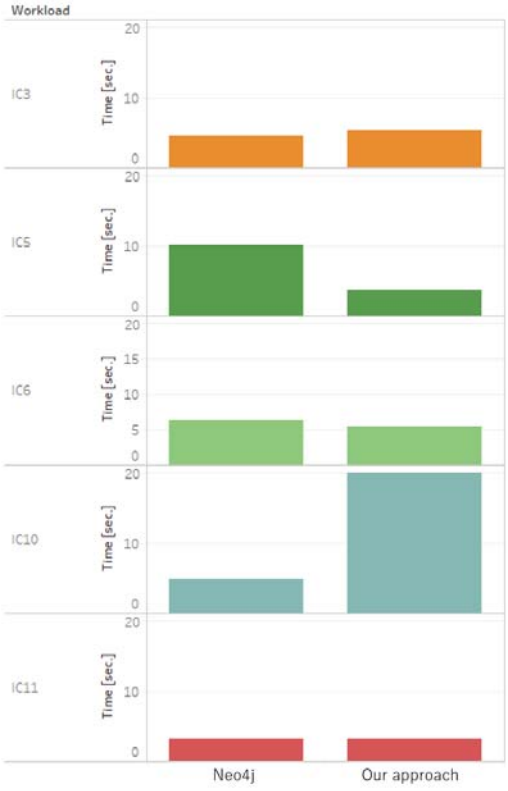

Fig. 3. The consumed time for conducting IC Workloads

the performance for conducting queries on Neo4j with the one on using our graph index.

As a result of constructing our graph index, we unable to construct our graph index that the length of *recurrent path* is more than three for a dataset of SF 0.1. The cause of this result due to a characteristic called the small-world effect that it is possible to reach almost all nodes by traversing some edges. In this study, hence, we use a query that includes a recurrent path with a length of 2 or less as shown in TABLE I.

B. Experimental Result

We executed queries as shown in TABLE I on each case that is Neo4j only and Neo4j with our graph index. We show the result of this experiment in Fig. 3, which the *x-axis* represents each approach: Neo4j on the left side and our approach on the right side, the *y-axis* represents each execution time by IC Workloads.

Our approach enabled to improve the performance of execution time for conducting a query at IC5 and IC6, the reason for this result was that these queries traversed edges to search sub-graphs and only outputted it. However, our approach doesn't enable to conduct queries inefficient at IC3, IC10, and IC11 compared with Neo4j as shown in Fig. 3. Considering it was the success to traverse efficiently at IC5 and IC6, it was a possibility to contain inefficient processes at aggregate functions in our implemented queries.

V. CONCLUSION

In this paper, we proposed a graph index for traversing *recurrent path*s efficiently in order to improve the performance of graph traverse. As we described in Section IV-B, our approach enabled improve the performance of graph traversal limited under queries that have no aggregate operations.

In nearly future tasks, we have to extend our graph index in order to improve the performance of queries contained aggregate functions as well.

REFERENCES

[1] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*. O'Reilly Media, Inc., 2015.

[2] V. Kolomičenko, M. Svoboda, and I. H. Mlýnková, "Experimental comparison of graph databases," in *Proceedings of International Conference on Information Integration and Web-based Applications & Services*, ser. IIWAS'13. ACM, 2013, pp. 115:115–115:124.

[3] S. Jouili and V. Vansteenberghe, "An empirical comparison of graph databases," in *Proceedings of the 2013 International Conference on Social Computing*, ser. SOCIALCOM '13. IEEE Computer Society, 2013, pp. 708–715.

[4] M. A. Rodriguez and P. Neubauer, "Constructions from dots and lines," *Bulletin of the American Society for Information Science and Technology*, vol. 36, no. 6, pp. 35–41, 2010.

[5] O. van Rest, S. Hong, J. Kim, X. Meng, and H. Chafi, "Pgql: A property graph query language," in *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, ser. GRADES '16. ACM, 2016, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/2960414.2960421

[6] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An evolving query language for property graphs," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. ACM, 2018, pp. 1433–1445. [Online]. Available: http://doi.acm.org/10.1145/3183713.3190657

[7] R. Angles, M. Arenas, P. Barcelo, P. Boncz, G. Fletcher, C. Gutierrez, T. Lindaaker, M. Paradies, S. Plantikow, J. Sequeda, O. van Rest, and H. Voigt, "G-core: A core for future graph query languages," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. ACM, 2018, pp. 1421–1432. [Online]. Available: http://doi.acm.org/10.1145/3183713.3190654

[8] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat, M.-D. Pham, and P. Boncz, "The ldbc social network benchmark: Interactive workload," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: ACM, 2015, pp. 619–630. [Online]. Available: http://doi.acm.org/10.1145/2723372.2742786

[9] G. Szárnyas, A. Prat-Pérez, A. Averbuch, J. Marton, M. Paradies, M. Kaufmann, O. Erling, P. Boncz, V. Haprian, and J. A. Benjamin, "An early look at the ldbc social network benchmark's business intelligence workload," in *Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, ser. GRADES-NDA '18. New York, NY, USA: ACM, 2018, pp. 9:1–9:11. [Online]. Available: http://doi.acm.org/10.1145/3210259.3210268