Reproducibility Report for ACM SIGMOD 2022 Paper: "GraphZeppelin: Storage-Friendly Sketching for Connected Components on Dynamic Graph Streams"

NIV DAYAN, University of Toronto, Canada

This report describes the reproducibility process and results for the SIGMOD'22 paper GraphZeppelin [5], which proposes a novel method for finding connected components in large, dynamic, dense graphs.

1 INTRODUCTION

GraphZeppelin [5] addresses the problem of how to efficiently find connected components in large, dynamic, and dense graphs that exceed the size of the available memory. It does this by devising a new $\ell 0$-sampling sketch data structure, CubeSketch, that improves update cost and space consumption relative to existing $\ell 0$-sampling sketches. It also uses buffering techniques for better storage I/O efficiency. The paper is joint work between Rutgers University, Stony Brook University, and MongoDB.

The paper features two tables comparing CubeSketch to the state-of-the-art $\ell 0$-sampling sketch from [1]. Both tables were reproduced to a convincing degree. Furthermore, the evaluation contains seven figures comparing GraphZeppelin to Aspen [2] and Terrace [3], two recent graph processing systems that store the whole graph in memory. Five out of the seven figures were reproduced successfully. In two of the figures, however, Terrace performs much better than reported in the paper. The reason is that Terrace was accidentally run in debug mode during the evaluation. The authors were forthright and transparent about this, and they provided comments on how this affects the conclusions of their paper.

2 SUBMISSION

The authors put their code in an online repository [4]. There is a readme file providing information about how to install dependencies on different architectures (x86_64 and aarch64). There are also system setup instructions for memory management including how to use cgroups and a swapfile. The authors provide a script to compile the code, run the experiments, and produce visual output that enables a one-to-one comparison with the tables and figures in the paper. We consider this an ideal submission.

3 HARDWARE AND SOFTWARE ENVIRONMENT

The experiments were reproduced on the same server that the authors used for evaluation in the paper. The details are at the start of Section 5 in the paper [5].

4 REPRODUCIBILITY EVALUATION

4.1 Process

We cloned the repository into the server and used the script provided by the authors to compile the code and run the experiments. The authors helped us configure the timeout to 10 hours to make the experiments run more quickly.

4.2 Results

Tables 4 and 5 in the paper show the speedup and time reduction of the proposed $\ell 0$-sampling sketch compared to the state-of-the-art sketch from [1]. Compared to the results in the paper, the reproduced results exhibit a slightly higher size reduction and a slightly lower speedup. The authors clarified that this is because of a space optimization they added after the paper was published. This optimization slightly compromises speed. Nevertheless, the reproduced results are all within a factor of 5 − 10% of the ones reported in the paper, so they support the claims made to a satisfying degree.

Figures 11 to 16 in the paper compare GraphZeppelin to Aspen and Terrace. All but two of these figures were reproduced successfully. The exceptions are the two figures that showcase Terrace's query and ingestion rate performance, namely Figures 12 and 16 (b). The authors advised that the reason for the discrepancy is that Terrace was accidentally evaluated in debug mode (i.e., with no compiler optimizations). We appreciate the authors' candidness about this matter.

In the reproduced Figure 12, Terrace's ingestion rate is roughly 3-5 times faster in RAM, and at most 3 times faster on Disk than in the paper. Nevertheless, even with compiler optimizations, Terrace's ingestion rate still lags behind that of GraphZeppelin by more than an order of magnitude. Therefore, the overall conclusion that GraphZeppelin supports faster ingestion is not affected by this change.

In the reproduced Figure 16 (b), Terrace is approximately three times faster in processing queries than reported in the paper and outperforms GraphZeppelin throughout the experiment. Hence, the experiment fails to conclusively support the claim that GraphZeppelin's query latency on dense graphs is less than that of Terrace when both systems' data structures fit completely in RAM. Nevertheless, the authors maintain in our email exchange the overall conclusion that GZ outperforms Terrace for dense graphs still holds because of "GraphZeppelin's much higher ingestion rate (in RAM, and especially on disk), Terrace's query latency scales with the graph density, and Terrace's poor query latency on disk."

5 SUMMARY

This report showcases a successful collaboration between the reproducibility committee and the authors in bringing light to an honest experimental mishap. The nuanced discussion that ensued suggests that despite this mistake, GraphZeppelin still provides many advantages relative to existing algorithms for finding connected components in large graphs. This is a testament to the utility of the SIGMOD Reproducibility Reviewing process in making the scientific output of our community more robust and credible. It will serve future researchers seeking to work on this problem.

REFERENCES

[1] Graham Cormode and Donatella Firmani. 2014. A unifying framework for l0-sampling algorithms. *Distributed and Parallel Databases* 32, 3 (2014), 315–335.

[2] Laxman Dhulipala, Guy E Blelloch, and Julian Shun. 2019. Low-latency graph streaming using compressed purely-functional trees. In *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation.* 918–934.

[3] Prashant Pandey, Brian Wheatman, Helen Xu, and Aydin Buluc. 2021. Terrace: A hierarchical graph container for skewed dynamic graphs. In *Proceedings of the ACM SIGMOD 2021 International Conference on Management of Data.* 1372–1385.

[4] David Tench, Evan West, Victor Zhang, Michael A Bender, Abiyaz Chowdhury, J Ahmed Dellas, Martin Farach-Colton, Tyler Seip, and Kenny Zhang. 2013. GraphZeppelin. *https://github.com/GraphStreamingProject/ZeppelinExperiments* (2013).

[5] David Tench, Evan West, Victor Zhang, Michael A Bender, Abiyaz Chowdhury, J Ahmed Dellas, Martin Farach-Colton, Tyler Seip, and Kenny Zhang. 2022. GraphZeppelin: Storage-Friendly Sketching for Connected Components on Dynamic Graph Streams. In *Proceedings of the ACM SIGMOD 2022 International Conference on Management of Data.*