Reproducibility Report for ACM SIGMOD 2023 Paper: "Efficient and Effective Algorithms for Generalized Densest Subgraph Discovery"

ZHEQI SHEN, UC Riverside, USA

In the reproducibility evaluation, we review the performance of five proposed algorithms and four baselines with respect to time cost, memory usage, and accuracy, over 12 real-world datasets. Most results in the paper are reproduced, showing 3.83×–9774.67× speedups, of which some numbers are even better than those in the paper, and less memory usage compared to the baselines. Specifically, the approximation algorithms can achieve the same high accuracy as the baselines while providing up to three magnitudes of speedups, being overall consistent with the claimed results. Besides, the near-optimal algorithm, Dal*k*S, shows less than 1% of density lower than 0.8 in the approximation ratios as claimed. However, `cCoreExact` and `cCoreApp*` perform worse than baselines in a few tests, not supporting some non-core statements. Regarding ease of reproducibility, the initial artifact needed to be better documented and easier to evaluate, thus entering the shepherd phase. Consequently, the final artifact provides a single master script that prepares the datasets, builds the executable, collects the results, and directly generates figures in the paper with one click.

1 INTRODUCTION

This reproducibility report is related to the paper *Efficient and Effective Algorithms for Generalized Densest Subgraph Discovery*[1] published at SIGMOD 2023 in the *subgraph matching and counting* session. The paper's authors are Yichen Xu, Chenhao Ma, Yixiang Fang, and Zhifeng Bao, where Zhifeng affiliates with RMIT University, Australia, while the other authors are affiliated with the Chinese University of Hong Kong, Shenzhen, China.

The reproduction mainly evaluates the three parts of the experiments in the paper, 1) c-core-based acceleration, 2) approximation algorithms, and 3) Dal*k*S algorithm, all of which will be detailed expanded in section 4. Overall, the reproducibility results indicate the proposed algorithms are efficient and effective, as claimed, producing up to 9774.67× speedup compared to the baselines while achieving the required accuracy. All the running time are shorter compared to the results in the paper. Although the absolute running time largely differs from the numbers in the paper, probably due to the variance of the machine configuration, the most relative speedup to the baselines matches the tables and figures in the paper, as well as the memory usage and approximation ratios.

2 SUBMISSION

2.1 Initial artifact

The reproducibility submission is available at https://github.com/Xyc-arch/DS_DalkS on the GitHub. The initial artifact is the commit e20e29 with all the instructions in `README.md` in the root directory. The data used for experiments are online public datasets linked in the document. The reviewers or anyone who wants to try the code need to manually download the datasets, extract the files, rename the paths, and accordingly modify the build commands following the incomplete steps in the `README.md`. To evaluate the reproducibility, the reviewers need to write their own scripts to loop over all the executables and collect the results. The key metrics in the output are mixed with other logs, and the output formats are different with the varying experiments, so extra efforts are required from the reviewers to locate the relevant numbers in the outputs, which sometimes need clues by reading the source code. Besides, since no figure is automatically generated, to decide whether the results support the claims, reviewers need to either draw the plots on their own or compare them with each single data point of the paper figures.

2.2 Shepherding

The initial artifact is far away from the standards of an ideal reproducibility submission, which provides only a few scripts to recreate the vast majority of the results. Taking the ease of reproducibility into account, the initial artifact was required for shepherding to improve the usability despite the reviewers managing to reproduce the results to some extent with much effort. Zheqi Shen is the shepherd. The shepherding phase includes four stages:

- Fix all the typos and errors found in the initial review and complete the instructions in the documentation
- Refine the existing scripts and improve the portability by removing all the absolute paths and improper assumptions
- Add a single master script that does all the automation, including data preparation, executable building, results collection, and figure generation
- Do final checks and ensure the script is usable on a newly installed operating system

2.3 Final artifact

After the shepherding, the final artifact is the commit 000029 at the same GitHub address and is much easier to use. All the code is in the repository, and the only script that needs attention is `reproduce.py` at the root directory. By simply running `reproduce.py`, one can have the results available in the same form of figures in the paper without the need for other actions, which meets the requirement of the ideal reproducibility submission regarding ease of use.

3 HARDWARE AND SOFTWARE ENVIRONMENT

The hardware platform used for reproduction is equipped with dual Intel Xeon Silver 4314 CPUs with a total of 32 cores at 2.40GHz, enabling hyperthreading. The CPU model is selected to best approach the original configuration in the paper. Although the memory specifications were not provided in the paper, our platform has adequate memory capacity with higher frequency and more channels due to the CPU support. The software environment for reproduction is based on CentOS Stream release 8 with GCC 13 and Python 3.10. The details are listed in the table 1.

Table 1. Hardware & Software environment

	Paper	Repro Review
CPU	2 × Intel Xeon Silver 4210R	2 × Intel Xeon Silver 4314
cores	20	32
GHz	2.40GHz	2.40GHz
RAM	up to 2400MT over 12 channels	128GiB@2667MT over 16 channels
GCC	9.4.0	13.2.1
Python	3.8.10	3.10.12

4 REPRODUCIBILITY EVALUATION

4.1 Process

Before reproducing the results in a one-click manner, one needs to firstly clone the git repository and install the required python packages as described in the *Setup* section of `README.md`. To simplify the operations, we asked the authors to collect the used python packages into `requirements.txt`

with specified versions so that the verified compatible packages can be installed by running `pip3 install -r requirements.txt`

After setting up the python environment, running `python3 reproduce.py` generates all the results in the form of PDF figures in the `outputs/` directory.

Some baseline algorithms may run too long time, and the timeout in the paper is 72 hours. If any error occurs in the script, the reproduction may abort, losing the ongoing results and wasting a few hours. To help verify that the script works on our platform, we discussed with the authors having the timeout configurable and handling the outputs with shortened settings properly so that we can first start a trial run with a small timeout to capture all the potential errors and afterward formally run the script with the standard timeout. The authors also add the functionality to preserve the progress allowing to resume from the last abortion.

4.2 Results

The primary results that are related to key claims are tables 4 to 6 and figs. 9 and 10 in the paper. We use the same index numbers of tables and figures in this section for consistence. Recalling the basic experiment settings, the proposed algorithms in the paper are

- `cCoreExact`, `FlowApp*`, `cCoreApp*`, `cCoreG++`, and `DecomDalkS`

and the baselines are

- `FlowExact`, `FlowApp`, and `Greedy++`

All the reproduction results with respect to running time are shorter than those in the paper. Most results are reproduced, and the key thesis is supported. On the other hand, we notice that `cCoreExact` performs worse than baselines in rare cases (see section 4.2.1), and `cCoreApp*` performs one order of magnitude worse than expected, though still being better than its baseline (see section 4.2.4).

4.2.1 Table 4. This table shows the running time of each algorithm on all the datasets and the relative speedups over the baselines. In the paper, it was to support the claims that 1) the proposed algorithms have much higher performance, 2) *c*-core-based algorithms have higher scalability compared to flow-based approaches, and 3) `cCoreExact` cost less time than `Greedy++`. From the reproduction results at table 4, we observe that claims (1) and (2) are well supported. As for claim (3), most results over the datasets are reproduced, supporting the claim, but a few of them show rather higher time of `cCoreExact`.

Specifically, most time through table 4 are shorter than those in the paper, differing up to 91.25× with `Greedy++` on OF, except `cCoreExact` runs rather 17.79× longer on LW. Although our reproduction platform provides overall higher performance, considering the similar CPU architectures and the same frequency, such a huge discrepancy should be induced by hardware factors that are other than the CPU and unspecified in the paper such as memory channels and cache behaviors. Nevertheless, the relative speedups of proposed algorithms over baselines presented in the last three columns are more reasonable, which were used to support the claims. The speedups of $\frac{FlowExact}{cCoreExact}$ and $\frac{FlowApp*}{cCoreApp*}$ are much better, deviating between +20.18% and +491.30% from the paper results. The last column, $\frac{Greedy++}{cCoreExact}$, shows some different results. The numbers are higher on LJ, DP, and YW up to 1.93× but lower on the rest. Especially on AZ and LW, the speedups are even lower than one, meaning worse performance than the baselines. This is contrary to the statement in the paper that all ratios in the last column are greater than one. However, it is also worth mentioning that the initial artifact ever showed speedups of 3.83× and 168.3× in these two entries, respectively.

Table 4

Dataset	cCoreExact	FlowExact	cCoreApp*	FlowApp	FlowApp*	Greedy++	cCoreG++	$\frac{FlowExact}{cCoreExact}$	$\frac{FlowApp*}{cCoreApp*}$	$\frac{Greedy++}{cCoreExact}$
LJ	13.64 s	>12 h	31.02 s	>12 h	>12 h	8 m 19 s	8.14 s	>3166.04	>1392.51	36.60
FT	>12 h	>12 h	>12 h	>12 h	>12 h	>12 h	>12 h	—	—	—
OK	7 m 25 s	>12 h	3 m 11 s	>12 h	>12 h	8 m 43 s	26.30 s	>97.05	>226.62	1.18
YT	3.27 s	8 h 53 m	3.79 s	>12 h	5 h 19 m	42.83 s	1.32 s	9774.67	5046.46	13.08
DP	0.35 s	15 m 44 s	0.39 s	1 h 11 m	7 m 57 s	11.57 s	0.27 s	2725.16	1210.20	33.41
AZ	51.46 s	23 m 30 s	44.20 s	1 h 21 m	14 m 31 s	14.76 s	7.59 s	27.40	19.72	0.29
LB	28.67 s	1 h	16.41 s	>12 h	>12 h	32.25 s	3.09 s	126.70	>2631.95	1.13
NM	0.01 s	2.36 s	0.01 s	16.65 s	2.02 s	0.26 s	0.01 s	237.76	206.38	26.38
FF	0.02 s	1.17 s	0.02 s	16.23 s	2.43 s	0.07 s	0.01 s	54.05	110.73	3.42
OF	0.02 s	0.32 s	0.04 s	4.47 s	0.49 s	0.04 s	0.01 s	13.47	11.86	1.84
LW	12 m 44 s	>12 h	47.29 s	>12 h	>12 h	5 m 39 s	7.82 s	>56.55	>913.49	0.44
YW	3.52 s	9 h	5.59 s	>12 h	5 h 41 m	1 m 18 s	1.67 s	9216.98	3664.86	22.13

4.2.2 Table 5. The table shows the statistics of three directed graphs and the corresponding time cost of `FlowExact` and `cCoreExact`. The statistics are consistent with that in the paper. The time in reproduction, again, is much shorter than the paper results, differing by 1.93×–6.36×, and there is no explicit clue on such huge speedups of both two algorithms. From table 5, we observe that `cCoreExact` consistently outperform `FlowExact` over these three datasets up to 41.38×, which is a significant number but lower than the claimed 100×.

Table 5

Statistics				Reproduction		Paper	
Dataset	# vertices	# edges	$\rho(S^*)$	FlowExact	cCoreExact	FlowExact	cCoreExact
WV	7,115	103,689	71.68	1.56 s	0.61 s	7.18 s	3.38 s
SF	281,903	2,312,497	75.95	>30 m	3 m 51 s	5 h 26 m	12 m 34 s
ND	325,729	1,497,134	123.73	24 m 50 s	36.76 s	2 h 38 m	1 m 11 s

4.2.3 Table 6. This table presents the densities of subgraphs found by `cCoreExact` and `Greedy++` over 12 datasets, evaluating the quality of the two algorithms. The paper claims that, while costing much less time than `Greedy++`, `cCoreExact` only fails to attain the optimal density over FT, LJ, LB, and OF, i.e., larger $\hat{\rho}(S^*)$ in the table. In our reproduction results, `cCoreExact` fails on LB, NM, and LW instead. The $\hat{\rho}(S^*)$ on LW (2504.00) is much larger than the optimal value (774.05), while the other numbers match the paper results with the maximum absolute bias of 0.35.

4.2.4 Figure 9. The figure depicts how the time cost grows with the accuracy increases to show the tradeoff between efficiency and accuracy of the approximation algorithms `cCoreApp*` and `Greedy++`. The trends and data points of the baseline, `Greedy++`, are consistent with the paper results. The proposed algorithm `cCoreApp*`, however, shows slightly different trends on LJ and a one-order-of-magnitude slowdown over LB and YT compared to the same figure in the paper.

4.2.5 Figure 10. This figure shows the proportion of $\frac{k}{|\hat{K}*|}$ over four datasets, intending to present the capability of `DecomDalkS` to find solutions with an accuracy over 0.8. We observe that the blue area, representing solutions with a density less than 0.8 of the optimum, is only in a tiny ratio on each dataset, which matches the results in the paper.

5 SUMMARY

With the improvement of ease of reproducibility after shepherding, the final artifact clearly shows the short running time of the proposed algorithms and the high speedups over the baselines

Table 6

Dataset	$\rho(S^*)$ by cCoreExact	$\hat{\rho}(S^*)$ by Greedy++
FT	273.52	273.52
OK	227.87	227.87
LJ	193.51	193.51
YT	45.60	45.60
DP	56.57	56.57
AZ	4.80	4.80
LB	1068.41	1068.40
FF	1632.10	1632.10
NM	47.76	47.75
OF	39.85	39.85
LW	2504.00	774.05
YW	168.05	168.40

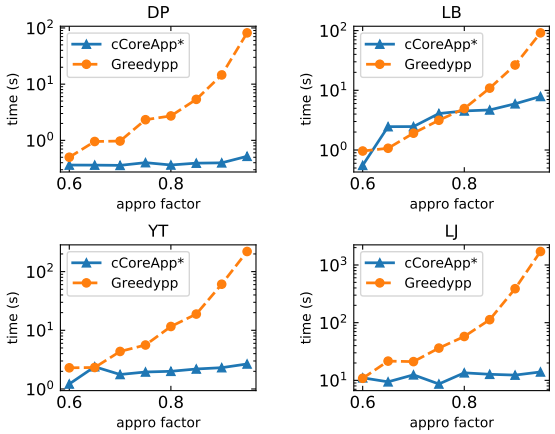

Fig. 9

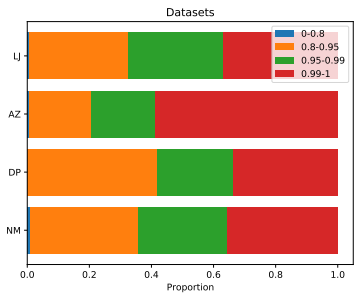

Fig. 10

while maintaining the required accuracy, supporting the core thesis of the paper. In less critical experiments, we noticed that `cCoreExact` and `cCoreApp*` can perform worse than expected in

some cases; thus, not all reproduction results perfectly match the paper data. Having said that, the core thesis of the paper is well supported.

REFERENCES

[1] Yichen Xu, Chenhao Ma, Yixiang Fang, and Zhifeng Bao. 2023. Efficient and Effective Algorithms for Generalized Densest Subgraph Discovery. *Proceedings of the ACM on Management of Data* (2023). https://doi.org/10.1145/3589314