Reproducibility Report for ACM SIGMOD 2023 Paper: "Efficient Star-based Truss Maintenance on Dynamic Graphs"

LUCA GAGLIARDELLI, Università degli Studi di Modena e Reggio Emilia, Modena, Italy
DANIEL KOCHER, University of Salzburg, Austria

We have been able to reproduce the findings of the original paper, obtaining similar results. The authors provided data files, a set of scripts to run the experiments automatically, and routines (i.e., Python scripts) to plot the results.

1 INTRODUCTION

The original paper [1] has been published at ACM SIGMOD 2023:

> *Efficient Star-based Truss Maintenance on Dynamic Graphs*
>
> ZITAN SUN, Hong Kong Baptist University, China
> XIN HUANG, Hong Kong Baptist University, China
> QING LIU, Zhejiang University, China
> JIANLIANG XU, Hong Kong Baptist University, China

Summary of the Original Paper. The original paper studied the truss maintenance problem for star-shaped insertions/deletions into/from evolving graphs in the context of k-truss decomposition, which discovers all non-empty k-trusses in a graph for varying k. A star-shaped graph consists of a set of edges that are incident to a single common node, and the insertion/deletion of star-shaped graphs occurs naturally in the real world, e.g., a new member of a social network often connects to many other members. Existing solutions neglect the local correlation of graph updates and treat each update as an isolated single-edge insertion/deletion. The authors proposed a novel AffBall-based truss decomposition technique to estimate the trussness of the local neighborhood that is based on the observation that a trussness update exhibits a local property for a star-shaped insertion. Furthermore, the authors proposed onion layers and onion support to develop an efficient trussness refinement technique with a time complexity that is bounded by the number of edges that change onion layers. This also allows the authors to extend the star-based insertion technique to deletions. In their experiments on 9 real-world graphs, the authors showed that their star-based approach outperforms the state of the art by several orders of magnitude.

Summary of the Reproducibility Result. The reproducibility package replays all experiments and our local results (over multiple different machines) are similar to the experimental results reported in the original publication [1]. Most importantly, the key statement of Star approach outperforming the state of the art seems to hold.

2 SUBMISSION

The paper is accompanied by a reproducibility setup that allows the reader to replay all experiments and to evaluate the runtime/memory efficiency as well as the effectiveness of the proposed approach. There is a `README` file that briefly describes each single subdirectory and file in the root of the reproducibility package. Such a `README` file is present within each subdirectory. In addition, there is a description of how to execute the main script that generates experimental results in the corresponding directories (including an estimation on how long the experiments will be running, and some more options to configure the main script such as number of repetitions and parallelization of the experiments).

- The reproducibility package is hosted in a GitHub repository: https://github.com/jinrdfh/TrussMaintenance
- The repository contains a directory for the code and a directory for replaying the experiments of the original paper [1].
- The data files are downloaded by the main script.
- The top-level `README` file (briefly) describes the structure of the code repository and how to customize the main script, e.g., the number of repetitions or the degree of parallelization for the experiments.
- Shell scripts are provided to automatically download the datasets for the experiments (hosted on third-party websites) and to compile the code (of the proposed approach and its competitors).

3 HARDWARE AND SOFTWARE ENVIRONMENT

Table 1 compares the original setup [1] and the setups that have been used to reproduce the experiments (independently). Additionally, we were able to compile the code and execute the experiments on at least one additional non-Intel machine; we omit the results because they are similar to those reproduced on our main machines (cf. Table 1). The `README` file estimates a running time of about 64 hours (about 2.6 days) for all experiments, which we can confirm based on our observations.

Table 1. Hardware & Software environments.

	Paper	Repro Review 1	Repro Review 2
OS	Oracle Linux 8.7 (64 bit)	Debian 12 (64 bit)	Debian 10 (64 bit)
CPU	Intel Xeon E5-2630 v4	Intel i9-13900KF	Intel Xeon E5-2630 v3
Cores	20 (from CPU spec.; not in [1])	24	16
GHz	2.2	3.0	2.4
RAM [GB]	256	128	96

4 REPRODUCIBILITY EVALUATION

4.1 Process

We followed the instructions reported in the `README` file. (1) We executed the main script (`master.sh` (after setting the privileges using `chmod +x master.sh`). Running all experiments in a single batch took roughly the estimated time period, i.e., about 64 hours. We used the `screen` manager to multiplex the terminal (i.e., detach from and re-attach to the host machine), and we observed that no user intervention is required while running the experiments. (2) We also ran each experiment separately to better control them (e.g., we can repeat a single experiment in case of a failure). The corresponding files (e.g., `Exp-I.sh` in the `Exp-I` subdirectory) can be customized but there is not much documentation to be found within the respective scripts; however, the experimental setup is tweakable to fit different parameters. In the original setup each experiment is repeated 100 times, for the reproducibility due to time and memory constraint, we repeated them at most 10 times. We also tried to run some experiments in parallel and we noted that a high repetition number may cause an out-of-memory error with less than 256GB of main memory. The number of repetitions must be set in the `repeat.txt` file before starting the experiments. This parameter is used to build indexes (script `data/buildIndex.sh`) that are used in all experiments. Therefore, if the repetition number is increased, the indexes are built again.

Overall, repeating the experiments was easy to achieve and no difficulties were encountered.

4.2 Results

The reproducibility package replays all experiments found in Section 7 of the original paper [1].

Overall, we were able to reproduce the main results that were reported in the original paper, showing that the proposed algorithm (Star) outperforms the state-of-the-art competitors. In the following, we provide the obtained results with a brief comment for each experiment.

4.2.1 Exp-I: Efficiency evaluation. The results of the first experiment are reported in Table 2. They are mostly consistent with those presented in the original paper (except for some negligible differences in the indexing times).

4.2.2 Exp-II: Evaluation of two phases of Star. The results of the second experiment are presented in Figure 2. In the original paper *Phase II* is always slower than *Phase I*. In our results, we were unable to verify this behavior for the `Wise` dataset (in particular), and the `DBLP` and the `Pokec` datasets (minor differences), respectively.

Exp-III: Evaluation of the number of examined edges in Star. We obtained the results that are presented in Table 3. The trends are consistent with those reported in the original paper. The absolute numbers differ but we ran each experiment with fewer repetitions (at most 10) than the original paper (which reports the avg. over 100 runs). Due to this and due to randomization, e.g., to choose a center node, we consider the obtained results to be consistent with the original paper.

4.2.3 Exp-IV: The effect of star motif size. The obtained results are presented in Figure 1. They are consistent to those reported in the original paper.

4.2.4 Exp-V: Evaluation of single-edge insertion/deletion. The results for insertions and deletions are reported in Figure 3 and Figure 4, respectively. In the insertion phase (cf. Figure 3), XH and Star have a runtime close to those in the original paper, while Order presents higher times. In the deletion phase (cf. Figure 3), the results are slightly different, the highest difference is shown on `Orkut` network. However, we performed fewer repetitions (1) than the original paper (100) and the obtained runtimes are very small (sub-millisecond range). Therefore, this variation seems acceptable.

4.2.5 Exp-VI: The efficiency evaluation of handling multiple star insertions. The results of this experiment are shown in Figure 5, they are coherent to those presented in the original paper.

4.2.6 Exp-VII: scalability evaluation. Figure 6 presents the results that are consistent with those in the original paper: Increasing the size of the graph increases the time needed for running the algorithms.

4.2.7 Exp-VIII: case study on a patent citation network. The results are presented in Figure 7: In the original paper XH is always the slowest algorithm, here it performs better than Order for some timestamps. However, Star outperforms both competitors as reported in the original paper.

Table 2. Efficiency and indexing evaluation of all truss maintenance algorithms, best results are reported in bold. This Table reproduce *Exp-I* of the original paper.

	Star Insertion (ms)				Star Deletion (ms)		
Networks	XH	NodePP	Order	Star	XH	Order	Star
Deezer	0.397	2	0.612	**0.006**	0.089	**0.002**	0.003
Amazon	2.226	11	4.808	**0.007**	.505	**0.003**	**0.003**
DBLP	5.077	37	14.144	**0.023**	1.133	0.023	**0.014**
Skitter	22.821	96.5	61.866	**0.019**	4.650	**0.009**	**0.009**
Patents	28.859	128	80.262	**0.011**	6.216	0.008	**0.006**
Pokec	48.189	241	119.225	**0.087**	10.622	0.045	**0.029**
LJ	89.115	362	159.041	**0.066**	16.341	0.034	**0.024**
Orkut	238.745	1165	408.027	**0.412**	45.846	**0.284**	2.173
Wise	61626	2387351	929.216	**2.129**	50.928	**0.569**	1.074

	Index Size (MB)				Indexing Time (seconds)			
Networks	**XH**	**NodePP**	**Order**	**Star**	**XH**	**NodePP**	**Order**	**Star**
Deezer	**2**	**2**	3	3	0.033	0.033	**0.032**	0.034
Amazon	**14**	**14**	18	18	0.362	0.362	0.349	**0.339**
DBLP	**34**	**34**	44	44	**1.044**	**1.044**	1.117	1.050
Skitter	**165**	**165**	212	212	**21.996**	**21.996**	23.143	22.438
Patents	**284**	**284**	316	316	**10.910**	**10.910**	11.121	10.954
Pokec	**338**	**338**	426	426	**24.803**	**24.803**	26.114	25.209
LJ	**690**	**690**	818	818	54.949	54.949	60.237	**54.838**
Orkut	**1943**	**1943**	2236	2236	**305.312**	**305.312**	333.626	305.630
Wise	**4931**	**4931**	4985	4985	**1165.909**	**1165.909**	1184.144	1179.429

Table 3. The evaluation of the number of examined edges for Star

	Deezer	**Amazon**	**DBLP**	**Skitter**	**Patents**	**Pokec**	**LJ**	**Orkut**	**Wise**
$\|E\|$	$1.3 \cdot 10^5$	$9.3 \cdot 10^5$	$2.3 \cdot 10^6$	$1.1 \cdot 10^7$	$1.7 \cdot 10^7$	$2.2 \cdot 10^7$	$4.2 \cdot 10^7$	$1.2 \cdot 10^8$	$2.6 \cdot 10^8$
$\|B_{in}\|$	4.5	2	6.5	6	9.5	24.5	25	72	44.5
$\|L\_CHG\|$	4	1.5	19.5	7	4.5	47.5	39	90	148.5
$\|T\_CHG\|$	2.5	1.5	29	7.5	5	41.5	24.5	79	61.5

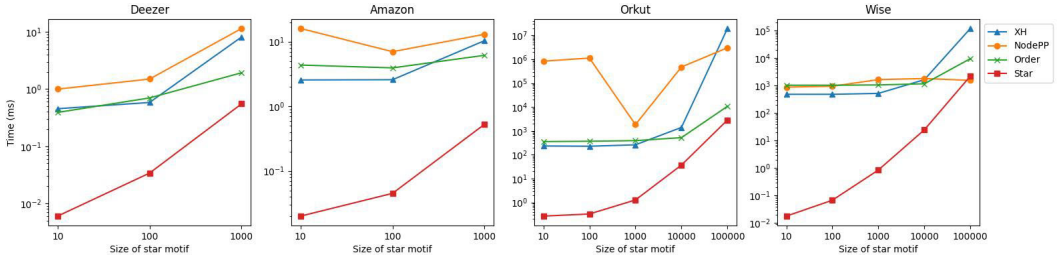

Fig. 1. The effect of star motif size (insertion)

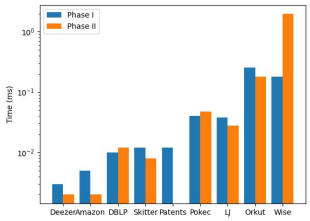

Fig. 2. Evaluation of two phases of Star

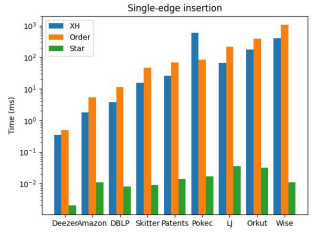

Fig. 3. Evaluation of single-edge insertion

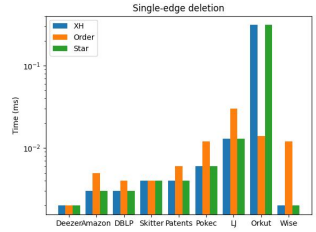

Fig. 4. Evaluation of single-edge deletion

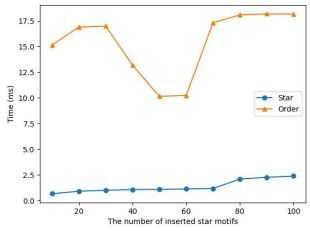

Fig. 5. The efficiency of handling multiple overlapping star motifs on DBLP

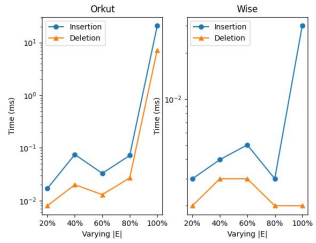

Fig. 6. Scalability evaluation

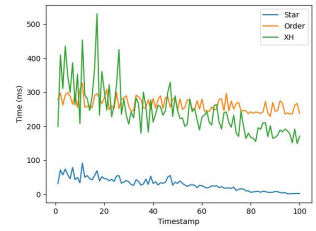

Fig. 7. Case study on Patent citation network

REFERENCES

[1] Zitan Sun, Xin Huang, Qing Liu, and Jianliang Xu. 2023. Efficient Star-Based Truss Maintenance on Dynamic Graphs. *Proc. ACM Manag. Data* 1, 2, Article 133 (jun 2023), 26 pages. https://doi.org/10.1145/3589278