Homework 3
TRIÈST Implementation

Badai Kesuma Hamid Dimyati

November 2020

1. Code commentary

We implement TRIÈST streaming graph processing methods [1]. The implementation of the algorithm is done in Python mainly using `defaultdict` from `collections` library, `numpy, pandas, matplotlib`, and `random` library. The application counts the local and global triangles of the git network [2] with fixed memory size using the reservoir sampling technique. This application is capable of extracting triangles using base and improved versions of the algorithm.

Within the code (`triest.py`), we created a class called `edges` for insertion and deletion operation of edges into the sample `S`. The next class is called `triesBase`, which implements the given pseudocode of a first algorithm in [1]. This class gives an output in the form of global triangles estimation obtained from the multiplication between the estimator and the global counter recorded within fixed sized memory. The estimator used in the first algorithm:

$$\xi^t = max\left\{1, \frac{t(t-1)(t-2)}{M(M-1)(M-2)}\right\}$$

The next one, we created a class called `triestImprove`, provided with the same parameter as previous class object `memory`, which processes the improved version of `triesBase`. This algorithm incorporates three changes covering; (1) moving `update_counter` function before `if` block, (2) removing `update_counter` when an edge is removed from selected sample `S`, and (3) performs a weighted increase of the counters. Instead of multiplying the estimator and the global counter, to find the global triangles in this algorithm we use the global counter itself as the estimator. Instead of using 1 and -1 as the global counter, we calculate the estimator using:

$$\eta^t = max\left\{1, \frac{(t-1)(t-2)}{M(M-1)}\right\}$$

2. How to run

```
git clone https://github.com/hamiddimyati/id2222-data-mining-advanced.git
cd id2222-data-mining-advanced/assignment-3
python3 triest.py
```

We could also modify all the parameters through the python code, such as `memory`.

3. Results

In the beginning, we compare the Triest base version with the improved version by using memory size as much as 10,000 out of 289,003 total edges. We can see the result in Figure 1 that the Triest improve could estimate the global triangles far better than the Triest base version. In order to see the complete performance comparison of two algorithms, we run an experiment by defining different values of `memory` as the input parameter. For detail results could be seen in Figure 2. The red line is the ground truth of the total global triangles. It could be concluded that the Triest improve version outperforms the base version.

```
(base) Hamids-MacBook-Pro:assignment-3 hamiddimyati$ python3 triest.py
Estimating Global Triangles of Graph using Memory Size: 10000
Ground truth: 523810
Estimation by Triest Base: 362179 in 11.822s
Estimation by Triest Improve: 544781 in 11.226s
```

Figure 1. Output of the code `triest.py`

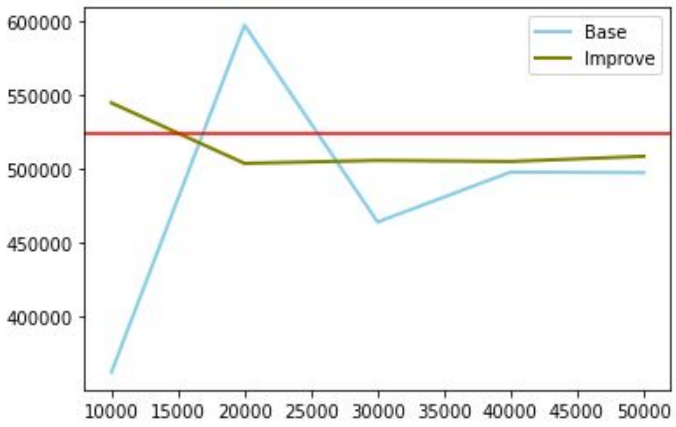

Figure 2. Global triangles estimations comparison based on the memory used

4. Extra Bonus

1. What were the challenges you have faced when implementing the algorithm?

Deciding the size of the memory is the trickiest part. If we choose the value too small, the estimation accuracy could be low. It is also compounded by the run time which becomes worse when the memory used is around 50% of the total data.

2. Can the algorithm be easily parallelized? If yes, how? If not, why? Explain.

We think that this algorithm is possible to be implemented in the parallel settings. For instance, since this algorithm keeps maintaining the sample S as much as memory provided in the single machine, we could think that in a parallel setting, we will have S_i for each machine in the cluster. From the sample S_i, the algorithm also keeps the number of global triangles T_i. Thus, we could modify the estimation function by averaging the Ti from different machines and obtain a single estimator for the global triangles of the whole graphs. But since S and T are the core of the algorithm with a global approach, it will be difficult to parallelize the algorithm.

3. Does the algorithm work for unbounded graph streams? Explain.

One of the advantages of this algorithm is to use a fixed amount of memory size. This approach overcomes the drawbacks caused by using a fixed sampling probability which are the possibility of running out of memory when using a large p and suboptimal estimation when using small p, it is difficult to determine the correct p. Differently, since the Tries use a fixed amount of memory size, it could avoid any potential danger of running out of memory due to unknown size of data stream. In other words, the Triest algorithm could be obviously implemented in the unbounded graph stream.

4. Does the algorithm support edge deletions? If not, what modification would it need? Explain.

The first two algorithms (base and improved) work on insertion-only streams, while the third (FD) can handle fully-dynamic streams where edge deletions are allowed. To support edge deletion in arbitrary order, fully-dynamic TRIÈST implements Random Pairing (RP), a sampling scheme that extends reservoir sampling. The RP scheme's idea is that edge deletions seen on the stream will be “compensated” by

future edge insertions. The algorithm keeps a counter d_1 (resp. d_0) to track the number of uncompensated edge deletions.

References

[1] Stefani, L. D., Epasto, A., Riondato, M., & Upfal, E. (2017). Triest: Counting local and global triangles in fully dynamic streams with fixed memory size. ACM Transactions on Knowledge Discovery from Data (TKDD), 11(4), 1-50.

[2] Rozemberczki, B., Allen, C., & Sarkar, R. (2019). Multi-scale attributed node embedding. arXiv preprint arXiv:1909.13021.