ID2222 Lab 3 report

Fynn van Westen and Axel Myrberger

November 2021

1 Paper selection

We decided to go with the paper "TRIÈST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size". TRIÈST implements a suit of algorithms for estimating local and global triangle counts in streamed graph data with a fixed memory size. This report covers the reimplementation of the TRIÈST-BASE and TRIÈST-IMPR algorithms.

2 The solution and instructions

The code is divided into 3 different sections. First there is algo_lib.py that contains the algoritms used to solve the task, namely TriestBase, TriestImproved and GraphSample. Additionally the module data_loader.py reads the data. In this class, Dataloader, the text file is read line by line and and the data saved and returned from the class. One very important feature of this class is that changes edges to go to one direction. The algorithm in the Triest is only for undirected graphs so if we have an directed graphs we need to change the directions. The run.py function is used to run all the program and start the functions. Here the we have selected the dataset facebook_combined that we found on the web page, all the data needed to evaluate the algorithms. The data set used to evaluate is undirected.

The modules needed to run the code:

- time
- argparse
- os
- numpy
- random
- itertools
- collections

3 Results

The data set used for this test had a total of 1612010 global triangles. We benchmarks our algorims against this number as can be seen in the tables below. The data set have 88233 lines and the computing time was satisfying even for high number of M on a laptop computer.

3.1 Trieste Base

M used	Diff against true count	time
500	3916310	0.21
1000	-234081	0.29
2000	193808	0.54
5000	-226390	2.531
10000	35669	5.39

3.2 Trieste Improved

M used	Diff against true count	time
500	118800	0.33
1000	66719	0.42
2000	44200	0.74
5000	11140	2.66
10000	57031	5.49

4 Bonus points

1. **What were the challenges you have faced when implementing the algorithm?**
   Our biggest problem was that the first dataset we chose was a directed graph with back, forth connections. This lead to multiple deletions of the same node in the sample, resulting in the crash of the algorithm, since the node was not present after the first deletion. We circumvented this problem by using a strictly undirected graph from Facebook data and further tried to create an on the fly quick-fix for directed graphs, by ordering the node-IDs of edges(bringing both directions in the same order) and checking if they already exist in the sample. this is not perfect, since it only catches edges that are currently in the sample but reads back edges that are currently not in the sample double (once for each direction).

2. **Can the algorithm be easily parallelized? If yes, how? If not, why? Explain.**
   We believe it would be possible to parallalize the algotrims, but certainly not easily. The algorithm works on global counters (e.g. t, global-triangle-count, etc.) which would have to be synchronized between processes. Synchronization can be achieved with a fundamentally different algorithm, so not easily as the question states.

3. **Does the algorithm work for unbounded graph streams? Explain.**
   Due to the use of reservoir sampling, which fixes the memory size, the algorithm is working for unbounded graph streams. The potential problem with unbounded graph streams is that it can be very memory inefficient, or more precisely infinitely memory intense. The approach suggested in the paper works around this issue by using global counters.

4. **Does the algorithm support edge deletions? If not, what modification would it need? Explain.**
   No. The implemented algorithms only support insertions, a solution to also allow deletions is the fully dynamic algorithm introduced in the paper. This one tries to compensate deletions with later encountered insertions. For such an approach to work, the algorithm needs to keep track of uncompensated deletions as counters.