ID2222 Data Mining

Niklas Barth and Yu Gao

**Lab 3 Report**

**Abstract**

In this lab we established a implemented an algorithm that allows to estimate the triangle and transitivity count in a streamed graph through reservoir sampling. We followed the proposed algorithm from this paper:

*M. Jha, C. Seshadhri, and A. Pinar,*[*A Space-Efficient Streaming Algorithm for Estimating Transitivity and Triangle Counts Using the Birthday Paradox*](https://arxiv.org/pdf/1212.2264.pdf)*, ACM TKDD, 9-3, 2015.*

**Code Structure**

**Class TriangleCountsStreamingAlgorithm**

This class implements the pseudo code from the cited paper above. Please go in the code to see more comments as the code is heavily commented. Tt was needed for efficient bug fixing during development.

**Class GraphStorage**

Loads the data from the test file. We choose a test graph from google which you can find here: <https://snap.stanford.edu/data/web-Google.html>

While reading the edges, we had to make sure that we don’t count duplicates in the edge.

**Instructions for use**

1. Unzip the project folder. In it you find the dataset in zipped format. Make sure to unzip that, too. (Having problems? Go to GraphStorage Class and make sure your file path is set up correctly)
2. Run the main class

**Result**

* We choose a reservoir size for both edges and wedges of 150 as increasing would lead to more and more run time
* Our result for the first 2000 lines of the input file was around 294 estimated triangles and a transitivity of 6,4% (which is close to the estimated 5,5% of the whole graph)

Text

Description automatically generated

*Figure 1 Estimates reading the first 2000 lines of the file which corresponds to t=1623 actual edges due to duplications in the file*

OPTIONAL BONUS PART:

**1) What were the challenges you faced when implementing the algorithm?**

Firstly, the presented pseudo code is very vague for an efficient implementation. This is because the two presented functions (streaming triangles and update) are working together but parameter passing between the two is not described in detail. Therefore, one is to understand very well what the algorithm does to conclude how to implement this algorithm.

Secondly, from the above little errors came up at first such as wedges and edges being added multiple times due to the presented pseudo code. So, we figured we had to implement breaks to stop that dynamic.

Thirdly, in the input data we had many, duplicated edges (in terms of viewing the graph as undirected) which we didn’t realize in a long time. Therefore, our triangles counts were inflated a lot because the sample data had multiple chances to discover closed wedges. However, after removing the duplicates are estimates decreased and we ended up with solid estimates (around 6.5% for our sample vs 5,5% for the whole graph).

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

Yes, it does. That's the whole point. It will always output the new T and K for new incoming elements. Therefor it can continuously be used. The reservoirs will just continue to update, and checks are done to count how many triangles there are in the sampled data.

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

The algorithm does currently not support edge-deletion. However, with some tweaks it could be adjusted. If that’s computationally efficient is a different question. If an edge on the graph was deleted the algorithm would need an streamed input while it’s running (this could be done like so: For every iteration t check if there was a deletion performed. Deletions could for instance be added to a buffer that the algo accesses). If an edge was deleted the following needs to be done immediately before running the usual procedure:

* Clean wedge reservoir (remove all the wedges that contain the deleted edge(s))
* Clean isClosed (set isClosed for the affected wedges to null/false depending on implementation)
* Clean edge reservoir (remove all the deleted edges from the reservoir)

The previous estimates will then of course not reflect the graph with deletions but that’s fair because all the new estimates will reflect the changes of the graph. Please also not that of course the new incoming edges need to be updated such that no deleted edge comes in again in case the data was previously loaded.