Report 3: Mining Data Streams

Group 34: Sofia Krylova, Iosif Koen

November 28, 2022

1 Introduction

In the previous home assignments, we assumed that we were mining a database, so all the data was available at any time. However, this assignment assumes that data arrives in a stream or streams, and if it is not processed immediately or stored, then it is lost forever. Also, the data arrives so rapidly that it is not feasible to store it all in active storage.

In this home assignment, we were asked to implement a streaming graph processing algorithm described in one of the provided papers of our choice.

To accomplish this task, we performed the following two steps:

1. First, implement the reservoir sampling or the Flajolet-Martin algorithm used in the graph algorithm presented in the paper you have selected;

2. Second, implement the streaming graph algorithm presented in the paper that uses the algorithm implemented in the first step.

We chose the following paper: L. De Stefani, A. Epasto, M. Riondato, and E. Upfal, TRIÈST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size, KDD'16.

Triest is a suite of sampling-based, one-pass algorithms for adversarial fully-dynamic streams to approximate the global number of triangles and the local number of triangles incident to each vertex. These algorithms significantly and consistently reduce the average estimation error by up to 90%. We have implemented basic and improved versions that work on insertion-only streams.

We did not use any dedicated data frameworks. The task is implemented in Python.

2 Data source

To test our implementation, we used one of the publicly available graph datasets, that also provides some metrics.

3 Solutions

3.1 Reservoir sampling

Reservoir sampling is a technique for maintaining a uniformly chosen sample of positions in the stream.

1. If $t \leq M$, then the edge $e_t = (u, v)$ on the stream at time t is deterministically inserted in S.
2. If $t > M$, triest-base flips a biased coin with heads probability M/t. If the outcome is heads, it chooses an edge $(w, z) \in S$ uniformly at random, removes *(w, z)* from S, and inserts *(u,v)* in S. Otherwise, S is not modified.

We have implemented the reservoir sampling in a *sample-edges* method, as was suggested in the paper.

3.2 TRIEST-BASE algorithm

We have implemented a Triest-Base algorithms, as was proposed by the paper. The basic workflow can be seen here:

function SAMPLEEDGE($(u, v), t$)
  if $t \leq M$ **then**
    return True
  else if FLIPBIASEDCOIN($\frac{M}{t}$) = heads **then**
    $(u', v') \leftarrow$ random edge from $\mathcal{S}$
    $\mathcal{S} \leftarrow \mathcal{S} \setminus \{(u', v')\}$
    UPDATECOUNTERS($-, (u', v')$)
    return True
  return False

Estimation is counted by the following formula:

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

The estimation of global number of triangles is very sensitive to the choice of M. Bigger $M - >$ better accurancy, however the computational time also increases.

3.3 TRIEST-IMPR algorithm

As described in the paper, the main improvements we introduced to the Triest algorithm were the following:

1. *UpdateCounters* is called unconditionally for each element on the stream, before the algorithm decides whether or not to insert the edge into S;
2. Triest-impr never decrements the counters when an edge is removed from S;
3. *UpdateCounters* performs a weighted increase of the counters using the following formula as weight:

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

These changes can be seen in the $Triest\_Impr.py$ class.

4 Evaluation

We ran both algorithms on the same *Arxiv HEP-TH* dataset. Its information can be found on a screenshot below:

Dataset statistics	
Nodes	9877
Edges	25998
Nodes in largest WCC	8638 (0.875)
Edges in largest WCC	24827 (0.955)
Nodes in largest SCC	8638 (0.875)
Edges in largest SCC	24827 (0.955)
Average clustering coefficient	0.4714
Number of triangles	28339
Fraction of closed triangles	0.1168
Diameter (longest shortest path)	17
90-percentile effective diameter	7.4

Console output:

```
====================700====================
BASE: Global T 700: 0
BASE: Global triangle count with M 700: 0
BASE: Spent time 0.153s.
IMPR: Global T with M 700: 220748
IMPR: Spent time 0.208s.
```

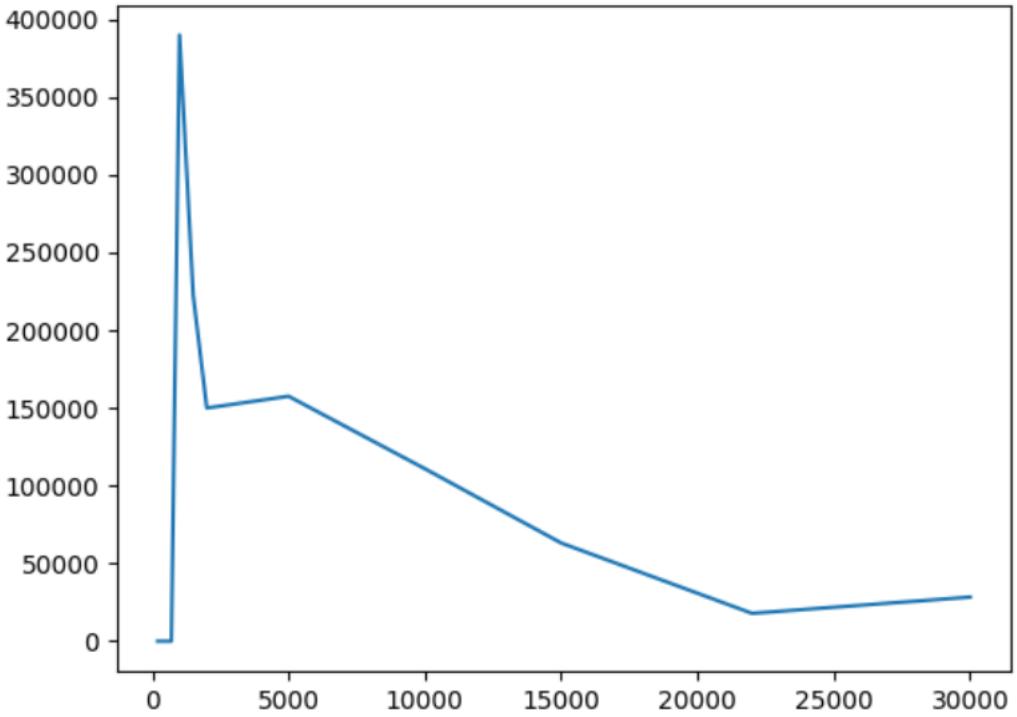

Figure 1: Results for the Triest-base algorithm

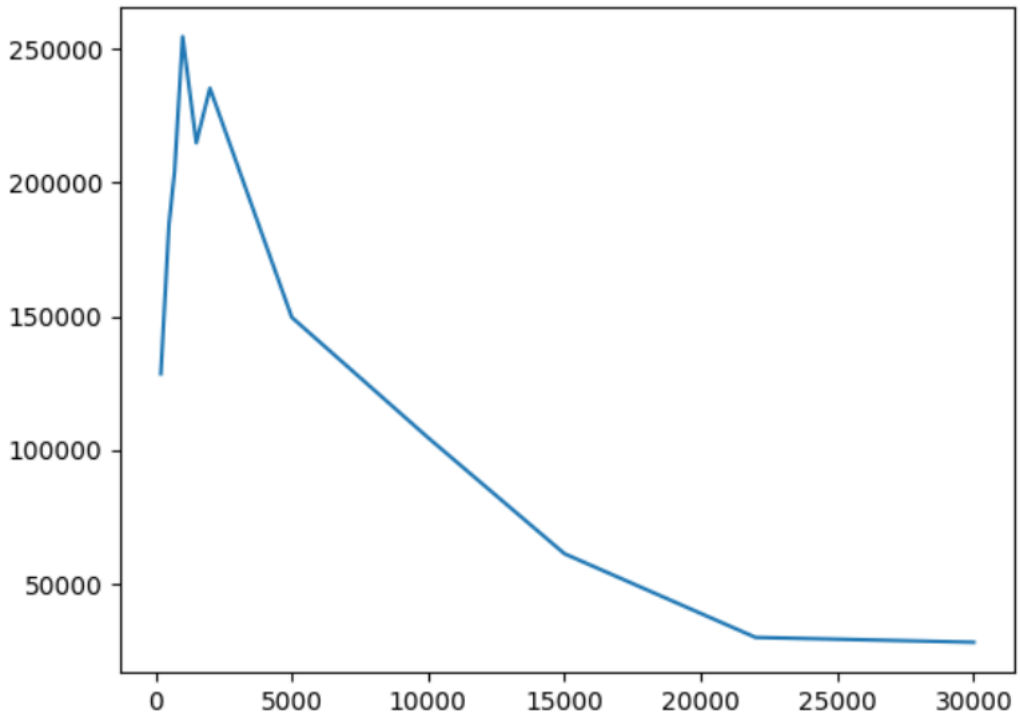

Figure 2: Results for the Triest-Impr algorithm

```
====================1000====================
BASE: Global T 1000: 0
BASE: Global triangle count with M 1000: 0
BASE: Spent time 0.182s.
IMPR: Global T with M 1000: 216982
IMPR: Spent time 0.251s.
====================1500====================
BASE: Global T 1500: 3
BASE: Global triangle count with M 1500: 110979
BASE: Spent time 0.45s.
IMPR: Global T with M 1500: 224381
IMPR: Spent time 0.364s.
====================2000====================
BASE: Global T 2000: 12
BASE: Global triangle count with M 2000: 179808
BASE: Spent time 0.49s.
IMPR: Global T with M 2000: 197926
IMPR: Spent time 0.546s.
====================5000====================
BASE: Global T 5000: 173
BASE: Global triangle count with M 5000: 129231
BASE: Spent time 1.884s.
IMPR: Global T with M 5000: 156992
IMPR: Spent time 1.899s.
====================10000====================
BASE: Global T 10000: 1932
BASE: Global triangle count with M 10000: 112056
BASE: Spent time 3.079s.
IMPR: Global T with M 10000: 101425
IMPR: Spent time 3.908s.
====================15000====================
BASE: Global T 15000: 6536
BASE: Global triangle count with M 15000: 65360
BASE: Spent time 4.149s.
IMPR: Global T with M 15000: 60373
IMPR: Spent time 3.761s.
====================22000====================
BASE: Global T 22000: 18329
BASE: Global triangle count with M 22000: 18329
BASE: Spent time 2.703s.
IMPR: Global T with M 22000: 29960
IMPR: Spent time 2.514s.
====================30000====================
BASE: Global T 30000: 28339
```

```
BASE: Global triangle count with M 30000: 28339
BASE: Spent time 0.165s.
IMPR: Global T with M 30000: 28339
IMPR: Spent time 0.315s.
```

As we can see from the results above, the improved algorithm gives a smaller range of estimations, which allows us to provide better estimations with smaller M.

4.1 Extra effort: Questions

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

   Triest algorithm is for undirected graph. Since we used a directed graph, we had to clean up the data, which was not obvious from the first sight. Before the clean-up, this algorithm gave highly inaccurate results.

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

   This algorithm can be parallelized by splitting the incoming stream into subsets. However, we should be extremely careful about dealing with shared values. Since there are a lot of shared values, that can become bottlenecks, the parallization benefits can be vanished by these bottlenecks.

3. Does the algorithm work for unbounded graph streams? Explain. This algorithm can be used for unbounded graph streams. Since we use a reservoir sampling, the algorithms always works with a fixed memory size of M. It allows us to utilize the available memory as we want. However, we would need to use bigger M values, which impacts performance.

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

   In this home assignment, we covered only Triest-Base and Triest-Impr algorithms, which are insertion-only algorithms, However, the mentioned paper also covers a Fully-dynamic algorithm, that can work with both insertions and deletions. According to the paper, the FD algorithm is based on random pairing (RP), a sampling scheme that extends reservoir sampling and can handle deletions. The idea behind the RP scheme is that edge deletions seen on the stream will be "compensated" by future edge insertions. Following RP, triest-fd keeps a counter $d_i(resp.d_o)$ to keep track of the number of uncompensated edge deletions involving an edge e that was (resp. was not) in S at the time the deletion for e was on the stream.

5 How to run

To run, you can access *test.py* class in our project. It only contains a main method, that calls methods for estimating a global triangle count. You can set various M values in the main method. You would need Python 3+ (we used 3.9) to start a program.

6 Conclusions

In this homework assignment, we got familiar with streaming graph processing methods and corresponding algorithm. We also implemented one of the suggested algorithm groups. The accuracy and time of this algorithm depend on the initial M values we set.