Introduction

In this assignment we were asked to implement reservoir sampling technique in the context of streaming graph data to count the number of triangles formed. We chose to read the paper by L. De Stefani called "TRIÈST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size" where the author presents an algorithm to sample graph data in order to estimate the global count of triangles.

The main idea of TRIEST base algorithm is to use reservoir sampling to have a sample of edges and to count the number of triangles formed in the sample. Then the global count of triangles is given by: $\zeta^t\tau^t$ where $\zeta^t = \max\left(1, \frac{t(t-1)(t-2)}{m(m-1)(m-2)}\right)$ and τ^t is the triangle count in the sample at time t. The authors also claim that if t<M where M is the size of the sample and t is the size of the stream, then the estimate of global triangles is *exact*.

Implementation

We created a class called Sample which has the following functions:

-addEdge (u,v)

- It adds the edge (u,v) to the Sample set S
- In the dictionary Neighbor it adds v as the neighbor of u
- In the dictionary Neighbor it adds u as the neighbor of v

-removeEdge

- Remove edge(u,v) from the Sample set S
- In the dictionary Neighbor remove v as the neighbor of u (if length of u is 0 also remove u from dictionary)
- In the dictionary Neighbor remove u as the neighbor of v (if length of v is 0 also remove u from dictionary)

Class Triest with functions:

-sampleEdge

- (same as TRIEST paper algorithm)

-updateCounters

- (same as TRIEST algorithm)
- Also removes local vertex counters once they reach 0.

-launchTriest

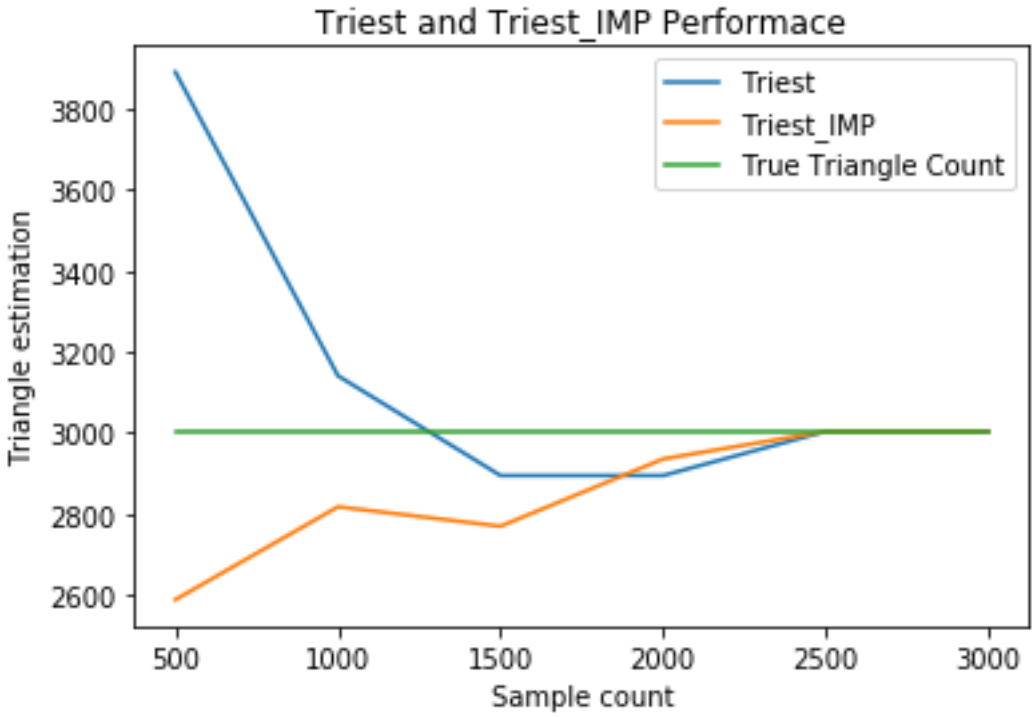
- This is the main function to run the Triest algorithm, input is the filepath to read.
- Reads line by line and does the first part of the algorithm of the TRIEST paper.

Class Triest_IMP

- Similar to TRIEST except the way counters are incremented are changed.

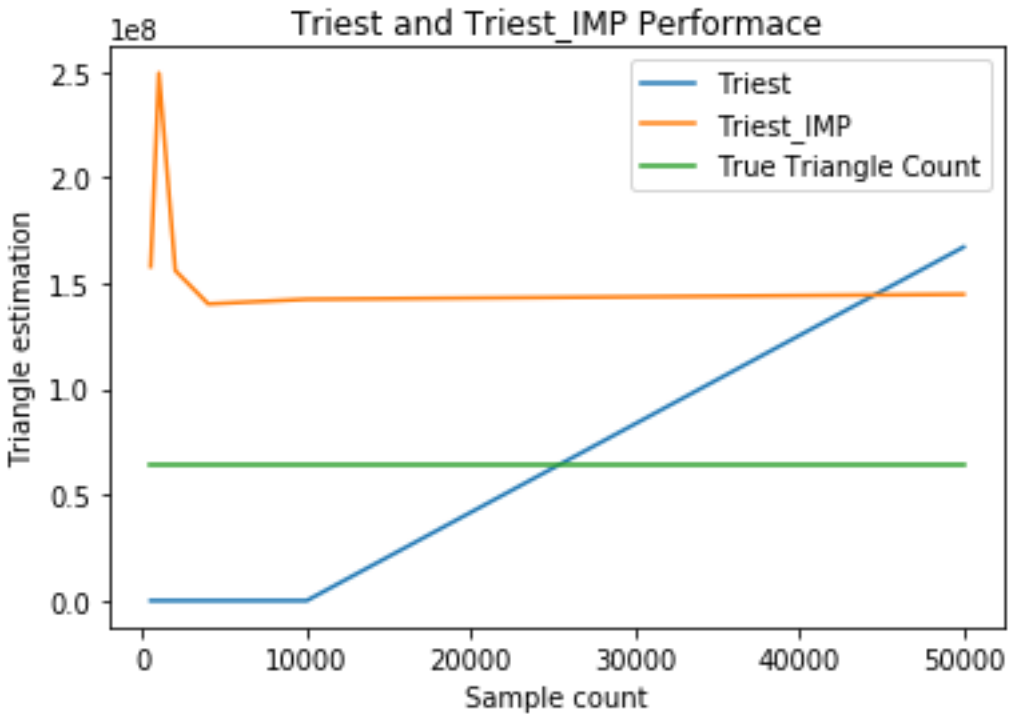
Results

We first tested the Triest and Triest improved algorithms on a small data set. This data set, web-polblogs.mtx only contains 2280 input edges and 3004 triangles can be formed (source http://networkrepository.com/web-polblogs.php) We can test the authors claim that if the sample size is bigger than the number of inputs then the triangle count will be *exact*. The results show that the author's claim is true.



Ms	500	1000	1500	2000	2500	3000
Count Triest	3890	3141	2895	2895	3004	3004
Count Triest improved	2589	2818	2770	2936	3004	3004

We then ran a much bigger data set called web-Berkstan.txt which contains 7,600,595 edge inputs and 64,690,980 triangles (source https://snap.stanford.edu/data/web-BerkStan.html). The results are below:



Ms	500	1000	2000	4000	10000	50000
Count Triest	0	0	0	0	0	167169842
Count Triest improved	157947273	249404886	156009783	140268151	142464498	144810801

Firstly, the results show that for a data set that is small (web-poblogs) the Triest algorithm manages to perform very well. But for a data set that is big (web-Berkstan) Triest does not perform well. The best error rate achieved was with Triest improved algorithm and this error was still 116% for a sample size of 4000.

Secondly as the authors claimed we found that Triest improved had a lower variance than Triest base algorithm. This observation can be seen from the graphs above.

Thirdly, the global estimate of triangles seems to converge to the around the same value for Triest and Triest base algorithm, as can also be seen from the graphs and tables above.

Bonus Questions:

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

- Challenge 1:

When we implemented the TRIÈST base algorithm, sometimes we got a very large number of global triangles, e.g.thousands of billions. But the instruction of the database said there are only several millions of triangles in this database. For example the web-NotreDame dataset have 8,910,005 of triangles (https://snap.stanford.edu/data/web-NotreDame.html).

Then we read the article again, and we found that according the TRIÈST base algorithm, at a certain time,

```
the number of global triangles = the triangles in sample / the probability a
triangle (a, b, c) of the global triangles is in the sample
```

As the time increasing, the probability a triangle (a, b, c) of the global triangles is in the sample will decrease according to the function

```
       Probability = (M*(M-1)*(M-2))/(t*(t-1)*(t-2)) (M is the size of
sample, t is the time)
```

If t is much larger than M, for instance t= 1,000,000, M= 10,000. The probability will be approximate 0.000001. And the number of triangles in the fixed size (M) sample will be relatively stable, because it is based on the connection between the points.

So we figured out that if we have a database which has a very large t(e.g. 1,000,000), and we we only take a relative small sample(e.g., 10,000), we will get a very large number of global triangles which is much large than the actual triangle number in this database and it is of course a bad estimation for the triangle counting.

This maybe one weakness of the TRIÈST base algorithm.

- Challenge 2:

When we implemented the TRIÈST base algorithm, sometimes we got zero global triangles as a result. But the instruction of the database said there are only several million triangles in this database. For example the web-Berkstan dataset has around 64 million triangles (https://snap.stanford.edu/data/web-BerkStan.html).

We found that if we have a too small size of sample compared to the number of points in the total database, for example the sample set {(1, 8), (1, 17), (2, 254915), (254913, 438238), (100, 2), (2, 5)} , we will actually get no triangle in the sample. So, it make sense that we get zero global triangles as the result.

But for the TRIÈST improved algorithm, by weighting the increment of local triangle counts by their global probability, we never get counts that are decreasing the estimate. Additionally since the increment is by at least a value of 1, we are almost guaranteed to have a non-zero estimate for global triangles.

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

We believe that the algorithm can be easily parallelized, by having multiple samples of the same graph data. These samples could be of smaller size then one bigger sample and perform equally well. By having multiple independent reservoir samples at the same time, the problem will be parallelized. Later multiple samples can be combined using their mean.

3. Does the algorithm work for unbounded graph streams?

The algorithm can work for unbounded graph streams, because we use reservoir sampling and biased random edge removal from the sample, so we can get a fixed size sample. And we also use a estimates function base on the incremental of each time step. In TRIÈST, the triangle number are exact number at time t<=M, and unbiased estimations for t > M. We do not need to re-run the algorithm from beginning after each update, which can save a lot of running time. Finally, TRIÈST uses estimation calculation instead of counting the exact triangle number in the sample which will spend a long time. So that no matter how big the dataset is, we can always get an estimation.

Additionally, Triest algorithms should work very well for unbounded graph streams because as t increases the probability to sample a new edge becomes very small. Thus, for a very large t, the algorithm, essentially, does nothing, making it very efficient. For both Triest base and Triest IMP the current estimate can be retrieved at any time, making it good for unbounded data.

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

As currently implemented Triest base and Triest_IMP are not suitable for edge deletions. An algorithm that supports edge deletions is called Triest_FD (fully dynamic). The main idea is to extend the reservoir sampling technique with a technique called Random Pairing (RP). "The idea behind the RP scheme is that edge deletions seen on the stream will be “compensated” by future edge insertions". So the Triest FD algorithm has additional counters d1 and d0 that keep track of uncompensated edge deletion that are in S and not in S respectively. These two new counters help determine whether an edge insertion should follow normal procedures as in Triest base or to insert an edge with a certain probability. Additionally, the final estimate of global triangles will be different from Triest base and Triest improved algorithms.