A Space Efficient Streaming for Triangle Counting using the Birthday Paradox

(Advanced Algorithms Final Project Report)

Sheetal Krishna Mohanadas Janaki (u1135144)
Chanchal Kariwala(u1134986), Greeshma Mahadeva Prasad(u1141804)

December 16, 2017

Description

Triangle counting has been an important problem in graph mining. Its real world applications include spam detection, link recommendations, finding common topics on the web etc. Transitivity ratio of a graph is a metric often used in complex network analysis. Computing this parameter is expensive because of the sheer size of the graphs.

In our project we have implemented the single-pass streaming algorithm and a baseline algorithm, that uses birthday paradox to make this operation more space efficient, https://users.soe.ucsc.edu/ sesh/pubs/conf-streaming-triangles.pdf. We ran experiments on smaller graphs like Facebook, Adbogato. Given that transitivity is a constant and the number of edges is greater than the number of wedges, the algorithms requires $O(\sqrt{n})$ space to provide good estimates of the number of triangles.

Implementation Details

We implemented the algorithm from the paper with a few changes.

Algorithm STREAMING TRIANGLES

procedure COUNT TRIANGLES(s_e, s_w)
    Initialize *edge_reservoir* and *wedge_reservoir* with sizes *s_e* and *s_w* respectively
    for For every edge *e_t* in the stream **do**
        Call *UPDATE* procedure
        ρ = fraction of entries set to *true* in *isClosed*
        transitivity = 3ρ
        triangle_count = $[\rho * t^2/s_e(s_e - 1)]$ · *total_wedges*

Algorithm UPDATE

procedure UPDATE(e_t)
    For every wedge in *wedge_reservoir*
    If *wedge_reservoir*$[i]$ is closed by e_t
    isClosed$[i] = true$
    Set *heads_probability* = $1 - (1 - 1/t)^{s_e}$ and flip a coin. if the result is tails continue with the next edge in the stream. If the result is heads proceed with the rest of the *UPDATE* procedure
    Choose a uniform index $i \in [s_e]$ and set *edge_reservoir*$[i] = e_t$
    Determine N_t, the number of wedges formed by the edge e_t and *new_wedges* = $|N_t|$
    Update *total_wedges*, the total number of edges formed by the updated *edge_reservoir*
    Calculate q = *new_wedges*/*total_wedges*
    Pick $(q * s_w)$ number of uniform indices $i \in s_w$
    for each i **do**
        Pick uniform random $w \in N_t$ with e_t in it
        Replace *wedge_reservoir*$[i] = w$
        Reset *isClosed*$[i] = false$

System Design and Components

- Implemented the algorithm using Python 2.7 for the above algorithm and Python 3.6 for the Baseline Algortihm.
- Maintained an edge reservoir using an adjacency list and wedge reservoir was maintained as a list of tuples(of edges). Usage of tuples to represents edges and wedges helped improve the run time.
- For a given edge, calculated the wedges it formed using the adjacency list of both the vertices of the edge.
- We modified the actual UPDATE algorithm, by changing the way the wedges formed by the new edge are placed in the reservoir. We replace the steps 7-11 from the original update method with the steps 10-14 in the above UPDATE method.
- In the steps 7-11, the original method iterated over the entire wedge reservoir and replaced each wedge with some probability(flipping a coin). In our implementation we calculated the expected number of such replacements and picked those many indices randomly for replacement.

Baseline Algorithm

As discussed we implemented the baseline algorithm,

- Calculated the number of wedges and number of triangles using Brute-Force and computed it's transitivity
- Computed the transitivity for a smaller set as shown in the algorithm below, we set the sizes of the reservoirs between $\sqrt{Number\ of\ edges} * 2$ and $\frac{Number\ of\ edges}{2}$ to keep the size reasonable when compared to the size of the graph.
- The aim was to calculate the number of triangles of the whole graph using the transitivity obtained by running the below algorithm and compare it with the actual number of triangles that we computed by Brute-Force. Also to compare the transitivities.

Algorithm

1: **procedure** BASELINE(*all_edges*)
2: Initialize *edge_reservoir_1* and *edge_reservoir_2* with sizes *s_1* and *s_2* respectively
3: Find *wedge_set*, all wedges formed by edges in *edge_reservoir_1*
4: **for** every edge $e_t \in$ *edge_reservoir_2* **do**
5: For every wedge, *wedge_set*[*i*] closed by e_t, set *isClosed*[*i*] to *true*
6: ρ = fraction of entries set to *true* in *isClosed*
7: *transitivity* = 3ρ
8: *triangle_count* = ρ * number of wedges in *wedge_set*

Performance and Results

```
2017-12-16 16:40:24.342000 : t = 72000, Kappa_t = 0.03, Traingle Count at t : 1193890.90909
2017-12-16 16:40:24.667000 : t = 74000, Kappa_t = 0.03, Traingle Count at t : 1216888.88889
2017-12-16 16:40:24.977000 : t = 76000, Kappa_t = 0.03, Traingle Count at t : 1283555.55556
2017-12-16 16:40:25.298000 : t = 78000, Kappa_t = 0.03, Traingle Count at t : 1352000.0
2017-12-16 16:40:25.620000 : t = 80000, Kappa_t = 0.03, Traingle Count at t : 1435151.51515
2017-12-16 16:40:25.931000 : t = 82000, Kappa_t = 0.03, Traingle Count at t : 1494222.22222
2017-12-16 16:40:26.240000 : t = 84000, Kappa_t = 0.03, Traingle Count at t : 1568000.0
2017-12-16 16:40:26.547000 : t = 86000, Kappa_t = 0.03, Traingle Count at t : 1643555.55556
2017-12-16 16:40:26.858000 : t = 88000, Kappa_t = 0.03, Traingle Count at t : 1720888.88889
2017-12-16 16:40:26.893000 : t = 88233, Kappa_t = 0.03, Traingle Count at t : 1730013.842
```

Dataset statistics	
Nodes	4039
Edges	88234
Nodes in largest WCC	4039 (1.000)
Edges in largest WCC	88234 (1.000)
Nodes in largest SCC	4039 (1.000)
Edges in largest SCC	88234 (1.000)
Average clustering coefficient	0.6055
Number of triangles	1612010
Fraction of closed triangles	0.2647
Diameter (longest shortest path)	8
90-percentile effective diameter	4.7

- The above image screenshots represent our program output and the actual Dataset statistics.
- As seen, our program could successfully predict the number of triangles for the Facebook dataset obtained from SNAP datasets. The predicted value was 1720888 which is very near to the actual dataset statistics value, 1612010.

Successes and Unexpected Events

- We were suggested to use the graphs mentioned in the paper such as DBLP, Stanford since they were comparatively smaller graphs. But, during implementation we observed that the computations were cumbersome. We tried different approaches to improve the run time but still it did not make much of a difference. Hence we chose to test our code on much smaller graphs like Facebook's.
- We could successfully fine-tune the size of edge and wedge reservoir by trial and error and studying it's behaviour.
- Although we implemented the baseline algorithm, we weren't able to get accurate values for the triangle count due to minor errors in counting the number of wedges closed by the available edges. However, we could get a good enough approximate transitivity value. It matched the value we obtained from the algorithm(0.03)

```
Transitivity of sketch graph: 0.017758008559503 51
 Transitivity of complete graph: 0.04162691992333 0535
```

References

TRIEST
Facebook Dataset from SNAP
Reading in Algorithms Counting Triangles
A space efficient streaming algorithm for triangle counting using the birthday paradox (Complete Paper)