Report 3: Homework Report Template

Georg Zsolnai, Oscar Reina

November 24, 2024

1 Introduction

The main goal of this assignment was to study and implement the streaming graph processing algorithm presented in the paper from M. Jha, C. Seshadhri, and A. Pinar, A Space-Efficient Streaming Algorithm for Estimating Transitivity and Triangle Counts Using the Birthday Paradox, ACM TKDD, 9-3, 2015.

For doing this, our assignment divided into two main challenges:

1. Implement the reservoir sampling algorithm used in the graph algorithm presented in the chosen paper.

2. Implement the streaming graph algorithm presented in the paper that uses the reservoir sampling algorithm describe in the first step.

2 Dataset

The following dataset has been used for testing the implementation of the assignment: `https://snap.stanford.edu/data/web-BerkStan.html`. The dataset contains nodes that represent pages from berkely.edu and standford.edu domains and directed eges represent hyperlinks between them from 2002. Once downloaded, the file named *web-BerkStan.txt* should be placed within the */data* folder.

3 Implementation

The algorithm has been implemented by using reservoir sampling and a streaming triangles graph technique to estimate triangle counts and the transitivity coefficient (κ) in the large dataset specified in Section 2.

For this, we have created a fixed-size edge reservoir (denoted as EDGE-RESERVOIR-SIZE) which has been maintained to make sure we get uniform edge sampling, and giving an unbiased representation of the stream. From this reservoir, we extract wedges and sample them for later store them in

another reservoir (controlled by probability). These wedges added are tested for closure (i.e., whether they form a triangle) as soon new edges arrive from the stream.

For clarification, a wedge is considered **closed** if all the edges forming it create a triangle, and **open** otherwise. The transitivity coefficient (κ) allows us to calculate the probability that a random wedge is closed. Each triangle has exactly one wedge whose closing edge appears in the future. This ensures that the fraction of future-closed wedges is approximated to one-third of all closed wedges (deriving the formula).

To estimate the number of triangles (closures) and the transitivity coefficient, we use the ratio of wedge closures:

1. **Estimated Triangle Count (T):**

$$T = (\frac{\rho \cdot t^2}{s_e(s_e - 1)}) \cdot \text{tot\_wedges}$$

2. **Transitivity Coefficient (κ):**

$$\kappa = 3 \cdot T/W$$

3.1 Parameters

For the completion of this assignment, we established a number of global variables or parameters. These are the following:

- *EDGE-RESERVOIR-SIZE = 1000*: sets a maximum size for the edge reservoir.
- *WEDGE-RESERVOIR-SIZE = 1000*: sets a maximum size for the wedge reservoir.

3.1.1 What if we had very short reservoir sizes?

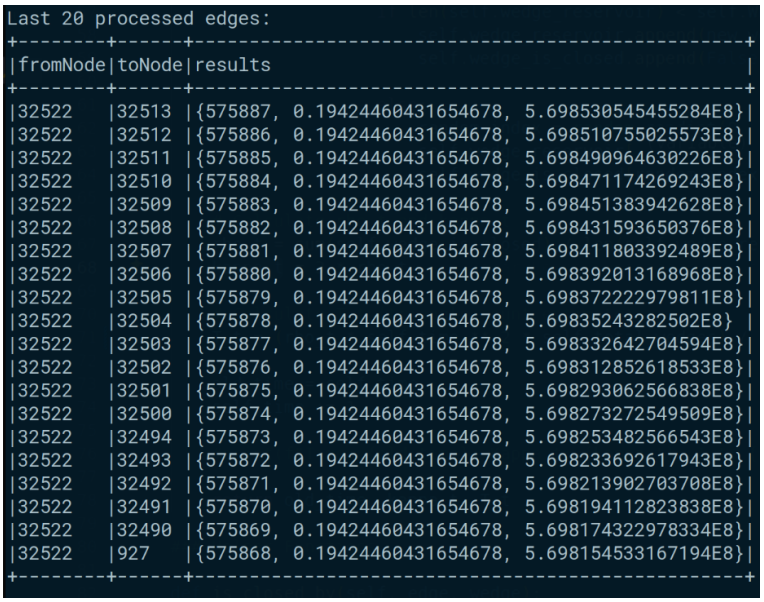
If we had set very short reservoir sizes, the algorithm would produce less accurate estimates of triangle counts since we can make very limited sampling. Though, processing would be faster as there would be less memory usage, we would be underestimating the total triangle count (we would be missing triangles). Therefore, we needed to weight the trade-offs between speed and accuracy.

3.1.2 What if we had very long reservoir sizes?

If we had very long reservoir sizes, the algorithm would output more accurate estimates of triangle counts and transitivity. However, as we discussed earlier this comes at cost of decreasing the overall performance, through needing of more memory resources.

4 Results

After the execution of the selected dataset in the implemented algorithm we could observe several aspects to be discussed about. This results can be seen in Figure 1, where we extracted the last 20 processed edges by Spark which shall contain the last metrics processed.



```
Last 20 processed edges:
+--------+------+--------------------------------------------------+
|fromNode|toNode|results                                           |
+--------+------+--------------------------------------------------+
|32522   |32513 |{575887, 0.19424460431654678, 5.698530545455284E8}|
|32522   |32512 |{575886, 0.19424460431654678, 5.698510755025573E8}|
|32522   |32511 |{575885, 0.19424460431654678, 5.698490964630226E8}|
|32522   |32510 |{575884, 0.19424460431654678, 5.698471174269243E8}|
|32522   |32509 |{575883, 0.19424460431654678, 5.698451383942628E8}|
|32522   |32508 |{575882, 0.19424460431654678, 5.698431593650376E8}|
|32522   |32507 |{575881, 0.19424460431654678, 5.698411803392489E8}|
|32522   |32506 |{575880, 0.19424460431654678, 5.698392013168968E8}|
|32522   |32505 |{575879, 0.19424460431654678, 5.698372222979811E8}|
|32522   |32504 |{575878, 0.19424460431654678, 5.69835243282502E8} |
|32522   |32503 |{575877, 0.19424460431654678, 5.698332642704594E8}|
|32522   |32502 |{575876, 0.19424460431654678, 5.698312852618533E8}|
|32522   |32501 |{575875, 0.19424460431654678, 5.698293062566838E8}|
|32522   |32500 |{575874, 0.19424460431654678, 5.698273272549509E8}|
|32522   |32494 |{575873, 0.19424460431654678, 5.698253482566543E8}|
|32522   |32493 |{575872, 0.19424460431654678, 5.698233692617943E8}|
|32522   |32492 |{575871, 0.19424460431654678, 5.698213902703708E8}|
|32522   |32491 |{575870, 0.19424460431654678, 5.698194112823838E8}|
|32522   |32490 |{575869, 0.19424460431654678, 5.698174322978334E8}|
|32522   |927   |{575868, 0.19424460431654678, 5.698154533167194E8}|
+--------+------+--------------------------------------------------+
```

Figure 1: web-BerkStan testing execution with reservoir sizes of 1000

Based on these results can extract the following metrics:

- Time / Total edges processed: The algorithm processed a total of over 1 million edges from the dataset which has been also used to identify our units of time.
- Transitivity Coefficient (κ): The calculated transitivity coefficient stabilized around 0.4895, indicating that about $0.4895/3 * 100 = 16.32\%$ of the possible triangles in the graph were closed.
- Estimated Triangle Count (T): The algorithm estimated approximately 2.55×10^9 triangles in the web-BerkStan graph.

These results show us that the algorithm can effectively synthesize the triangle structure within the dataset, giving us accurate metric estimates of both triangle counts and transitivity.

Additionally, it can be seen that as more edges were processed, the estimates for T and K became increasingly stable, which is due to the convergence behaviour expected from such streaming algorithms. This happens

due to the representative samples used that approach the population parameters.

Here are some other outputs that shows the randomness of Kappa and T values when running the algorithm multiple times see **Appendix**.

5 Bonus questions

5.1 What were the challenges you faced when implementing the algorithm?

Most of the challenges that appeared due to the lack of specification provided in the paper's pseudocode which required us to check additional sources and to understand why every variable or concept was needed. There were also few quirks regarding the implementation that needed to be discovered with testing on dummy data to fully realize. Furthermore, translating the mathematical formulas into code required also careful consideration and understanding of every step in the formula which added some complexity to the process, such as making sure to use the normalized formula in order to cause convergence over time. Finally, correctly implementing the reservoir sampling for both edges and wedges was a challenge. This is because they needed to be updated at very specific times and be consistent with each other.

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

No, the implemented algorithm cannot be easily parallelized because the implementation maintains state variables (edge and wedge reservoir) that need to be consistent across the iterations. These variables are updated for each incoming edge, and therefore would need of very efficient synchronization techniques across parallelizations / threads, which would add a lot of complexity. Additionally, making sure that each parallelized operation would update these global variables without any collisions would not amount to less improvements than just the single threaded application.

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

Yes, the implemented algorithm would work for unbounded graph streams. This is because it uses reservoir sampling with a fixed size for both edges and wedges. This makes it able to process an unbounded number of streams while using a constant amount of memory which is key for unbounded graph streams.

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

No, the current implementation of the algorithm does not support edge deletions. In order to add edge deletion logic we could modify the existing code in many ways:

- Firstly, every time and edge is deleted we would need to further update the edge reservoir.
- Secondly, this would involve also checking the wedge reservoir, to identify and remove the affected wedges. This could add significant overhead in the execution.
- Finally, if we update the wedge reservoir we would need to recalculate the triangle count and transitivity metrics to match the current state.

This would fundamentally change the outlined algorithm from the paper and would increase the complexity due to list operations to lookup and delete edges from the reservoirs. This could be curbed by using sets for the reservoirs but this would bring another set of changes to certain operations of the pre-exisitng algorithm. The final step in the bullet points would additionally add complexity and could result in further delay in the program's execution, especially with larger datasets.

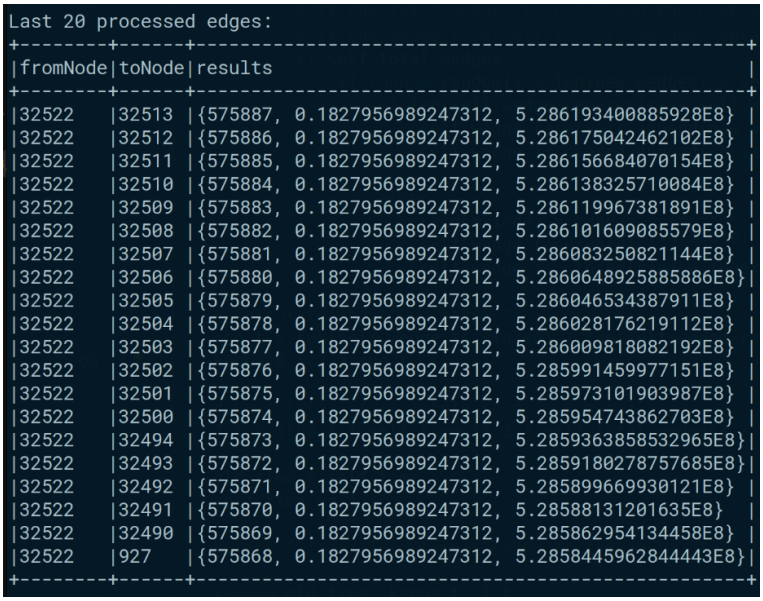
6 Setup and execution

To build and run this project, please follow these steps:

1. Ensure you have Python installed.
2. Make sure you are running a virtual environment to be able to install pip packages (or install pip packages globally at your own risk)
3. Download the dataset provided in Section 2 and place it within the */data* directory that you have to create in the root directory of the repository.
4. Once the dataset is downloaded and dependencies are installed, simply execute the following command:

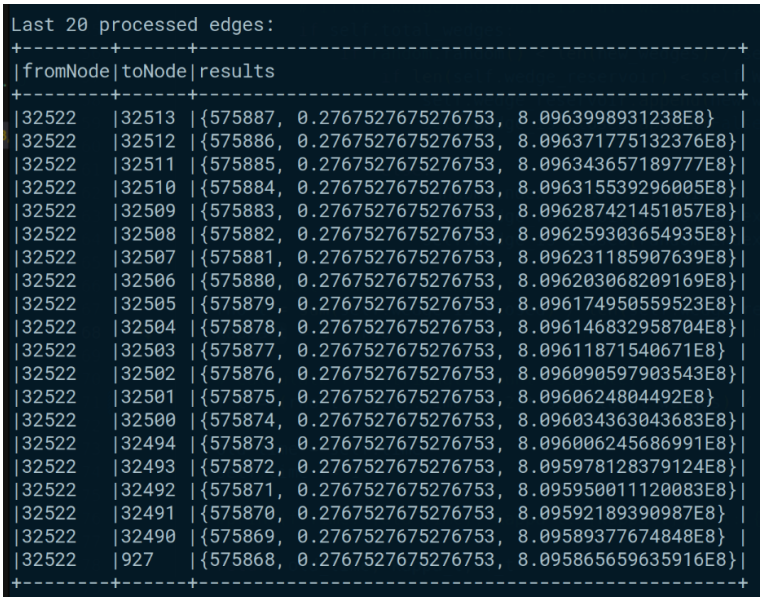
```
python streaming_triangles.py
```

7 Appendix



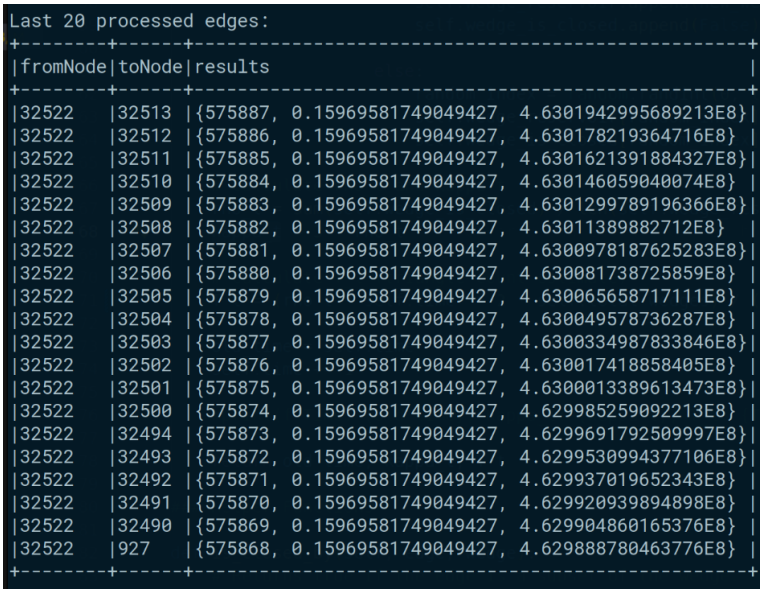
```
Last 20 processed edges:
+--------+------+-------------------------------------------------+
|fromNode|toNode|results                                          |
+--------+------+-------------------------------------------------+
|32522   |32513 |{575887, 0.1827956989247312, 5.286193400885928E8} |
|32522   |32512 |{575886, 0.1827956989247312, 5.286175042462102E8} |
|32522   |32511 |{575885, 0.1827956989247312, 5.286156684070154E8} |
|32522   |32510 |{575884, 0.1827956989247312, 5.286138325710084E8} |
|32522   |32509 |{575883, 0.1827956989247312, 5.286119967381891E8} |
|32522   |32508 |{575882, 0.1827956989247312, 5.28610160908557 9E8} |
|32522   |32507 |{575881, 0.1827956989247312, 5.286083250821144E8} |
|32522   |32506 |{575880, 0.1827956989247312, 5.2860648925885886E8}|
|32522   |32505 |{575879, 0.1827956989247312, 5.286046534387911E8} |
|32522   |32504 |{575878, 0.1827956989247312, 5.286028176219112E8} |
|32522   |32503 |{575877, 0.1827956989247312, 5.286009818082192E8} |
|32522   |32502 |{575876, 0.1827956989247312, 5.285991459977151E8} |
|32522   |32501 |{575875, 0.1827956989247312, 5.285973101903987E8} |
|32522   |32500 |{575874, 0.1827956989247312, 5.285954743862703E8} |
|32522   |32494 |{575873, 0.1827956989247312, 5.2859363858532965E8}|
|32522   |32493 |{575872, 0.1827956989247312, 5.2859180278757685E8}|
|32522   |32492 |{575871, 0.1827956989247312, 5.285899669930121E8} |
|32522   |32491 |{575870, 0.1827956989247312, 5.28588131201635E8}  |
|32522   |32490 |{575869, 0.1827956989247312, 5.285862954134458E8} |
|32522   |927   |{575868, 0.1827956989247312, 5.2858445962844443E8}|
+--------+------+-------------------------------------------------+
```

Figure 2: Iteration 1: web-BerkStan testing execution with reservoir sizes of 1000



```
Last 20 processed edges:
+--------+------+-------------------------------------------------+
|fromNode|toNode|results                                          |
+--------+------+-------------------------------------------------+
|32522   |32513 |{575887, 0.2767527675276753, 8.0963998931238E8}  |
|32522   |32512 |{575886, 0.2767527675276753, 8.096371775132376E8}|
|32522   |32511 |{575885, 0.2767527675276753, 8.096343657189777E8}|
|32522   |32510 |{575884, 0.2767527675276753, 8.096315539296005E8}|
|32522   |32509 |{575883, 0.2767527675276753, 8.096287421451057E8}|
|32522   |32508 |{575882, 0.2767527675276753, 8.096259303654935E8}|
|32522   |32507 |{575881, 0.2767527675276753, 8.096231185907639E8}|
|32522   |32506 |{575880, 0.2767527675276753, 8.096203068209169E8}|
|32522   |32505 |{575879, 0.2767527675276753, 8.096174950559523E8}|
|32522   |32504 |{575878, 0.2767527675276753, 8.096146832958704E8}|
|32522   |32503 |{575877, 0.2767527675276753, 8.09611871540671E8} |
|32522   |32502 |{575876, 0.2767527675276753, 8.096090597903543E8}|
|32522   |32501 |{575875, 0.2767527675276753, 8.0960624804492E8}  |
|32522   |32500 |{575874, 0.2767527675276753, 8.096034363043683E8}|
|32522   |32494 |{575873, 0.2767527675276753, 8.096006245686991E8}|
|32522   |32493 |{575872, 0.2767527675276753, 8.095978128379124E8}|
|32522   |32492 |{575871, 0.2767527675276753, 8.095950011120083E8}|
|32522   |32491 |{575870, 0.2767527675276753, 8.09592189390987E8} |
|32522   |32490 |{575869, 0.2767527675276753, 8.09589377674848E8} |
|32522   |927   |{575868, 0.2767527675276753, 8.095865659635916E8}|
+--------+------+-------------------------------------------------+
```

Figure 3: Iteration 2: web-BerkStan testing execution with reservoir sizes of 1000



```
Last 20 processed edges:
+--------+------+--------------------------------------------------+
|fromNode|toNode|results                                           |
+--------+------+--------------------------------------------------+
|32522   |32513 |{575887, 0.15969581749049427, 4.6301942995689213E8}|
|32522   |32512 |{575886, 0.15969581749049427, 4.630178219364716E8} |
|32522   |32511 |{575885, 0.15969581749049427, 4.6301621391884327E8}|
|32522   |32510 |{575884, 0.15969581749049427, 4.630146059040074E8} |
|32522   |32509 |{575883, 0.15969581749049427, 4.6301299789196366E8}|
|32522   |32508 |{575882, 0.15969581749049427, 4.63011389882712E8}  |
|32522   |32507 |{575881, 0.15969581749049427, 4.6300978187625283E8}|
|32522   |32506 |{575880, 0.15969581749049427, 4.630081738725859E8} |
|32522   |32505 |{575879, 0.15969581749049427, 4.630065658717111E8} |
|32522   |32504 |{575878, 0.15969581749049427, 4.630049578736287E8} |
|32522   |32503 |{575877, 0.15969581749049427, 4.6300334987833846E8}|
|32522   |32502 |{575876, 0.15969581749049427, 4.630017418858405E8} |
|32522   |32501 |{575875, 0.15969581749049427, 4.6300013389613473E8}|
|32522   |32500 |{575874, 0.15969581749049427, 4.629985259092213E8} |
|32522   |32494 |{575873, 0.15969581749049427, 4.6299691792509997E8}|
|32522   |32493 |{575872, 0.15969581749049427, 4.6299530994377106E8}|
|32522   |32492 |{575871, 0.15969581749049427, 4.629937019652343E8} |
|32522   |32491 |{575870, 0.15969581749049427, 4.62992093989489 8E8} |
|32522   |32490 |{575869, 0.15969581749049427, 4.629904860165376E8} |
|32522   |927   |{575868, 0.15969581749049427, 4.629888780463776E8} |
+--------+------+--------------------------------------------------+
```

Figure 4: Iteration 3: web-BerkStan testing execution with reservoir sizes of 1000