Report 3: Mining Data Streams

Group 17: Jingmeng Xie, Yuning Liu

Selected Paper2: TRIÈST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size

1. Introduction

Objective: To calculate the number of "triangles" (i.e., the relationships where "a friend of a friend is also a friend") in a graph data stream with a large volume of data and continuous updates (such as real-time dynamics of social networks).

Challenge: There is too much data, and it cannot all be stored in memory.

Solution: Use the TRIÈST algorithm. Its core concept is reservoir sampling - only retain a fixed number `(M)` of random edges in memory, and use this portion of the sample to estimate the number of triangles in the entire graph.
We have implemented two versions: BASE and IMPR. Currently, we have implemented the insertion of edges into a dynamic sample set `(S)`. We will discuss how to allow edges to be inserted/deleted in an arbitrary adversarial order in a fully dynamic stream in the future.

2. Instructions

2.1 Build

- Python 3.9
- Git:

```
git clone https://github.com/Esther979/DataMining3-MiningDataStream
```

2.2 Dataset

This dataset consists of 'circles' (or 'friends lists') from Facebook. Facebook data was collected from survey participants using this Facebook app. The dataset includes node features (profiles), circles, and ego networks.

```
facebook_combined.txt
```

Data content: The friendship relationships among users.

Total number of edges: 88,234.

Total number of real triangles: 1,612,010.

Source Website: https://snap.stanford.edu/data/ego-Facebook.html

2.3 Execution

```
python3  Triest.py
```

3. Solution

3.1 Reservoir sampling

Reservoir sampling is the core mechanism that allows TRIÈST to operate under strict memory constraints.
Since the complete edge stream may be arbitrarily large, it is impossible to store all edges in memory.
Instead, we maintain a fixed-size sample set **S** of at most **M** edges, chosen uniformly at random from all edges seen so far.

Goal:
The objective of Task1 is to implement this sampling layer:
· Maintain a reservoir of size **M**
· Ensure each edge among the first **t** processed edges is included with probability **M/t**
· Keep an adjacency list that reflects exactly the edges currently stored in the reservoir

Method:

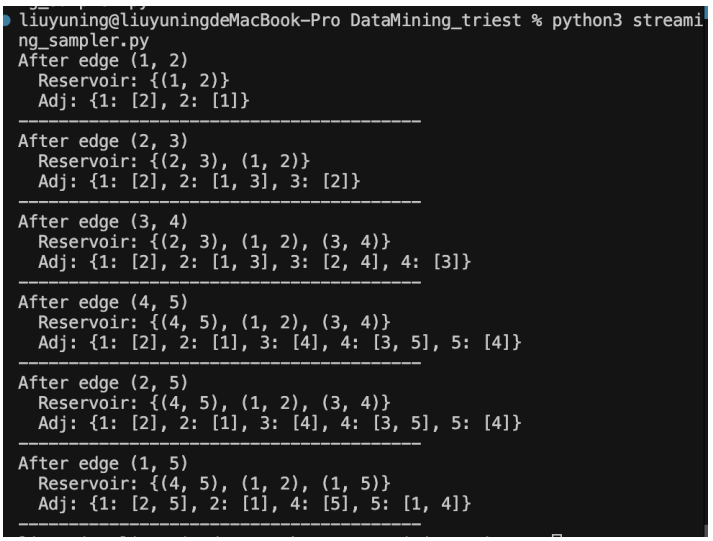
1. Maintain:
   · t: number of processed edges
   · S: the reservoir (a set of at most M sampled edges)
   · Adj: adjacency list for the sampled graph
2. For every incoming edge (u,v):
   · If $t \leq M$, simply insert the edge into S and update Adj
   · If $t > M$, accept the new edge with probability:

$$\frac{M}{t}$$

      · If the edge is not accepted -> ignore it
      · If accepted:
         · Uniformly remove one existing edge from S
         · Insert the new edge into S
         · Update the adjacency list accordingly

   This ensures that each edge in the prefix of length t has the same probability M/t of being in the reservoir, achieving uniform sampling without storing the entire stream.

Example Output:
A small test example verifies the correctness of the sampling behavior:



The reservoir always contains at most M edges, and the adjacency list is updated consistently, validating the sampling logic.

3.2 TRIÈST-BASE

TRIÈST-BASE is the straightforward application of Reservoir Sampling for triangle counting. It detects and counts triangles only when an edge is added to the sample (S) or removed from S.
Insertion: If a new edge (u, v) is sampled, the algorithm calculates the number of shared neighbors w in S and increments the global counter by this amount.
Deletion: If an old edge is randomly selected for replacement and removed, the counter is decremented by the number of triangles that edge formed in S.
Finally, multiply the count by an amplification factor (since only a small portion of the data was stored).

Cons: If the memory is limited, it is difficult to gather all three sides in the sampling, resulting in significant fluctuations in the outcome (large variance).

3.3 TRIÈST-IMPR

TRIÈST-IMPR is an optimized variant designed to significantly reduce the estimation variance by utilizing more stream history.
Unconditional Pre-Update: When a new edge arrives, first compare it with the edge in the pool to calculate how many triangles can be formed. For every incoming edge (u, v), the algorithm unconditionally calculates the triangles it forms with the current sample S, before deciding whether to sample the edge.
Weighted Update: The count is not increased by 1, but by a time-varying weight.
Non-Decremental Counter: The algorithm never decrements the counter, even when an edge is removed from S.

Estimation: Due to the weighted accumulation strategy, the counter already contains the corrected, scaled value, and the final estimate is simply the counter. This version is shown to have a much smaller variance, leading to higher accuracy in practice.
Pros: Even with a small amount of memory, it is capable of using each edge that passes through to update the information, resulting in extremely accurate results.

4. Bonus

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

1) Integrating the counting logic with the core Reservoir Sampling logic:
   Using class inheritance (StreamingGraphSampler) and override the add_edge and remove_edge methods, allowing the counting logic to set into the sampling process without redundant data structure copying.

2) Ordering: When implementing the IMPR version, it is essential to strictly follow the sequence of "first update the counts using the new edge, and then decide whether to add the new edge to the sample", otherwise the result will be too low.

3) Maintaining consistency between the reservoir and the adjacency list:
   When an old edge is removed from the reservoir, its two endpoints must also be correctly removed from the adjacency list.
   A small mistake here leads to incorrect neighbor sets, which then produces incorrect triangle counts.
   Debugging this required printing the entire adjacency structure after each update.

4) Handling edge duplicates and self-loops:
   Real-world SNAP datasets contain repeated edges, isolated nodes, and occasionally malformed lines.
   We had to ensure that duplicates do not enter the reservoir and that self-loops (u,u) are ignored, otherwise the triangle counting logic would break.

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

No.
Reason: The algorithm is highly dependent on the **global state**.
Global counter t: Determines the sampling probability and must strictly increase in sequence.
Sampling set S: If multiple threads simultaneously read and write to the same fixed-size sampling set, conflicts will occur.
In addition, the adjacency list must remain perfectly synchronized with the counting process. Any concurrent update may cause missing or extra neighbors, which directly breaks the triangle counting process.

To parallelize the algorithm safely, a global lock would be required, but this would serialize the entire computation and defeat the purpose of parallelization.

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

Yes.
Reason: The memory usage of the algorithm is locked at `M`. No matter how many billions of data entries come in, the memory will not overflow. The algorithm will **automatically adjust** the weights to ensure that the estimation is unbiased.
Corrective Scaling: As the stream grows, the sampling probability M/t approaches zero. The core mechanism prevents the estimate from collapsing by applying the increasing scaling factors, which ensures the estimate remains unbiased and valid across the entire lifespan of the stream.

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

No. To handle fully-dynamic streams (insertions and deletions), the algorithm must be replaced by **TRIÈST-FD (Fully-Dynamic)**, which is based on the Random Pairing (RP) sampling scheme:
TRIÈST-FD (Fully-Dynamic): Introduce the "random pairing" mechanism. When a deletion operation occurs in the stream, use future insertion operations to "compensate" for the gap, maintaining the randomness of the sample.
Deletion Handlers: When an edge is deleted from the stream, the algorithm does two things:

If the deleted edge was in S, it is removed, and the counter di (in-sample deletions) is incremented.
If the deleted edge was not in S, the counter do (out-of-sample deletions) is incremented.
Compensatory Insertion: When a new edge is inserted, it first checks $di + do$. If positive, the new edge is used to "compensate" for a past deletion with probability $\frac{di}{di+do}$. This compensatory mechanism maintains the fixed-size reservoir's properties under dynamic operations.

5. Evaluation

We compared the error rates (MAPE) under different memory sizes:

```
esther@estherdeMacBook-Air DataMining % /usr/bin/python3 /Users/esther/coding/Python3.9/DataMining/HW3/Triest.py
Loading the dataset: HW3/facebook_combined.txt ...
Total edges: 88234
Calculating Ground Truth ...
Ground Truth (The number of real triangles): 1612010 (Time cost: 0.2226s)
-------------------------------------------------------------------------------------------
Algorithm       | M        | Sample %    | Estimate     | MAPE (%)    | Time (s)
-------------------------------------------------------------------------------------------
TRIÈST-BASE     | 1000     | 1.1      % | 2066894      | 28.22       | 0.0454
TRIÈST-IMPR     | 1000     | 1.1      % | 1453590      | 9.83        | 0.0819
TRIÈST-BASE     | 5000     | 5.7      % | 1605560      | 0.40        | 0.3760
TRIÈST-IMPR     | 5000     | 5.7      % | 1626497      | 0.90        | 0.5198
TRIÈST-BASE     | 10000    | 11.3     % | 1654550      | 2.64        | 1.1229
TRIÈST-IMPR     | 10000    | 11.3     % | 1619861      | 0.49        | 1.0913
TRIÈST-BASE     | 20000    | 22.7     % | 1634979      | 1.42        | 3.9210
TRIÈST-IMPR     | 20000    | 22.7     % | 1595270      | 1.04        | 3.9867
TRIÈST-BASE     | 40000    | 45.3     % | 1620334      | 0.52        | 9.2910
TRIÈST-IMPR     | 40000    | 45.3     % | 1614701      | 0.17        | 9.2681
esther@estherdeMacBook-Air DataMining %
```

- When the memory capacity is extremely limited (only storing 1% of the data), the performance of TRIÈST-IMPR is awesome, with an error rate of around 10%, while the error rate of the BASE version is close to 30%. The two are quite different.
- As the memory capacity increases, the error rates of both will become very small and approximately the same.

6. Conclusion

The TRIÈST framework successfully solved the problem of triangle counting under memory constraints.
TRIÈST-IMPR is the best choice. It significantly reduces estimation errors by improving the update strategy.
For massive and high-speed graph data streams, using the IMPR algorithm can achieve high-quality statistical results with minimal memory cost.