HW3: Mining Data Streams

Authors: Andrei Capastru, Alex Porciani | Group 11

The task is to implement the reservoir sampling and then implement one of the papers algorithm. We chose to implement TRIEST algorithm from **TRIEST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size** (De Stefani L., et al. 2017).

1.1 How to run the code

0. Install python3 and install Jupyter Notebook
1. Run in unix terminal:

```
cd ~
git clone https://github.com/andreicap/data-mining.git
cd data-mining/HW3
jupyter notebook
```

2. Open the browser on `http://localhost:8888` and run the `TRIEST.ipynb` file

1.2 Reservoir sampling

In this part of the code we do **Reservoir Sampling** - which is a randomized sampling of specified size. We have `M` as the specified memory size and `i` is the specific element in the stream of edges.

Do continuously while streaming edges:

1. Keep first `M` items in memory.
2. With Probability (M/i) we keep the i^{th} item and discard one of the previous ones with the same probability.

1.3 TRIEST Algorithm

We use TRIEST algorithm to approximate the global and local number of triangles in a streamed graph.

Do continuously while streaming edges:

1. If the received edge is not currently in the edge store, we do reservoir sampling and then process the edge, otherwise we skip the edge and do not process it.
2. For every processed edge we call a counters' update:
    - check the common neighbours of the vertices of the current edge (intersection of their neighbours sets);
    - we compute the update weight using this formula: $max\{1, \frac{(i-1)(i-2)}{M(M-1)}\}$;
    - update local counters for each vertex and also the global triangle counter.
3. Return the number of global triangles.

1.4 Questions

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

   A: We faced a bit of difficulty in implementing the improved TRIEST algorithm over the base one, because we implemented only the improved version and there is no pseudocode for it.

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

   A: Yes, we can parallelize the edge processing and counters update easily, as we use global counters and shared edge store. Both phases: reservoir sampling and TRIEST with counters update can be computed in parallel, using shared memory.

3. Q: Does the algorithm work for unbounded graph streams? Explain.

   A: Yes, it is an online algorithm. Reservoir sampling considers only a specified memory size and only current items. Furthermore, the idea of reservoir sampling is to mine potentially infinite streams of data, without taking into account the stream length. TRIEST dynamically updates the counters of global and local triangles but processing each edge independently and in relation with only the ones present in the memory.

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

   A: We implemented the improved version of TRIEST-base that does not support edge deletions. In order to support it, a modification of TRIEST-base is needed, and specifically it needs a tracker that monitors the uncompensated edge deletions and decreases the number of triangles if necessary. It also needs modification on reservoir sampling to support edge deletions in the dataset, by tracking if the edge-to-be-deleted is in the current edge list or not, computing a different probability of insertion. If the edge-to-be-deleted was in the sampled list, the algorithm will compensate the deletion with an immediate insertion. These modification were added in TRIEST-FD which is a fully-dynamic version of the improved TRIEST-Base.