Data Mining - ID2222

Homework 3

Mining Data Streams

André Silva, Jérémy Navarro

November 23, 2020

1 Overview

We chose *Python* as the programming language and an OOP approach for the development of the project.
We decided to implement the *fully-dynamic* algorithm of the TRIÈST algorithm suite [3].

Information about the structure of the project, as well as instructions on how to run it follow:

```
$ unzip homework3.zip
$ cd homework3/
$ python3 main.py
```

The project is composed by:

- `classes.py`: Contains the classes implementing the chosen algorithm.
- `main.py`: Executable script that uses the algorithm classes on the dataset.
- `facebook_combined.txt`: Social circles: Facebook [2].

2 Classes

2.1 FullyDynamic

The class `FullyDynamic` takes 1 argument:

- `M`: Number of edges to keep as a sample.

The class initializes sets and dictionary use to estimate the triangle in the graph streamed.

Then, for every new event of the stream, a call to the method `sample` is necessary. This takes 1 argument, an array representing an edge.

Then the algorithm performs as describe in TRIÈST paper [3].

3 Results

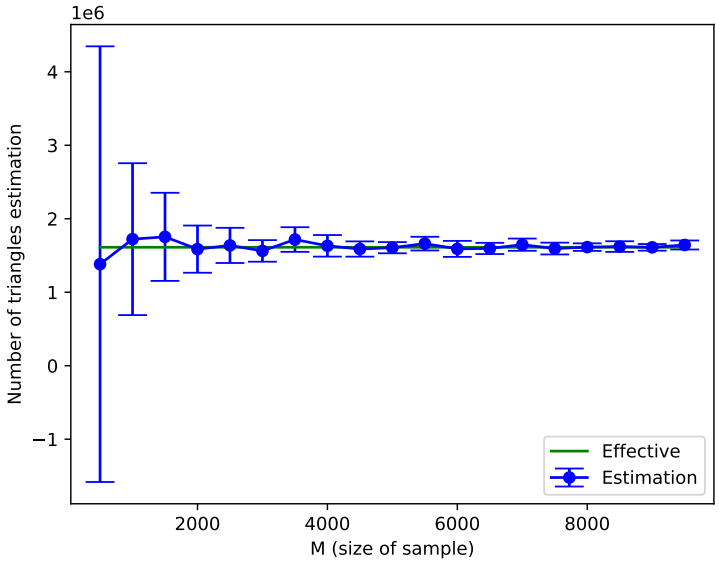

Figure 1: Estimated number of triangles given a sample size M. Points represent means, error bars 2σ, of 20 observations.

4 *Bonus*

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

The main challenges encountered where in the choice of data structures, such that would allow for the algorithm to run efficiently.

To represent the graph sample we opted for a set, containing edges, and an adjacency list for each node, stored in a dictionary, which allows us for $O(1)$ access to neighborhoods.

To represent the local counters we opted for a dictionary.

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

Yes. The main problem with parallelism is concurrent memory access. Indeed, this process can be easily parallelized with a mechanism of mutual exclusion.

This being said, we would be losing a lot on efficiency, as our algorithm depends heavily on accessing shared data structures (see 4.1), such as the main sample and local, vertex-wise, counters.

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

Yes. As we use random pairing, an extension of reservoir sampling, we only maintain a fixed size sample over the course of the algorithms execution. This is the main advantage of using this type of sampling, as opposed to Bernoulli sampling, where we would keep items with a probability of p, making the sample size dependent on the size of the stream.

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

Yes. By using random pairing [1], instead of reservoir sampling, we can handle deletions. This is achieved by making new edge insertion compensate previous deletions.

References

[1] Rainer Gemulla, Wolfgang Lehner, and Peter Haas. "A Dip in the Reservoir: Maintaining Sample Synopses of Evolving Datasets." In: Jan. 2006, pp. 595–606.

[2] *Social circles: Facebook*. `https://snap.stanford.edu/data/ego-Facebook.html`. accessed: 2020-11-23.

[3] Lorenzo De Stefani et al. *TRIÈST: Counting Local and Global Triangles in Fully-dynamic Streams with Fixed Memory Size*. 2016. arXiv: `1602.07424 [cs.DS]`.