KTH - Kungliga Tekniska högskolan - DATA MINING 2017

Assignment 3: Mining Data Streams

Filippo Boiani <boiani@kth.se>
Riccardo Sibani <rsibani@kth.se>

19 Nov 2017

NOTE
The code can be found at the following link: https://github.com/filippoboiani/data-mining

Introduction

Implemented algorithm:
TRIÈST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size.

How to Run

- via console, open the assignment directory
- run the following set of commands

```
cd mining_data_streams_jar
java -jar mining_data_streams.jar <file name> <sample length>
```

Command Line Parameters

NOTE: the file must be download from this link facebook-wosn-links and placed in the input folder. We didn't included it in the delivery as it's too big.

The .jar file accepts either 2 or no parameter.

NO PARAMETER:

```
java -jar mining-data-streams.jar
```

Defaults:
- file: out.facebook-wosn-links
- sample size:

WITH PARAMETERS:

```
java -jar mining-data-streams.jar  out.facebook-wosn-links 150000
java -jar mining-data-streams.jar <filename> <sample size>
```

Approach

We decided to implement the algorithm proposed in the second paper: L. De Stefani, A. Epasto, M. Riondato, and E. Upfal, TRIÈST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size.

The algorithm works on undirected graph and uses reservoir sampling to keep the number of elements in the sample fixed. To test the algorithm, we took a file representing the Facebook Friendships. The file has the characteristics listed below and can be retrieved at the following link: facebook-wosn-links

Parameter	Value
Vertex Type	User
Edge Type	Friendship
Format	Undirected
Size (number of edges)	**63,731 vertices (users)**
Volume (number of edges)	**817,035 edges (friendships)**
Triangle count	**3.500.542**

The project has the following classes:
- **Main**: the main class reads the input parameters (or the default ones), instantiate the **TRIEST (or TRIEST_IMPR)** class with these parameters and calls its method for every element in the steam. The stream is created by reading the graph file sequentially.
- **TRIEST**: is class representing the first algorithm. It exposes 2 methods the **`analyse(boolean addition, int[] edge)`** and **`getGlobalCounters()`**. The method **`analyse`** is called for every element of the stream. In our case, the element is an (undirected) edge representing a friendship on Facebook. The method **`getGlabalCounters`** can be called at any point in time (i.e. by a concurrent thread).
- **TRIEST_IMPR**: this class represent the improvement of the algorithm, proposed by the authors of the paper.

Bonus Point Questions:

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

The main challenges were mostly related to the optimisation of memory utilisation. We had to prefer arrays over ArrayList and try to avoid creation of new instances as much as possible. The size of the sample was of paramount importance as a huge number would have increased the occupied memory and the running time.

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

In our opinion the algorithm would be hard to implement in parallel. The main reason is that the algorithm keeps a global set of sampled edges and local counters (and . This counters, as well as the sample must be broadcasted to the entire cluster of parallel nodes. Otherwise, the sample can be split among the parallel computing nodes as well as the local counters for that sample split. Then the reservoir sampling could take a random node in the cluster and that node would take a random edge from its split. The problems arise when it comes to delete edges because the two vertexes could be location in different parallel nodes.

Does the algorithm work for unbounded graph streams? Explain.

The algorithm works for unbounded graph streams. This is possible thanks to the sampling technique and the estimation used. The state of the algorithm can be queried at any point in time to get an estimation of the number of edges at time t.

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

The first version of the algorithm supports only insertions. In order to support deletions the paper takes advantage of a sampling technique called random pairing which is an extension of the reservoir sampling. This technique requires to keep two additional counters, respectively for uncompensated deleted edges and for deletions of edges that are not in the sample. This two counters are kept in consideration when the number of triangles is requested at time t.

Results

The number of total triangles should be 3.500.542.

Out tests on different sample sizes resulted in different running times and different estimation accuracies. We noticed that the variance of the estimation was reducing with the sample size.

TRIEST

```
sample size: 100000
Process Finished after 10.0 seconds.
Estimation: 3463423
Error: 1,06 %

sample size: 150000
Process Finished after 14.0 seconds.
Estimation: 3477569
Error: 0,65 %
```

TRIEST IMPR

```
sample size: 100000
Process Finished after 11.0 seconds.
Estimation: 3478211
Error: 0,63 %

sample size: 150000
Process Finished after 15.0 seconds.
Estimation: 3519248
Error: 0,53 %
```

TRIEST IMPR returns better estimations by computing it each time a new element arrives. This estimation has lower variance.