Data Mining: Similar Objects

Braulio Grana Gutiérrez Adrián Ramírez del Río

December 4, 2016

1 Description

For this assignment we implemented the graph streaming algorithm StreamingTriangles. We implemented said algorithm using the Scala programming language and no framework.

The objective of the algorithm is to estimate the number of triangles in a graph stream given the fact that you (obviously) do not have access the whole dataset at once. StreamingTriangles is able to estimate the number of triangles using the birthday paradox.

1.1 Implementation

To implement the algorithm we developed a Scala class named *StreamingTriangles*, which has two public function: *getEdgeStream* to get the stream from the file, and *execute* that runs a step of the algorithm. We also built a *Main* object in which we instantiate the previous class, get the stream and then call *execute* iteratively every time we get an item from the stream.

The following structures were used to implement those specified in the paper:

edge_res
: An array of reservoir edges that is at the same time a subgraph of the stream. The size of this array is decided by an input parameter called *size_edge_res*.

wedge_res
: An array of reservoir wedges which size is decided by the input parameter *size_wedge_res*.

is_closed
: An array of booleans of the same size as *wedge_res* indicating with a *true* if the wedge on a given index is considered closed.

tot_wedges
: The number of wedges contained in the current *edge_res* array.

Nt or new_wedges
: List of all wedges containing the edge received in step t.

Apart from these, other flags and counters are used to maintain the state of the class.

Each step of the algorithm works as follows:

1. Update the structures
   (a) Update the structure edge_res using reservoir sampling
   (b) If *edge_res* was updated, update *wedge_res* and *is_closed*
2. Calculate ρ, which is the fraction of *is_closed* set to true
3. Calculate the transitivity, which is 3ρ
4. Calculate the estimate of triangles using the following formula:

$$\rho \cdot iter/(size\_edge\_res \cdot (size\_edge\_res - 1)) \times tot_w edges$$

2 Instructions

To execute this project, go to the project's root folder and run the following commands:

```
cd src
sbt "run-main Main path/to/dataset size_edge_res size_wedge_res"
```

Where *path/to/dataset* is a string containing the path to the graph dataset to be used, *size_edge_res* is the size of the edge reservoir array and *size_wedge_res* is the size of the wedge reservoir array.

3 Questions

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

We had to struggle with Scala programming language since we are not really used to it. Some parts of the paper remain unclear or are misleading:

- The formula to compute the estimation of the number of triangle has a typo.
- It's not clear if the variables tot_wedges and Nt must consider repetitions or ignore them.
- It is not stated how to pick the size of the reservoirs (input parameters) since all their experiments are run with a fixed value (20k for both parameters). However, performing our tests with smaller graphs that presented higher transitivities we have found that the algorithm is very sensitive to these parameters' values.
- It's hard to tell whether the implementation is correct or not since the output of the algorithm are estimations.

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

Yes. A lot of the operations are naturally parallelizable (as filters or maps). For example, in the update function, lines 1-3, 4-7 and 11-16 can be written as map operations since these for loops just go through the arrays updating them but each iteration is independent from the rest.

These kind of operations can be implemented in distributed computing frameworks such as Apache Spark.

Does the algorithm work for unbounded graph streams? Explain.

Yes, since it keeps a snapshot of the graph (the edges reservoir gives the algorithm a partial picture of the graph, a subgraph) and the memory it needs is held constant. However, the longer the algorithm is run, the less updates it performs, so there will be a point in time when the algorithm has already converged and new edge insertions will rarely update the estimated values. Thus the transitivity must be constant (if it changes a lot over time, the algorithm won't be able to perceive it).

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

It doesn't support edge deletion in the sense that an edge that the algorithm has already seen might no longer be kept in the reservoir, and the information that the algorithm took from it can not be erased. However, some kind of edge deletion can be performed for edges that are still in the reservoir, for example, overwriting them by another edge in the reservoir selected at random.

Nonetheless, it would be needed to study further how this kind of edge deletion affects the algorithm from the theoretical point of view.