DataMining ID2222 - Homework 3
Mining Data Streams

Sherly Sherly and Anna Martignano

November 25, 2019

1 Introduction

In this project, we will study and implement a streaming graph processing algorithm, Trièst [1]. We will be implementing the algorithm in these 2 steps:

1. Implement the reservoir sampling in the graph algorithm presented in the Trièst paper.
2. Implement the streaming graph algorithm with the algorithm implemented in the first step.

The project is organised in the following settings: **Section 2** describes the dataset used to implement and test the algorithms, **Section 3** describes the Trièst algorithms and in more detail, the two variants of Trièst implemented in this project, **Section 4** describes how to run the codes and lastly, **Section 5** is a discussion section for the follow-up questions to the implementations.

2 Dataset

The dataset used in this project is the Air traffic control network dataset. This network was constructed from the USA's FAA (Federal Aviation Administration) National Flight Data Center (NFDC), Preferred Routes Database. Nodes in this network represent airports or service centers and links are created from strings of preferred routes recommended by the NFDC.

Details	
Vertex type	Airport/service center
Edge type	Preferred route
Format	Directed: Edges are directed
Edge weights	Unweighted: Simple edges Unweighted
Size	1,226 vertices (airport/service centers)
Volume	2,615 edges (preferred routes)

3 Trièst

Trièst (TRIangle Estimation from STreams) is a suite of one-pass streaming algorithms to approximate, at each time instant, the global and local number of triangles in a fully-dynamic graph stream (i.e., a sequence of edges additions and deletions in arbitrary order) using a fixed amount of memory. Trièst only requires the user to specify the amount of available memory, an interpretable parameter that is definitively known to the user.

Parameters. Trièst algorithms keep an edge sample S of up to M edges from the stream, where M is a positive integer parameter.

Counters. Trièst algorithms keep counters to compute the estimations of the global and local number of triangles. They always keep one global counter τ for the estimation of the global number of triangles. Only the global counter is needed to estimate the total triangle count. To estimate the local triangle counts, the algorithms keep a set of local counters τ_u for a subset of the nodes $u \in V^{(t)}$. The local counters are created on the fly as needed, and always destroyed as soon as they have a value of 0.

3.1 Trièst-BASE

```
**Algorithm 1** Trièst-BASE Algorithm
**Input:** Insertion-only edge stream $\sum, integer M \geq 6$
S $\leftarrow \emptyset, t \leftarrow 0, \tau \leftarrow 0$
for element (+, (u, v)) in $\sum$ do
    $t \leftarrow t + 1$
    if $SampleEdge((u, v), t)$ then
        $S \leftarrow S(u, v)$
        $UpdateCounters(+, (u, v))$
    end if
end for
function SampleEdge((u, v), t)
    if $t \leq M$ then
        return True
    else if FlipBiasedCoin($\frac{M}{t}$) = heads then
        $(u', v') \leftarrow$ random edge from S
        UpdateCounters$(-, (u', v'))$
        return True
    end if
    return False

end function

function UpdateCounters(•, (u, v))
    $\mathcal{N}^s_{u,v} \leftarrow \mathcal{N}^s_u \bigcap \mathcal{N}^s_v$
    for all $c \in \mathcal{N}^s_{u,v}$ do
        $\tau \leftarrow \tau \bullet 1$
        $\tau_c \leftarrow \tau_c \bullet 1$
        $\tau_u \leftarrow \tau_u \bullet 1$
        $\tau_v \leftarrow \tau_v \bullet 1$
    end for
end function
```

3.2 TRIEST-IMPR

Trièst-impr is a variant of trièst-base with small modifications that result in higher-quality (i.e., lower variance) estimations. The changes are:

1. UpdateCounters is called unconditionally for each element on the stream, before the algorithm decides whether or not to insert the edge into S. W.r.t. the pseudocode in Algo. 1, this change corresponds to moving the call to UpdateCounters before the if block.

2. Trièst-impr never decrements the counters when an edge is removed from S. W.r.t. the pseudocode in Algo. 1, we remove the call to UpdateCounters on line 6.

3. UpdateCounters performs a weighted increase of the counters using $\eta(t) = max\{1, \frac{(t-1)(t-2)}{M(M-1)}\}$ as weight. W.r.t. the pseudocode in Algo. 1, we replace "1" with $\eta(t)$

The algorithm is described as follows.

Algorithm 2 Trièst-Impr Algorithm

```
Input: Insertion-only edge stream $\sum$, $integer M \geq 6$
S $\leftarrow \emptyset, t \leftarrow 0, \tau \leftarrow 0$
for element (+, (u, v)) in $\sum$ do
    $t \leftarrow t + 1$
    UpdateCounters$(+, (u, v))$
    if $SampleEdge((u, v), t)$ then
        $S \leftarrow S(u, v)$
    end if
end for
1:  function SampleEdge((u, v), t)
2:      if $t \leq M$ then
3:          return True
4:      else if FlipBiasedCoin($\frac{M}{t}$) = heads then
5:          $(u', v') \leftarrow$ random edge from S
6:          $S \leftarrow S \backslash \{(u', v')\}$
7:          return True
8:      end if
9:      return False
10:
11: end function
12:
13: function UpdateCounters(•, (u, v))
14:     $\mathcal{N}^s_{u,v} \leftarrow \mathcal{N}^s_u \bigcap \mathcal{N}^s_v$
15:     $\eta(t) = max\{1, \frac{(t-1)(t-2)}{M(M-1)}\}$
16:     for all $c \in \mathcal{N}^s_{u,v}$ do
17:         $\tau \leftarrow \tau + \eta(t)$
18:         $\tau_c \leftarrow \tau_c + \eta(t)$
19:         $\tau_u \leftarrow \tau_u + \eta(t)$
20:         $\tau_v \leftarrow \tau_v + \eta(t)$
21:     end for
22: end function
```

4 How to run the code

To develop our application, we have used Jupyter Notebook. Therefore, it is sufficient to launch the Jupyter Notebook environment from the conda shell

(3.x) and simply run all the code cells in a consecutive order. Please check that the input dataset is in the same file_path level of the .ipynb notebook.

5 Discussions

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

The only difficulty was to retrieve exactly the proper steps of the improved version of the TRIEST algorithm, since for that version is not provided a pseudo-algorithm, but it only explain how to modify the base one. Furthermore, to see obtain proper results it is important to keep a M size sufficiently large, otherwise there is the risks to underestimate the counters.

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

It is possible to parallelize the algorithm, but with the only condition that both the global counters and the set of M edges are shared in memory and not distributed across different executors to maintain consistent values. Indeed the reservoir sampling need to know exactly how many edges are already present and compute the probability to replace an edge with the incoming one and the update counters increase the value of the counters and needs to access the previous counters values.

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

Yes, the main advantage of this family of algorithms is being able to process unbounded graphs as it uses Reservoir Sampling method which has a fixed number of sample from the stream and thus, fully utilizing the available memory. The other approach of keeping elements by a probability p will not work well for unbounded graphs the memory usage grows over time till it hits the limit.

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

The Triest-Base and Triest-Impr algorithms are insertion only algorithms. On the other hand, Triest-FD algorithm can handle fully dynamic streams including edge deletions. In order to support edge deletions, we should modify it to use Random Pairing(RP) instead of reservoir sampling. RP extends reservoir sampling to handle edge deletions by compensating edge deletions with future edge insertions. The algorithm works by maintaining counters to keep track of uncompensated edge deletions.

References

[1] L. De Stefani, A. Epasto, M. Riondato, and E. Upfal, TRIÈST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size, KDD'16.