Data mining assignments 3 : Counting traingles

Benedith Mulongo

October 2019

1 TRIEST : the algorithm

TRIEST is a streaming algorithm that computes an unbiased, low-variance and reliable approximation of the global and local number of triangles in a given graph. The algorithm is presented in the paper : *TRIÈST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size Lorenzo* [1].

Three different algorithms is proposed the first two use a simple version of reservoir sampling that is based on insertion-only and the third is fully-dynamic and support both insertions and deletions in adversarial manner.

This report will on focus on the first two variant namely : TRIESTbase and TRIESTimpr. For further details read the article [1].

1.1 Reservoid sampling

Reservoir sampling is a simple sampling that is nonetheless very effective and is used extensively in streaming algorithms. Especially when we need to estimates the number of coming in a stream but we can not store all the element in the stream. A good starting point in order to fully understand the reservoir sampling algorithm is the article [2].

```python
import numpy as np
def reservoirSampling(k. stream) :
    reservoir = list(np.zeros(k))
    n = 1
    for item in stream :
        if n < k :
            reservoir[n] = item
        elif flipBiasedCoin(n/k) :
            random_k = np.random.randint(k, size=1)[0]
            reservoir[random_k] = item
        n = n + 1
    return reservoir
def flipBiasedCoin(prob_heads=.5) :
    turn = np.random.uniform(0,1)
    return turn < prob_heads
```

Listing 1: SimpleReservoirSampling

The TRIEST rely heavely on the reservoir sampling algorithms in order to estimate the number of triangles with a fixed size of memory.

1.2 What makes a triangle in a graph ?

The next important property of the TRIEST is a rather intuitive property of a triangle in a graph.

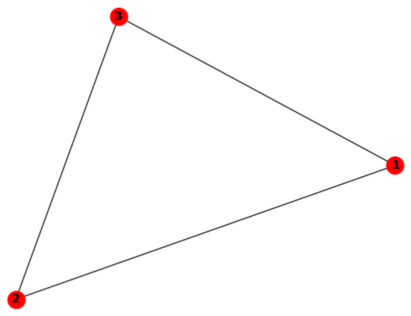


Figure 1: Triangle graph

Figure 1 shows a graph with exhibits a triangle. We can observe the following property :

Two nodes u and v forming an edge $e = (u, v)$ are nodes of a triangle if and only they have nodes in common

Let define a neighbour of node u in a graph G as all the nodes that are incident to u except u itself. Given edge $e = (u, v)$ we have :

$$\mathcal{N}(u)^G = \{p \in V^t : (u, p) \in G\}$$

$$\mathcal{N}(v)^G = \{s \in V^t : (s, v) \in G\}$$

The above property can be then be summarized as : an edge $e = (u, v)$ forms a triangle whenever

$$\mathcal{N}^G_{u,v} \leftarrow \mathcal{N}^G_v \cap \mathcal{N}^G_u \tag{1}$$

and

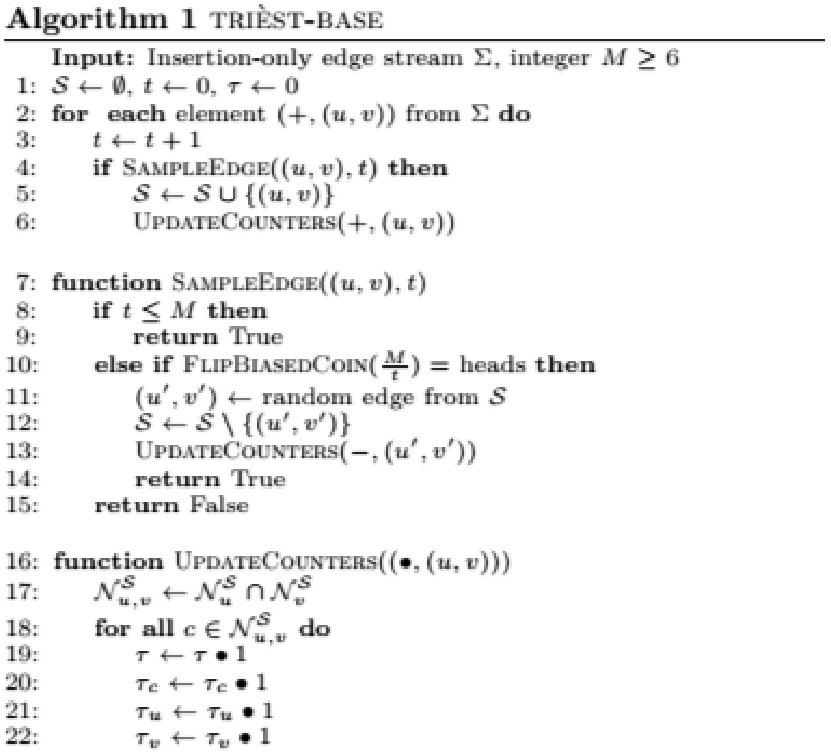
$$\mathcal{N}(u, v)^G \neq \varnothing \tag{2}$$

X

The goal of the algorithm is to count the number of times property in equation (1) and (2) is true. That will represent the total number of global number of triangles in graph G.

1.3 The TRIEST base algorithm

Using the reservoir sampling and the property of triangle in a graph G, we have all the essential part of the TRIEST algorithm shown in figure 2



Algorithm 1 TRIÈST-BASE

```
Input: Insertion-only edge stream $\Sigma$, integer $M \geq 6$
$\mathcal{S} \leftarrow \emptyset$, $t \leftarrow 0$, $\tau \leftarrow 0$
for each element $(+, (u, v))$ from $\Sigma$ do
    $t \leftarrow t + 1$
    if SampleEdge($(u, v), t$) then
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{(u, v)\}$
        UpdateCounters($+, (u, v)$)

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 $\mathcal{S}$
        $\mathcal{S} \leftarrow \mathcal{S} \setminus \{(u', v')\}$
        UpdateCounters($-, (u', v')$)
        return True
    return False

function UpdateCounters($(\bullet, (u, v))$)
    $\mathcal{N}^{\mathcal{S}}_{u,v} \leftarrow \mathcal{N}^{\mathcal{S}}_{u} \cap \mathcal{N}^{\mathcal{S}}_{v}$
    for all $c \in \mathcal{N}^{\mathcal{S}}_{u,v}$ do
        $\tau \leftarrow \tau \bullet 1$
        $\tau_c \leftarrow \tau_c \bullet 1$
        $\tau_u \leftarrow \tau_u \bullet 1$
        $\tau_v \leftarrow \tau_v \bullet 1$
```

Figure 2: Triest Base algorithm

The procedure SAMPLEEDGE computing the reservoir sampling operation for size M. UPDATECOUNTERS counts the number of triangle using the property explained above. The FOR-loop check that the reservoir can add new element and then update the count of triangle as shown in the pseudocode. •

1.4 The TRIESTImpr algorithm

Algorithm 1 TRIESTImpr

```
procedure RUNTRIEST($Streams, M$)                ▷ Main procedure
    $Graph \leftarrow 0$
    $t \leftarrow 0$
    $\tau \leftarrow 0$
    for each $e = (u, v) \in$ Streams do
        if $SAMPLEEDGE\{(u, v), t\}$ then
            $Graph \leftarrow Graph \cup (u, v)$
        $UPDATECOUNTERS+, (u, v)$
procedure SAMPLEEDGE($(u, v), t$)
    if $t \leq M$ then
        return $True$
    else if FLIPBIASEDCOIN($\frac{M}{t}$) $= head$ then
        $(u', v') \leftarrow Graph[randomEdge]$
        $Graph \leftarrow Graph \diagdown (u', v')$
        return $True$
    return $False$
procedure FLIPBIASEDCOIN($prob$)
    $turn \leftarrow random.uniform(0, 1)$
    if $turn < prob$ then
        return $head$
    else
        return $Pile$
procedure UPDATECOUNTERS($+, (u, v)$)
    $\mathcal{N}^G_{u,v} \leftarrow \mathcal{N}^G_v \cap \mathcal{N}^G_u$
    $\eta \leftarrow max\{1, \frac{(t-1)(t-2)}{M(M-1)}\}$
    for all c $\in \mathcal{N}^G_{u,v}$ do
        $\tau \leftarrow \tau + \eta$
        $\tau_c \leftarrow \tau_c + \eta$
        $\tau_u \leftarrow \tau_u + \eta$
        $\tau_v \leftarrow \tau_v + \eta$
```

2 Reflections

2.1 Implementation challenges

The main difficulty too implement the algorithm is to understand deeply the pseudo-code. The data representation is also a difficulty because the algorithm is designed for streaming data. The pseudo-code is not very clear, but when it is finally understand the implementation is pretty simple. I have used the

Python library NetworkX[1] in order to easy handle edges deletions/insertions, neighbourhoods etc.

2.2 Parallelisation of the algorithm

The algorithm can be parallelized if and only if given a graph G, we can divide the graph G in **n** components/sub-graphs such as $G_1, G_2, \ldots, G_n$ and

$$\Delta(G) = \Delta(G_1) + \Delta(G_2) + \ldots + \Delta(G_n)$$

$$G = G_1 \cup G_2 \cup \ldots \cup G_n$$

and for $(i, j) \in [1, \ldots, n]$ and $i \neq j$

$$G_i \cap G_j = \varnothing$$

is true. Where Δ is the number of triangle in the sub-graph/graph G. In order to divide the graph G in **n** such sub-graphs for which the triangles can be counted independently, we need to find **n** edges/vertex that will result in **n** sub-graphs, but that can be possible only if we have access to the Graph G before processing which is not possible when we only have access to edges during each timestamp. In order words, given that we do not have access to the all graph G, it is not possible to parallelize the algorithm and ensure the correctness of the algorithm.

2.3 Algorithm's behavior for unbounded graph streams

The algorithm work well for unbound graph as the algorithm make no assumptions about the size of the graph or the number of edges, however when the size of the graph G tends to be unbound, increase indefinitely and M is fixed the estimation correctness decreases drastically but nonetheless possible. For better estimation M need to vary with the graph size.

2.4 Modification for edges insertions and deletions

TRIEST base and improved support insertions only operations, in order to support deletions we need to make it fully-dynamic by modifying the reservoir sampling method by keeping track of two variables d_i = uncompensated edges deletions involving an edge that is in graph G and $d_o$uncompensated edges deletions involving an edge that is **not** in graph G at the time t. Having those variables we can check when to inset or delete edges from reservoir.

[1]https://networkx.github.io/

3 Data

The data used in this report is taken from `http://konect.uni-koblenz.de/networks/`. We have test the algorithms for 4 different dataset :

- **Les miserables :** This undirected network contains co-occurances of characters in Victor Hugo's novel 'Les Misérables'. A node represents a character and an edge between two nodes shows that these two characters appeared in the same chapter of the the book. The weight of each link indicates how often such a co-appearance occured. Number of triangles = 467
- **Europa roads :** This is the international E-road network, a road network located mostly in Europe. The network is undirected; nodes represent cities and an edge between two nodes denotes that they are connected by an E-road.Number of triangles = 32
- **Train bombing :** This undirected network contains contacts between suspected terrorists involved in the train bombing of Madrid on March 11, 2004 as reconstructed from newspapers. A node represents a terrorist and an edge between two terrorists shows that there was a contact between the two terrorists. The edge weights denote how 'strong' a connection was. This includes friendship and co-participating in training camps or previous attacks. Number of triangles = 527
- **Windsurfers :** This undirected network contains interpersonal contacts between windsurfers in southern California during the fall of 1986. A node represents a windsurfer and an edge between two windsurfers shows that there was a interpersonal contact. Number of triangles = 1096

4 Results

What we can observe is that the estimation found by the TRIESTimproved is much more accurate even for reservoir size half the number of edges. TriestBase needs a reservoir size almost equal to the number of edges ($M \approx t$) in order to find a accurate result.

Let's take the Miserables data as an example, it has 254 edges and 467 triangles

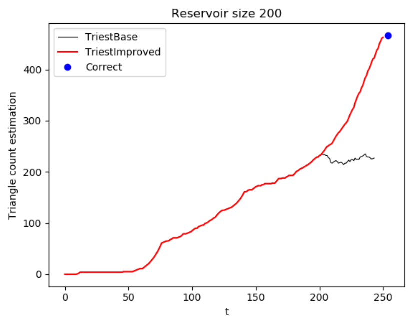


Figure 3: Triangle graph

Figure 3 shows that the TRIESTImproved and TriestBase, behaves similarly for the most time, but the TRIESTImproved increases its accuracy at the end. So the result from TRIESTImproved is much more reliable.

$TriestBase = 227$ $TRIESTImproved = 462.77532663316595 \approx 467$

It is better to note that Triest is a randomized algorithm so the result may changes for each runs.

4.1 Les miserables

Number of triangles = 467

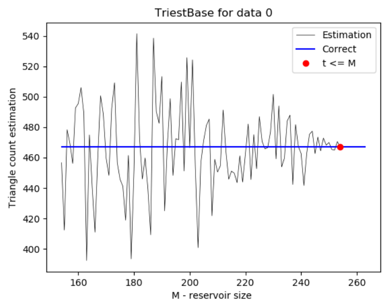


Figure 4: Result for TriestBase

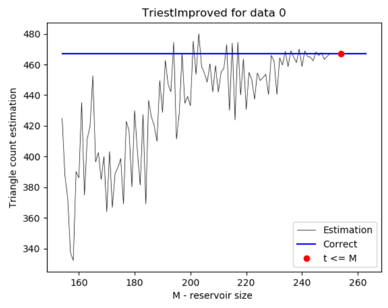


Figure 5: Result for TriestImpr

4.2 Train bombing

Number of triangles = 527

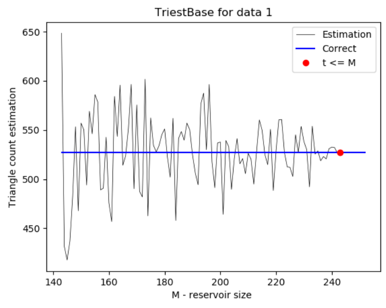


Figure 6: Result for TriestBase

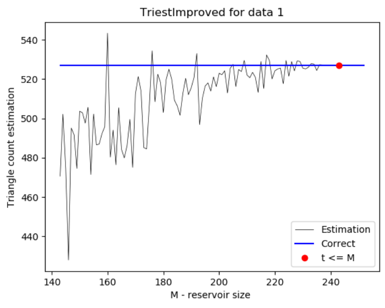


Figure 7: Result for TriestImpr

4.3 Windsurfers

Number of triangles = 1096

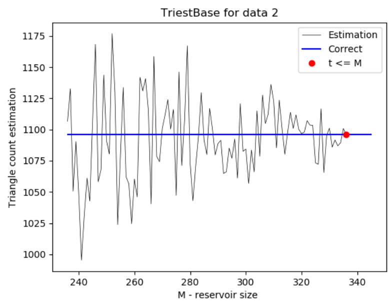


Figure 8: Result for TriestBase

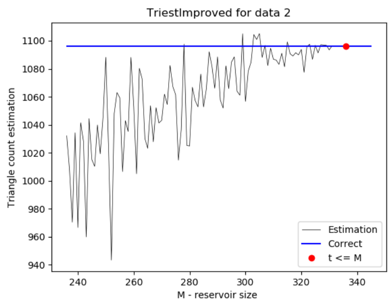


Figure 9: Result for TriestImpr

4.4 Europa roads

Number of triangles = 32

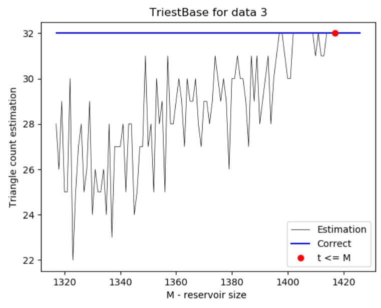


Figure 10: Result for TriestBase

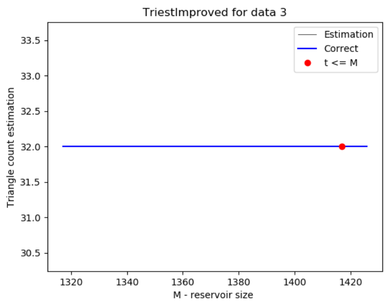


Figure 11: Result for TriestImpr

References

[1] Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. TRIÈST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size ”Ogni lassada xe persa” 1-Proverb from Trieste. 2016.

[2] Miles Osborne, Ashwin Lall, and Benjamin Van Durme. Exponential Reservoir Sampling for streaming language models. *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference*, 2:687–692, 2014.