# HW3 - Mining Data Streamsts

Tianxiao Zhao <tzh@kth.se>
Yantian You <yantian@kth.se>

## 1. Build and run

To run our program, change the variable "*data_path*" to the path of the input documents, and then run the Main class (including main function). The Main class will automatically execute testing function and print out the results.

## 2. Solution

The sampling and streaming graph algorithm we have chosen come from the second paper-*TRIÈST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size.*

### 2.1 Sampling

We develop two functions for data stream sampling, *sampleEdge* and *FlipBiasedCoin.* The *streaming* function has been used to add edges into edge sample **S** depends on return value(Boolean) of these two functions.

- *FlipBisedCoin* function has been used to determine whether a newly arrived edge will be add into edge sample **S** or not. It contains local random value(between 0 to 1) and an input heads probability **M/t**. **M** is a positive integer represents the size of **S**. **t** is the timestamp of input stream and every time when a new edge arrives it will be added by 1. If the random value is less than **M/t,** this function will return True.
- *sampleEdge* function has been used to remove an random edge in **S** once **t** is larger than **M** and *FlipBisedCoin* returns true value. It will return True when **t** is smaller than **M** or when the random edge has been successfully removed. Otherwise, this function will return False.
- Once the *streaming* function received true value from *sampleEdge* function, it will add the newly arrived edge into edge sample **S.** If false has been received, it will do nothing with the sample set **S**.

### 2.2 Streaming graph algorithm

After sampling the data stream, we implement the TRIÈST-IMPR algorithm to estimate the global count and local count of triangles. The pseudocode for TRIÈST-IMPR is presented below:

**Algorithm** TRIÈST-IMPR

```
    Input: Insertion-only edge stream Σ, integer M ≥ 6
1:  S ← ∅, t ← 0, τ ← 0
2:  for each element (+, (u, v)) from Σ do
3:      t ← t + 1
4:      UpdateCounters(+, (u, v), t)
5:      if SampleEdge((u, v), t) then
6:          S ← S ∪ {(u, v)}

7:  function SampleEdge((u, v), t)
8:      if t ≤ M then
9:          return True
10:     else if FlipBiasedCoin(M/t) = heads then
11:         (u0, v0) ← random edge from S
12:         S ← S \ {(u0, v0)}
13:         return True
14:     return False

15:  function UpdateCounters(•, (u, v), t)
16:      N_{u,v}^S ← N_u^S ∩ N_v^S
17:      delta = max{1, (t - 1)(t - 2)/M(M - 1)}
18:      for all c ∈ N_{u,v}^S do
19:          τ ← τ • delta
20:          τ_c ← τ_c • delta
21:          τ_u ← τ_u • delta
22:          τ_v ← τ_v • delta
```

According to the pseudocode above, we create a new method called *UpdateCounters* which updates the global and local counts of triangles during iterations with a weighted increase *delta = max{1, (t - 1)(t - 2)/M(M - 1)}*. Besides, *UpdateCounters* is called unconditionally for each element on the stream rather than only being called after meeting the condition *SampleEdge((u, v), t) == True* in TRIÈST-BASE. To keep track of counts, here we define an integer variable *tau* as global count, and a HashTable<Integer, Integer> *tau_c* as local count, with key set to the hash value of a chosen vertex and value set to corresponding local counts.

## 3. Results

The data set we used to test our program comes from Douban.com (http://konect.uni-koblenz.de/networks/douban), launched on March 6, 2005. It is a Chinese Web 2.0 website providing user review and recommendation services for movies, books, and music. This data set contains 654,188 edges and 154,907 nodes in total and the estimated number of Triangles is 40,612 which has been published on the official website.

When testing the program, we found that different M will result in different global number of triangles. The results of different M shows as below.

Table 1: Global counts of triangles wrt. value of M

| Value of M | Global Number of Triangles |
|---|---|
| 20000 | 52604 |
| 25000 | 45848 |
| 29000 | 44033 |
| 30000 | 43943 |
| 31000 | 44221 |
| 35000 | 45794 |
| 40000 | 50616 |

From this table, we found that when M has been set to 30000, the global number of triangles we had achieved through our model is 43943 which is very close to the real one with rather small error 0.082.

From this result we can make a conclusion that our model could achieve high accuracy on counting triangles through on line data stream when M is 30000. However, the best M value is various from different dataset and should be chosen carefully.

## 4. Bonus Tasks

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

It's really hard to choose an optimal value of M, and we haven't figured out a smart way to do this instead of trial and error. Another challenge for us could be to fully understand those equations and theorems in the provided literature.

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

Yes, local estimation can be conducted in different nodes in parallel and can be gathered to produce global estimation.

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

Yes, this algorithm uses reservoir sampling to sample the input data stream. The reservoir sampling is a kind of sampling method deployed for unbounded graph streams. However, for unbounded streams, it is impossible to count the total number of triangles until all the edges have been received. Some technique like sliding windows should be used to output rolling estimates for given time periods. And for unbounded streams, M should be carefully chosen since it will determine the estimation accuracy and is related to data size.

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

We used TRIÈST-IMPR instead of TRIÈST-FD which does not support edge deletions for streaming graph. The reason why we choose TRIÈST-IMPR lies in that the data  streaming for this lab doesn't require any delete operation so we haven't implement the edge deletion function. To modify the algorithm for future edge delete function, random pairing technique should be used. We need to add two more counters, $d_i$ and $d_o$, to keep track of the number of uncompensated edge deletions, which involves an edge ***e*** that was in **S** at the time the deletion for ***e*** was on the stream.