FINAL REPORT

Andrea Ghiotto 2118418 - Giacomo Masiero 1234600 - David Petrovic 2092073

January 26, 2024

Title: "Evaluation of (global) clustering coefficients obtained with different triangle count algorithms"

Abstract: Clustering coefficient is a core metric for graph analysis. The purpose of this project is to gain a comprehensive understanding of the variation in (global) clustering coefficients resulting from the implementation of different algorithms for approximating the number of triangles in a graph. In this paper has been described the overall scenario, the algorithms used, how they had been implemented and all the experiments done to reach final results.

1 Introduction

The global clustering coefficient of a graph is a measure that assesses the overall level of clustering in the network and it is derived from the number of closed triplets (triangles). Counting the number of triangles of a graph is a popular primitive, used for example in social networks analysis, for detecting web spam and in other scenarios. Since the exact count of the number of triangles of a large network takes a considerable amount of time, and since for many applications we do not have to work with the exact number, in the literature are proposed many algorithms for approximating the number of triangles of a network.

In this project we are interested in understanding how the (global) clustering coefficients are influenced by the use of different algorithms for approximating the number of triangles of the graphs. After some research we have decided to use three algorithms, one of which is the `Temporal locality-aware sampling` that we have seen in class. Then we have followed the steps below:

1. implement the three algorithms;
2. test them with five dataset of different dimension;
3. compare the approximate number of triangles resulting;
4. calculate the global clustering coefficient of each graph with the information retrieved by each algorithm;
5. compare the results.

2 Algorithms

Triangle count algorithms that we have used are presented and described in the following research papers:

- Temporal locality-aware sampling for accurate triangle counting in real graph streams (Dongjin Lee, Kijung Shin, Christos Faloutsos) [1].
- Reservoir-based sampling over large graph streams to estimate triangle counts and node degrees (Lingling Zhang, Hong Jiang, Fang Wang a, Dan Feng, Yanwen Xie) [2].
- Fast Counting of Triangles in Large Real Networks: Algorithms and Laws (Charalampos E. Tsourakakis) [3].

The first one is a research paper that presents accurate triangle counting in graph streams using temporal locality-aware sampling. `Waiting-Room Sampling` (WRS) is a family of single-pass streaming algorithms for estimating the count of global and local triangles in a fully dynamic graph. They exploit the temporal locality by storing the most recent edges, which future edges are more likely to form triangles with, in the waiting room, while they use reservoir sampling and its variant for the remaining edges.

The second research paper introduces `Triangle-Induced Reservoir Sampling` (T-Sample), a method that efficiently sampled edges and allows to approximate the number of triangles and estimates node degrees in large graph streams. T-Sample processes every edge only once and employs a dual sampling mechanism, uniform (used to count triangles) and non-uniform (for node degrees). For authors, T-Sample outperforms state-of-the-art reservoir-based methods in terms of time and memory costs, providing more accurate triangle counts with reduced errors and variances.

The last one introduces `eigenTriangle`, an advanced algorithm designed to accurately approximate the number of triangles in large graphs by using eigenvalue theory and the graph's adjacency matrix. By applying the Lanczos method, it efficiently approximates eigenvalues from the matrix, selecting the most significant ones to better capture the graph's structure. This method proves to be highly accurate and significantly faster than traditional approaches, providing a precise count of triangles in the graph based on the cubic sum of the selected eigenvalues.

3 Implementation

In this section, we will describe how we have implemented the three algorithms presented above. The whole repository of the project, with algorithms implementation, is available at `https://github.com/petrovicdavid/LFN-project.git` [13].

3.1 Temporal locality-aware sampling for accurate triangle counting in real graph streams

For what concern the first algorithm, in the original paper there is a link from which can be downloaded its implementation [9]. We only got it and used it. For running this implementation, the input dataset must be formatted in a certain way. Since not all the chosen graphs respected this structure, we have written a script in *edit_dataset.py* for formatting the datasets in the right way.

3.2 Reservoir-based sampling over large graph streams to estimate triangle counts and node degree

Regarding the Reservoir-based sampling algorithm, in the original paper is presented the pseudo-code to which we had referred to write the code; it is reported below.

This is an algorithm for the sampling that can be modified in order to approximate the number of triangles or the node degrees. Our goal was to compute an approximation of the number of triangles in the graphs so, by following what is written in the original paper, we needed to code only until the first part of the if statement, with the base reservoir (R_{base}), while the second part, the one inside the else statement, didn't help our case.

To compute the approximate number of triangles of a dataset, as illustrated here [2], we've used the following formula:

$$triagles = \sum_{i=3}^{n} \frac{m_i^{T-sample}}{prob_i}$$

where n is the number of edges in the graph, $m_i^{T-sample}$ is the exact number of triangles between the i-th edge and two edges from R_{base}, while $prob_i$ is a probability equal to

$$\text{prob}_i = \min\left(1, \left(\frac{c}{i-1}\right)^2\right)$$

where c is the value of the capacity of R_{base} and i is the number of the i-th iteration of the for statement. When c is higher or equal to i-1, the value of prob_i is equal to 1 and this is like counting

exactly all the triangles that involved the i-th edge and others two from R_{base}. To manage this we have added the first c edges in R_{base}, counting the exact number of triangles among R_{base} edges using this method [12] of NetworkX and then we iterate from the c+1 edge until the last. At each iteration, we insert the i-th edge in R_{base} if p_i^{in} is greater then a random number from 0 (excluded) and 1 (included), then we count the exact number of triangles in which that edge and the ones from R_{base} are involved at time i and finally we update the value of the approximate number of triangles (*triangles*).
We have decided to implement the R_{base} as a NetworkX graph and, when an edge is sampled, we delete an edge at random to insert the new one in the base reservoir. Despite this is not the most efficient way, we made this choice because, when iterating over all the edges, it was more convenient for us to count the triangles composed by i-th edge and by two from R_{base} edges at time i, thanks to the available methods of this library.
For counting the exact number of triangles in which the i-th edge and the ones from R_{base} are involved, we have managed the following cases:

1. if the i-th edge was sampled we directly count the number of triangles in which it is involved;
2. if the i-th edge was not sampled we insert the edge in R_{base}, we count the number of triangles in which it is involved and finally we erase that edge from the base reservoir.

Algorithm T-Sample

Input: $E = \{e_1, e_2, ..., e_n\}$: a graph stream and c: the capacity of R_{base}
Output: S: set of edge samples

$R_{base} \leftarrow \{e_1, ..., e_c\}$; $R_{incre} \leftarrow \{\}$
for $i = c + 1$ to n **do**
  $num_i \leftarrow num_{i-1}$
  $p_i^{in} = \frac{c}{i}$
  Generate r_1 randomly from $(0, 1]$
  if $r_1 < p_i^{in}$ **then**
    Remove an edge from R_{base} randomly
    $R_{base} \leftarrow R_{base} \cup \{e_i\}$
  else
    if e_i can form triangles with the edges in R_{base} **then**
      $num_i \leftarrow num_i + 1$
      Generate r_2 randomly from $(0, 1]$
      if $r_2 < \frac{c}{c+num_i}$ **then**
        $R_{incre} \leftarrow R_{incre} \cup \{e_i\}$
      end if
    end if
  end if
end for
$S = R_{base} \cup R_{incre}$

For the legend with the meaning of all symbols used in the above pseudo-code, please refer to [2].

3.3 Fast Counting of Triangles in Large Real Networks: Algorithms and Laws

As for the previous algorithm, also for this one there is not an implementation already available but, in the original paper, there is only the pseudo-code, that we have presented below.
This algorithm requires as input the adjacency matrix of the graphs and we do this firstly by create a graph and then by using this method [10] exploiting the NetworkX library.
This algorithm works by iteratively search for an eigenvalue of the adjacency matrix: at time k, the Lanczos method returns the k-top eigenvalue, compared with the others as absolute values (the absolute values are used in the selection phase, whereas in the computation of the approximate number of triangles eigenvalues are used as they are). The search phase ends when the last eigenvalue found affects less then a certain threshold (*tol*, that we set at 0.05 as in the original paper). Note that in the pseudo-code, this is not clear because it seems that the search phase go on until the last eigenvalue affects for less then 5%, but this is not possible because the search phase will stop immediately.

The crucial part that we have coded is thus the search of the eigenvalues; we didn't use an implementation of the Lanczos method because we didn't found an appropriate one, but we have found a scripy method named `eigsh` ([11]) that *"find k eigenvalues and eigenvectors of a real symmetric square matrix or of a compex Hermitian martrix A"*. Thus, this method returns in a single call the k eigenvalues of greater magnitude. An Hermitian matrix with elements in the field of real numbers is a symmetric matrix and so, since the adjacency matrix is a real symmetric matrix, we can properly use this method. Furthermore, in the function documentation we can stated that this method is based on ARPACK, which is a software suggested in the original paper for this task. Thus, we can conclude that anyway we have used a method that is referring to the original one. This method accepts as parameters the matrix and the number of eigenvalues to find. We have set to 50 the latter because, despite in the original paper is written that, in the most cases, the number of eigenvalues used is no more then 30, for one of our graphs 47 of them were needed.

After the call of the `eigsh` method, we iterate on them in order to select the ones that respect the condition (the eigenvalues that affects more then 5% of the sum of the cubic of all the selected ones so far).

Finally, the approximation of the number of triangles for a given graph is computed as the sum of the cubic of the selected eigenvalues, divided by 6.

Algorithm The EIGENTRIANGLE Algorithm

Require: Adjacency matrix A (nxn)
Require: Tolerance tol
Output: $\Delta'(G)$ global triangle estimation

$\lambda_1 \leftarrow \text{LanczosMethod}(A, 1)$
$\vec{\Lambda} \leftarrow [\lambda_1]$
$i \leftarrow 2$
repeat
  $\lambda_i \leftarrow \text{LanczosMethod}(A, i)$
  $\vec{\Lambda} \leftarrow [\vec{\Lambda}\ \lambda_i]$
  $i \leftarrow i + 1$
until $0 \leq \frac{|\lambda_i^3|}{\sum_{j=1}^{i} \lambda_j^3} \leq tol$
$\Delta'(G) \leftarrow \frac{1}{6} \sum_{j=1}^{i} \lambda_j^3$
return $\Delta'(G)$

For the legend with the meaning of all symbols used in the above pseudo-code, please refer to [3].

4 Datasets

Since the machines that we have used for the experiment are not so powerful and we didn't know how they would perform in this type of tasks, we have started with five dataset of relative small dimension (in the order of thousands). After some work, we have seen that the machines could manage datasets of larger dimension and, to better testing the three algorithms, we have decided to use graphs with a number of edges in the order of millions.

Since our purpose is to compare results using datasets that are significantly different from each other in terms of dimensions, we have decided to use the following ones:

- a network that is a snapshot of the structure of the Internet at the level of autonomous systems. It is composed of 22963 nodes and 48436 edges [4].
- a lexical network of words from the WordNet dataset. It contains 146005 nodes and 656999 edges [5].
- an undirected network of Flickr images sharing common metadata such as tags, groups and locations. It is composed of 105938 nodes and 2316948 edges [6].
- a catser friend graph containing 204473 nodes and 5448197 edges [7].
- a network of YouTube users and their friendship connections. This graph is composed of 3223589 nodes and 9375374 edges [8].

5 Experiments

In this section, we will described how we proceed in order to run the algorithms.

We have decided to run each algorithm three times for each dataset in order to see if there are some changes in the results. The machine used for the final results has the following specs: CPU Intel i7-1165G7; 16GB RAM; Ubuntu.

For the first dataset we have followed the suggestions in the original paper [1] by setting the parameters regarding the maximum number of sampled edges to 10% of the number of edges in the graph and the relative size of the waiting room to 0.1.

For what concern the second algorithm, in the original paper [2] there aren't precise indications about the value of the capacity of R_{base} to use and since we have not found a percentage or a value that allows good results over all the five datasets, we have run the algorithm several times for each dataset for finding "good" values for each of them. We are aware that it has no sense for an approximation algorithm to run it on different values until the one that better fits is found, but this was the only option we had to try to make this algorithm competitive with the other two presented in this project. The capacities of R_{base} founded and used for each dataset are reported in the result section in the *Capacity of R_base* column of the corresponding table.

Regarding the third algorithm there are no parameter to set, so we just run it three times for each dataset.

After the running of the three algorithms, we have also implemented a python script to compute the clustering coefficient given the number of edges of a graph and the approximate number of triangles that results from the previous tests.

The detailed instructions on how to run this experiments are available in the *README.md* files of each directory of the GitHub repository [13]. Please note that, in order to run the algorithms, you are required to download the datasets [4] [5] [6] [7] [8] and to save the *out* file of each of them as a *.txt* file in an appropriate folder named *dataset*, as they were to heavy to fit in the repository.

6 Results

In this section, we will illustrate the results obtained by running the three algorithms over the datasets previously described. In the tables below are reported only the averages of the approximations of the number of triangles and the variances over the three runs executed for each dataset; the whole results of each algorithm are available in the corresponding directories in the repository [13] (one .txt file for three run of a dataset over an algorithm).

Each file has been saved with the following format: *number_of_edges approximate_number_of_triangles* (or *clustering_coefficient*, depending on what had been run).

6.1 Temporal locality-aware sampling for accurate triangle counting in real graph streams

	Number of edges	Exact number of triangles	Avg. approx. number of triangles	Exact CC	Avg. approx. CC	Std. deviation
Netwotrk of the structure of Internet	48 436	46 873	47 588	4.13e-10	4.19e-10	0.19
Lexical netwokr of words from WordNet	656 999	1 144 910	1 146 406	4.04e-12	4.04e-12	0.02
Flickr images network	2 316 984	107 987 357	161 686 030	8.68e-12	8.67e-12	0.03
Catster friends network	5 448 197	185 462 177	185 272 154	1.15e-12	1.14e-12	0.04e-1
Network of YouTube users	9 375 374	12 226 580	12 172 717	1.48e-14	1.48e-14	0.02

As we can notice considering the obtained results we can state that the first algorithm is very precise. There's only a difference between the exact and the approximate number of triangles running it with Flickr images dataset [6], while the exact and approximate clustering coefficients are almost the same. We can also observe that this algorithm has a very low variance in the results.

6.2 Reservoir-based sampling over large graph streams to estimate triangle counts and node degree

	Capacity of R_base	Number of edges	Exact number of triangles	Avg. approx. number of triangles	Exact CC	Avg. approx. CC	Std. deviation
Netwotrk of the structure of Internet	2 165	48 436	46 873	46 950	4.13e-10	4.13e-10	0.09e-1
Lexical netwokr of words from WordNet	5 648	656 999	1 144 910	1 114 263	4.04e-12	3.93e-12	0.08e-1
Flickr images network	11 740	2 316 984	107 987 357	100 060 176	8.68e-12	8.04e-12	0.05
Catster friends network	13 406	5 448 197	185 462 177	156 991 244	1.15e-12	9.70e-13	0.32
Network of YouTube users	-	9 375 374	12 226 580	-	1.48e-14	-	-

Also regarding this algorithm we can assert that the results, both the approximate number of triangles and the clustering coefficient, are quite confident, especially for the firsts three datasets [4] [5] [6], while there is a visible difference in the approximate number of triangles for the Catster friends dataset [7].

We didn't fill the table for the last dataset [8] since for it the execution time was extremely high.

6.3 Fast Counting of Triangles in Large Real Networks: Algorithms and Laws

	Number of edges	Exact number of triangles	Approx. number of triangles	Exact CC value	Approx. CC value
Netwotrk of the structure of Internet	48 436	46 873	50 785	4.13e-10	4.47e-10
Lexical netwokr of words from WordNet	656 999	1 144 910	181 838	4.04e-12	6.41e-13
Flickr images network	2 316 984	107 987 357	81 975 677	8.68e-12	6.59e-12
Catster friends network	5 448 197	185 462 177	181 299 724	1.15e-12	1.12e-12
Network of YouTube users	9 375 374	12 226 580	9 339 710	1.48e-14	1.13e-14

For the last algorithm we can notice some more discrepancies between the exact number of triangles and the approximated obtained one. We can see that, for the first [4] and the fourth [7] datasets, the difference is minimal, while for the third [6], the fifth [8] and especially the second ones [5] it is very consistent. Is easy to observe how this differences has an impact in the resulting values of the clustering coefficient.

For this last algorithm we didn't report the values of the standard deviation, that is equal to 0 for all dataset, since the resulting values of the approximate number of triangles was the same for every run of the algorithm with the given dataset. In this case in fact, the columns of *Approx. number of triangles* and *Approx. CC value* represents the true obtained values and not an average like in the previous tables.

7 Conclusions

To summarize, we were interested in understanding how the values of the (global) clustering coefficient of several graphs change by using different algorithms for approximating the number of triangles.

We found three algorithms for approximating this sum. Then, we have implemented and tested them with five graphs of significant dimension and, at the end, we have compared the results.

The most difficult part of this project was to understand precisely what was written in the papers, especially for replicating the described algorithms basing only on the pseudo-code.

As it is described in the previous section, the difference between the exact value of the clustering coefficient and the approximated one is not very large in almost all the cases and we have shown that using an approximate algorithm do not cause sensible differences in the final result.

References

[1] *Temporal locality-aware sampling for accurate triangle counting in real graph streams (Dongjin Lee, Kijung Shin, Christos Faloutsos)*; https://link.springer.com/article/10.1007/s00778-020-00624-7

[2] *Reservoir-based sampling over large graph streams to estimate triangle counts and node degrees (Lingling Zhang, Hong Jiang, Fang Wang a, Dan Feng, Yanwen Xie)*; https://www.sciencedirect.com/science/article/pii/S0167739X19331577

[3] *Fast Counting of Triangles in Large Real Networks: Algorithms and Laws (Charalampos E. Tsourakakis)*: https://www.math.cmu.edu/~ctsourak/tsourICDM08.pdf

[4] Network of the structure of the Internet at the level of autonomous systems: http://konect.cc/networks/dimacs10-as-22july06/

[5] Lexical network of words from WordNet dataset: http://konect.cc/networks/wordnet-words/

[6] Flickr images network: http://konect.cc/networks/flickrEdges/

[7] Catster friends network: http://konect.cc/networks/petster-cat-friend/

[8] Network of Youtube users: http://konect.cc/networks/youtube-u-growth/

[9] Temporal locality-aware sempling algorithm implementation: https://github.com/kijungs/waiting_room

[10] to_scipy_sparse_array documentation: https://networkx.org/documentation/stable/reference/generated/networkx.convert_matrix.to_scipy_sparse_array.html

[11] eigsh: https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.eigsh.html

[12] NetworkX method to compute the exact number of triangles: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.cluster.triangles.html

[13] GitHub repository: https://github.com/petrovicdavid/LFN-project.git

Members contributions

We now give details about the contribution of each member of the group, in terms of research, implementation, testing and report writing.

Ghiotto Andrea

ID: 2118418
Email: andrea.ghiotto.3@studenti.unipd.it

- Main contributions: papers founding, understanding of algorithms, general approach.
- Report writings: Project Proposal, Mid-term Report, Final Report.
- Coding: T-sample.py, eigenTriangle.py, computeCC.py, focusing on implementation details and result saving process.
- Testing: second algorithm and clustering coefficient.
- Fraction of work done: 46%

Giacomo Masiero

ID: 1234600
Email: giacomo.masiero.4@studenti.unipd.it

- The only contribution of this member was finding one of the three papers.
- This member's name doesn't appear in the repository since his contribution in the implementation and testing phase was null.
- Fraction of work done: 1%

Petrovic David

ID: 2092073
Email: david.petrovic@studenti.unipd.it

- Main contributions: papers and datasets founding, understanding of algorithms, general approach.
- Report writing: Project Proposal, Mid-term Report, Final Report.
- Coding: T-sample.py, eigenTriangle.py, computeCC.Py, edit_dataset.py, focusing on implementation structure and details.
- Testing: first and third algorithms
- Fraction of work done: 53%