Counting triangles in large graphs

Boris Radovič[a,1], **Aljaž Verlič**[a], **and Samo Metličar**[a]

[a]University of Ljubljana, Faculty of Computer and Information Science, Večna pot 113, SI-1000 Ljubljana, Slovenia

The manuscript was compiled on May 14, 2021

In this paper we present the problem of triangle-counting in large networks, a task that has drawn researchers' attention for the past two decades. We first motivate the reasons for such interest and later on review the most recently developed algorithms that were proposed for this task. Our final goal is to perform a thorough comparison of a number of selected algorithms.

Triangles are probably the most basic non-trivial structures that populate graphs. They consist of three nodes that are connected to each other to form a closed triplet, i.e. a ring of length 3, and counting how many such structures are present in a given graph has been proven to be extremely useful by countless empirical results. For example, triangles may be a useful feature for differentiating between spam and legal websites (1) and more in general, to group websites based on their content (2); they may also be used for community detection purposes (3), or even to estimate content quality in social networks (1). Also, there are many measures that either directly or indirectly evaluate the presence of such structures in graphs - clustering coefficient and transitivity coefficient are just two of these - so in short, counting triangles is a task of primary importance.

It therefore comes as no surprise, that many algorithms have been proposed to tackle this task, task that is in fact extremely complex considering the huge dimension of today's graphs. What's more, is that the graphs we face today are constantly changing: consider for example how social networks constantly evolve as new people join the network and form new relations with each other, or even how does the web constantly change by the additions and removals of websites. As most researchers in their papers claim that the algorithm they propose outperforms any foregoing algorithm, there is the urge to perform a thorough and unbiased comparison between all these algorithms. In this paper we shall therefore perform such a study, comparing the time and space requirements, the simplicity with which they may be coded, the ability to evolve to dynamic graphs, and other relevant metrics, of some selected state-of-the-art algorithms.

Related work

Throughout this section we denote with $G(V, E)$ the graph consisting of $n = |V|$ nodes and $m = |E|$ edges. Also, d_v denotes the degree of the vertex $v \in V$.

The most naive algorithm we can think of to count triangles is exhaustive search: by listing all the (non-necessarily connected) triplets in a graph we may count, whether they are connected between each other or not. Considering that there are $\binom{n}{3}$ such triplets, we derive that the asymptotic time complexity of such approach is $O(n^3)$ - of course, assuming that checking if a set of three nodes forms a triangle takes constant time $O(1)$.

A considerable improvement of this strategy is achieved by enumerating all the length-2 paths that exist in a graph: for each such path, the algorithm verifies if there exists an edge that connects the two endpoints of the path, yielding this way an improvement in term of time complexity, as this decreases to $O(\sum_{v \in V} deg(v)^2)$ (4). It is worth noting that such a strategy causes each triangle to be detected three times, fact that leaves room for substantial improvements. For example, the algorithm might be further optimized by delegating low degree vertices, i.e. by searching the triangle candidates only on the node with a sufficiently low degree - in such a case, the algorithm takes the name *forward algorithm* and may be further improved into the *compact* forward algorithm (5). It might be proven, that the time complexity of both these algorithms is $O(m^{\frac{3}{2}})$, whereas the space they require is linear in the number of edges (5). Another similar approach suggests iterating through all edges, and for each edge check the size of the intersection of the neighborhoods of the two nodes, which can be done, assuming that the set of nodes in the neighborhoods are sorted, in $O(d_u + d_v)$ time - this algorithm takes the name *edge iterator algorithm*.

A different approach is counting the number of cycles of length 3 in a network (6). The proposed method runs in $O(V^\omega)$ time, where $\omega < 2.376$ is the exponent of matrix multiplication. It works especially well on sparse graphs, property that should make the algorithm suitable for real networks.

Another family of algorithms leverages the information that is explicitly stored inside the adjacency matrix A of the graph G. A representative example of this family, proposed in (7), computes the number of triangles using Equation 1:

$$n_T = \frac{1}{6} \sum_{ij} (A^2 \circ A)_{ij} \qquad [1]$$

where $\circ$ denotes element-wise product. Another related approach was proposed in (8), in which authors follow a similar strategy, though they also use the incidence matrix. The pitfall of both these approaches, and in general of any technique that requires the graph to be represented as a matrix, is evident: the size of the (typically sparse) adjacency matrices are bound to be too large to be held in memory even for modest-size networks.

Two algorithms are proposed in (9) that leverage the property expressed in Equation 2:

$$\Delta(G) = \frac{1}{6} \sum_{j} \lambda_j^3 \qquad [2]$$

i.e., the total number of triangles is proportional to the sum of cubes of the eigenvalues of the adjacency matrix. Though

All authors contributed equally to this work.

[1]To whom correspondence should be addressed. E-mail: br8097@student.uni-lj.si.

this formula might be exploited to compute the exact number of triangles, the authors settle for an approximation using the Lanczsos method, approximation that allows to speed up the execution time and reduce the space complexity to $O(m)$. The algorithm yielded very good results under the assumption that the absolute values of the eigenvalues follow a power law, a property that emerges often in real networks.

Nevertheless, the size of modern networks may prevent the networks from even being stored in the main memory. In order to cope with this issue we might resort to the *stream* or *semi-stream processing paradigm*, i.e. we may renounce to the random access that allows us for example to retrieve the neighbors of a node in constant time, and only read data in an arbitrarily ordered sequence (10). Of course, we cannot expect algorithms to return the exact result by scanning only a handful of times a sequence of edges, so in (1) were proposed two semi-streaming *approximation* algorithms that perform $k \leq \log(n)$ passes over the secondary storage. The algorithms approximate the Jaccard coefficients, which we know is defined as $c = \frac{|S_u \cap S_v|}{|S_u \cup S_v|}$ where S_u and S_v are sets of nodes representing the neighborhoods of nodes u and v. The time complexity is linear in the number of edges $O(k \cdot m)$, while the space complexity is of the order $O(n \log(n))$. In (1) is also proposed an improvement of this algorithm that reduces the access to the secondary memory to speed up the execution time by up to 60%. It is worth noting, that both these algorithms not only count the overall number of triangles, but yield the number of triangles incident to each node.

Another approximation algorithm was proposed in (11), in which the number of triangles is estimated through a single pass over the networks' edges by sampling edges to be kept in memory with reservoir sampling. In the study are leveraged some properties of $\hat{\tau}$, which is a random variable that depends on the number of edges encountered in the edges stream and on the time in which the edges contained in the reservoir were sampled. In particular, it turns out that the expected value of this random variable is equal to the number of triangles, i.e. $\hat{\tau} = t(G)$. The time complexity of the algorithm is $O(m + r)$, r being the number of estimators used by the algorithm.

Most recently, researches managed to approximate the overall number of triangles by using an algorithm running in sub-linear time (12), yielding good results by observing as little as 3% of the graph. The algorithm performs a random walk starting from an arbitrarily selected node v. After performing a walk of length R and assigning to each traversed edge a weight proportional to the degree of its two endpoints, it randomly selects l visited edges and checks, whether the selected edge and a randomly selected edge incident to either of the two endpoints form a triangle. This strategy has also the non-negligible advantage, that it does not require the presence of the whole graph, nor does it require any prior knowledge about the number of nodes or vertices in the graph. Therefore, such an algorithm may be used to estimate the triangle density.

Finally, let us just point out that parallelization has also drawn high interest, and some of the algorithms mentioned thus far can be run concurrently (13). In this light, and considering the recent establishment of MapReduce as the de-facto standard for parallel computing, in (14) is proposed an algorithm that divides the graph into multiple overlapping sub-graphs on which the map function is run concurrently. The reduce function is the one that actually counts the triangles.

Project proposal

Data. Given the diversity of modern graphs, we shall evaluate the algorithms on several real networks presenting different characteristics (different degrees of density, clustering, size etc.). We shall also test the algorithms on synthetic graphs, such as those generated by the Erdős–Rényi model and the Barabási–Albert model. When necessary, we shall resort to some generative model.

Methods. We will focus on algorithms that belong into two categories, exact triangle counting and approximate triangle counting. We will code the algorithms from scratch in the Java programming language by leveraging the capabilities of the JGraphT library (15), and compare the algorithms' performances on the graphs introduced above. Performance metrics will include average execution time, rate at which triangles are found and performance with different thread counts (if applicable to the algorithm). Approximate triangle counting algorithms will also be compared based on the accuracy of their predictions.

Expected results. The final goal that motivates us is to perform a thorough and unbiased study of the presented algorithms in order to declare which one, if any, best combines all the relevant metrics and is therefore suitable to be used for medium-sized networks and projects and in general, which algorithm should be recommenced based on the properties of the graph under observation.

If possible, we shall also propose a novel algorithm and include it into the above comparison.

1. Luca Becchetti, Luca, Paolo Boldi, Paolo, Carlos Castillo, Carlos, Aristides Gionis, and Aristides. Efficient semi-streaming algorithms for local triangle counting in massive graphs. 08 2008. .
2. Jean-Pierre Eckmann and Elisha Moses. Curvature of co-links uncovers hidden thematic layers in the world wide web. *Proceedings of the National Academy of Sciences of the United States of America*, 99:5825–9, 04 2002. .
3. Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The anatomy of the facebook social graph. *arXiv preprint*, 1111.4503, 11 2011.
4. Tim Roughgarden. Cs167: Reading in algorithmscounting triangles. 3 2014.
5. Matthieu Latapy. Theory and practice of triangle problems in very large (sparse (power-law)) graphs. 10 2006.
6. R. Yuster N. Alon and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17: 209–223, March 1997.
7. Paul Burkhardt. Graphing trillions of triangles. *Information Visualization*, 16, 09 2016. .
8. Michael Wolf, Jonathan Berry, and Dylan Stark. A task-based linear algebra building blocks approach for scalable graph analytics. pages 1–6, 09 2015. .
9. Charalampos E. Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *2008 Eighth IEEE International Conference on Data Mining*, pages 608–617, 2008. .
10. M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In *External Memory Algorithms*, 1998.
11. A. Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment*, 6:1870–1881, 09 2013. .
12. Suman Bera and C. Seshadhri. How to count triangles, without seeing the whole graph, 06 2020.
13. Chad Voegele, Yi-Shan Lu, Sreepathi Pai, and Keshav Pingali. Parallel triangle counting and k-truss identification using graph-centric methods. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2017. .
14. Ha-Myung Park and Chin-Wan Chung. An efficient mapreduce algorithm for counting triangles in a very large graph. pages 539–548, 10 2013. .
15. Jgrapht library. URL https://jgrapht.org/.