Journal of Multidisciplinary Engineering Science Studies (JMESS)
ISSN: 2458-925X
Vol. 8 Issue 1, January - 2022

Evaluation of Exact Triangle Counting for Huge Sparse Graphs

Fitriyani[1]
Institut Teknologi Bandung, School of Electrical Engineering and Informatics
Bandung, Indonesia
fitriyani@telkomuniversity.ac.id

Benhard Sitohang[2]
Institut Teknologi Bandung, School of Electrical Engineering and Informatics
Bandung, Indonesia
benhard@stei.itb.ac.id

Achmad Imam Kistijantoro[3]
Institut Teknologi Bandung, School of Electrical Engineering and Informatics
Bandung, Indonesia
imam@informatika.ac.id

***Abstract*—The growth of information technology increases the amount of data generated. Most of the resulting data can be represented as a network or graph. Graph analysis is needed to extract the information contained in the data. Triangle counting is one of the fundamental and important metrics in complex networks analysis. Due to the resulting graphs being very large and diverse behaviors, counting the number of triangles in a graph is a challenge. There are many methods or algorithms to count the number of triangles in a graph.**

Most real networks are sparse, and some of them are very sparse. Previous research mentions that two basic triangle counting algorithms, node iterator and edge iterator work well for this type of graph. We evaluated several known exact triangle counting algorithms and used very large sparse graph datasets. Furthermore, we found that node iterator performs the best performance. Although generally, node iterator has worse complexity than other algorithms, it provides good performance for data that have a small maximum degree. Then we optimize the node iterator algorithm by deleting the nodes with d(v) < 2 before counting the triangles. This optimization gives significant speedup for the node iterator algorithm.

Keywords—component; triangle counting, huge sparse graphs, algorithms performance, optimization

I. INTRODUCTION

Graphs and networks are a ubiquitous model representation of real-life problems to extract information inherent in today's data. For example, we can represent how Facebook works by imagining the connection and interaction between friends. Then we can find out what communities are on Facebook, who are influencers, who are spreading news, and so on.

A triangle in an undirected graph is a set of three vertices that connect each other. Finding, counting, and listing triangles in a graph becomes an interesting and important problem. These are because many of the fundamental and important metrics to extract information from complex networks used the triangle concept, such as measuring transitivity [1] and clustering coefficient [2]. Triangles concept has also been used in several real-world applications, such as spam filtering [3] and community detection [4] .

Triangle counting problems have been discussed for years with various approaches [5]–[10]. And the comprehensive reviews have been presented well in previous research [11]–[13].

Schank [11], [14] evaluates triangles exact counting and listing algorithm with various degree distribution. He found that the forward algorithm gives the best performance. Latapy [12] evaluated existing algorithms and proposed a new algorithm concerning space needs.

Hasan [13] divides three approaches of triangles counting algorithms in random access data: exact triangle counting, approximate triangle counting, and distributed and parallel triangle counting. The last approach should also be combined with exact and approximate triangle counting approaches. Although approximation approaches are much faster than exact approaches, exact approaches provide an actual number of triangles.

Real-life networks tend to be very large, sparse, and skewed. The size of graphs is very large and increasing rapidly. Graph today has million even billion nodes and increases every day. On the other hand, counting triangles in a large graph is computationally expensive. So finding an efficient algorithm is an important issue.

We focused on graphs that are very sparse, such as road networks, protein networks, and some of the social networks. The very sparse networks are the networks that have a small maximum degree, small average degree, and low standard deviation. In other words, the nodes of the graph have degrees that do not differ much from the average of degrees.

We have two contributions to this paper:

i. an experimental evaluation of some existing exact triangle counting algorithms for very sparse networks,

ii. the optimization of triangle counting for very sparse networks by deleting nodes that have d(v) < 2.

We expect that certain algorithms will be efficient for some datasets but not for others. Graph size, degree, and other graph properties distribution will be related to the performances of the algorithms. And deleting nodes with d(v)<2 of very sparse networks will significantly reduce the running time of algorithms. We evaluated several known algorithms using various sizes and sparsity of graph datasets.

II. EXACT TRIANGLE COUNTING ALGORITHMS

Consider an undirected, unweighted, and simple graph G = (V,E) with n = |V| vertices or nodes and m = |E| edges. The degrees of node v denote as d(v) and the maximal degree of graph G as dmax(G). The neighborhood of node v is denoted by N(v).

A triangle of graph G is a set of three vertices {u, v, w} that contain edges = {(u,v), (v,w), (w,u)}. The number triangle of node v denoted as T(v), and the total triangle of graph G denote as T(G).

The earliest triangle counting algorithm is based on matrix multiplication of adjacency matrix. Consider graph G is an undirected graph, and A[G] is the adjacency matrix of graph G. Then the total number triangle in graph G, $T(G) = \frac{1}{6}\mathrm{Tr}(A^3)$, where Tr is the sum along diagonals of the matrix. The complexity of this algorithm is O(n3). This algorithm also can be implemented with fast matrix multiplication with running time $O(n^{\gamma})$, with $\gamma = 2{,}373$, γ is fast matrix multiplication exponent[15]

There are various algorithms for exact triangles counting. Most of them rely on matrix multiplication, node counting, or edge counting. We selected several known algorithms to be evaluated, two basic algorithms (node iterator and edge iterator) and three great modified algorithms (AYZ, node iterator core, and forward).

A. *Node Iterator*

Node iterator examines each node that has pair of neighbors connected and counts it. The algorithm iterates all nodes of graph G, say v. For (u, w) ∈ adj (v), if there is an edge that exists between u and w, {u, v, w} forms a triangle, and otherwise not. The total number of triangles in graph G is the sum of all node's triangles divided by 3. Each triangle will be counted three times, so the division is needed [13]. To avoid double-counting, we can add boundary v<u<w. The running time of this algorithm is $O(nd_{max}^2)$.

B. *Edge Iterator*

Edge iterator iterates over each edge to examine if the edge contains a triangle. An edge e = (u, v) will form a triangle {u, v, w} if u and w have a common neighbor w. Therefore we can count the number triangle of edge e = (u, v) by counting the intersections between the neighbors of u and v. The total number triangle of graph G is a cumulative sum of triangles containing each edge divided three. As node iterator, to avoid double-counting, we can add boundary v<u< w. The running time of this algorithm is $O(m.d_{max})$ [13].

C. *AYZ-Counting*

Alon, Yuster, and Zwick [7], called AYZ, have proposed a triangle counting algorithm which is $O(m^{2\gamma/\gamma+1})$running time. AYZ algorithm divides nodes become two categories, 'low degree' and 'high degree'. It defines threshold $\partial = \frac{\gamma-1}{\gamma+1}$, $\gamma = 2{,}373$ where γ is fast matrix multiplication exponent[15]. Low degree nodes are nodes that have degrees less than ∂, and otherwise, are high degree nodes. The number triangle of low degree nodes counted using node iterator algorithms and (fast) matrix multiplication used to count triangle of high degree nodes.

D. *Node Iterator Core*

Node iterator core is the modification of the node iterator algorithm that uses the core concept. This algorithm takes a node with a currently minimal degree and counts its triangles using the node iterator algorithm, then removes the node from graph G [11], [14]. Count and remove all nodes v with $(v) \leq \sqrt{m}$. The sum triangles of removing nodes and remaining graph are the total number triangles of graph G. The k-core of graph G is the maximal connected subgraph in which every node has degrees at least k. The core number c(v) is the maximum k of all cores it belongs to node v. The node iterator core algorithm. The running time of this algorithm is $O(m^{3/2})$.

E. *Forward*

The forward algorithm is an improvement of the edge iterator algorithm. This algorithm was proposed in [11], [14] , and they concluded that the forward algorithm gives the best result.

The forward algorithm starts with ordering nodes from the lowest degree to the highest. The next step creates an empty array A[v] for each node. Then takes the lowest degree node v. For each node $u \in N(v)$ checked if d(v) < d(u), if yes add v to A[u]. The intersection of A[v] and A[u] form triangles. The running time of this algorithm is $O(m^{3/2})$.

TABLE I. ALGORITHMS COMPLEXITY

algorithm	complexity
node iterator	$O(nd_{max}^2)$
edge iterator	$O(m.d_{max})$
AYZ	$O(m^{2\gamma/\gamma+1})$
node iterator core	$O(m^{3/2})$
forward	$O(m^{3/2})$

Table.1 presents the computational complexity of each algorithm. The complexity expects the worst case of running times.

III. EXPERIMENTS AND EVALUATION

The algorithms were evaluated using several real-world networks taken from the network repository [16]. We selected some of the networks that are very sparse. The selected network datasets have various sizes and sparsity.

As mentioned in the previous section Schank [11], [14] have presented the comprehensive experimental study of several known triangle counting algorithms and several types of datasets. They proposed the forward algorithm, which has good performance for most of the datasets evaluated, and mention that two basic algorithm node iterator and edge iterator work well for graphs that the degrees do not differ much from the average degree. Furthermore, our experiment focused on analyzing the performance of several known algorithms for very sparse networks.

We present the several network datasets with some properties as follows: n = number of nodes, m = number of edges, T(G)= number of triangle graph G, dmean = average degree of graph G, Q1, Q2, Q3 = the first, the second, and the third quartile of the degree distribution, dmax = maximum number of node degree, dstd = standard deviation of degree distribution. The selected road networks are listed in Table II, and Facebook networks are listed in Table III.

Road networks are undirected graphs, nodes represent endpoints and intersections, and edges represent roads connecting the intersections or endpoints. Facebook networks are also undirected graphs, nodes represent people, and edges represent the interaction between people.

Road networks tend to be very sparse. The more sparse a graph is, the number of edges is getting closer to the number of nodes.

For example, the road network Italy-osm has 6.686.493 nodes and 7.013.978. The number of nodes is almost equal to the number of edges. And notice that the maximum degree is only about 9-12, the mean degree is very low, about 2-3 degrees per node, and consequences the standard deviation is also very low, about 0,41 to 1,02.

TABLE II. ROAD NETWORKS PROPERTIES

	PA	IT	TX	BE
n	1088092	6686493	1379917	1441295
e	1541898	7013978	1921660	1549970
T(G)	67150	7410	82869	2420
dmean	3	2	3	2
Q1	2	2	2	2
Q2	3	2	3	2
Q3	4	2	3	2
dmax	9	9	12	10
dstd	1,02	0,41	1,01	0,49

PA = road network of Pennsylvania, IT= road network of Italy-osm, TX= road network of Texas, BE = road network of belgium-osm

Facebook networks have different degree distribution and statistical properties compared to road networks. These networks are also sparse but denser than road networks. The average degree of Facebook networks is much higher than road networks, 74-102 degrees per node, and of course, the standard deviation is also higher. And Facebook networks also have a large maximal degree. Table III presents the properties of facebook networks

TABLE III. FACEBOOK NETWORKS PROPERTIES

	TS	IN	BR	DU
n	36364	29732	22900	9885
e	1590651	1305757	852419	506437
T(G)	11178552	9391083	5369593	5142558
dmean	87	88	74	102
Q1	28	30	23	32
Q2	61	67	54	83
Q3	116	123	103	148
dmax	6312	1358	3434	1887
dstd	104,39	80,06	83,02	91,74

TS = Texas, IN = Indiana, BR = Barkeley, DU = Duke

The experiments were running on machine Intel core i7 @2.60GHz x 12, 16GB memory, and 64-bit machine and using Python3 and compiled with Jupyter Notebook.

Network datasets are given in adjacency list representation. The execution times of algorithms were measured in seconds. Table IV and Table V show the result (time execution) of implementations.

TABLE IV. ROAD NETWORKS

	PA	IT	TX	BE
Node Iter	1,581	7,172	1,955	1,479
Edge Iter	2,660	11,956	3,240	2,539
AYZ	11,799	70,859	14,682	14,618
NIC	8,878	48,715	11,556	10,503
Forward	17,417	88,412	21,906	18,677

PA = road network of Pennsylvania, IT= road network of Italy-osm, TX=road network of Texas, BE = road network of belgium-osm

Node iterator outperforms other algorithms and followed by edge iterator. For more detail, see Fig. 2. Even though node iterator has running time $O(nd_{max}^2)$, the time execution result tends to be low. This is obviously due to road networks having a low maximal degree and standard deviation. The smaller the maximum degree, the smaller the running time. It can be predicted that for nodes that have a large maximum degree, the running time will also be greater. Even for a large maximum degree, the running time will be close to O(n3).

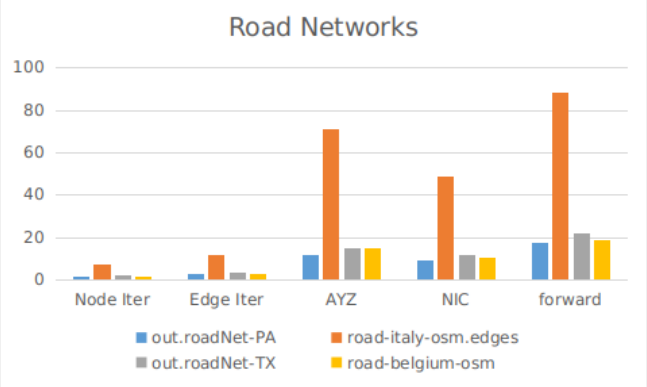

Fig. 1. *Road Networks*

On the other side, the algorithms complexities of AYZ counting, node iterator core, and forward do not rely on the measure of the maximum of degree but the size of the edge. Then we will see the performance algorithms for Facebook networks. Look at Table V and Fig. 3.

TABLE V. FACEBOOK NETWORKS

	TS	IN	BR	DU
Node Iter	554,849	257,026	175,812	132,224
Edge Iter	64,030	40,801	26,419	17,839
AYZ	109,112	52,731	20,143	5,291
NIC	70,865	45,453	29,538	18,969
Forward	14,088	11,320	7,211	4,495

TS = Texas, IN = Indiana, BR = Barkeley, DU = Duke

In contrast to road networks, Facebook networks are not as sparse as road networks but denser. Facebook networks have a much larger maximal degree and standard deviation than road networks.

Table V shows that the forward algorithm outperforms other algorithms. The second position is obtained by AYZ and edge iterator alternately.

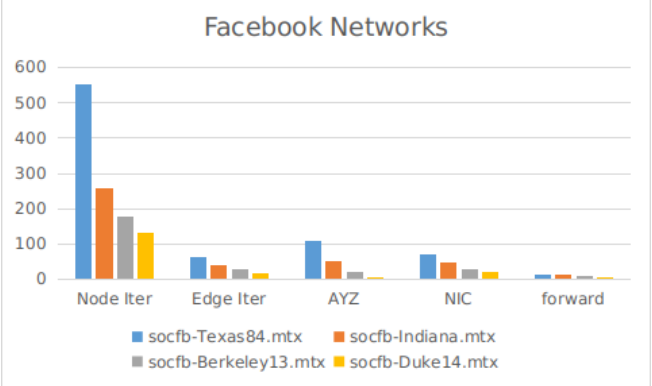

Fig. 2. *Facebook Networks*

Note that the size of Facebook networks is smaller than road networks. However, as previously discussed, node iterator will give poor performance if the maximum degree is enormous. For more detail, see Fig. 2.

Optimization: deletion of nodes that have d(v)<2

The following optimization method is deleting or removing all nodes with d(v) < 2. Deleting all nodes with d(v) < 2 repeatedly until no node has degrees less than two. After node deletion, counting triangles of graph G.

If d(v) < 2, node v must not have a triangle. Deleting or removing node v will not reduce the number of triangles. In fact, this will reduce graph size and complexity. See Fig. 1.

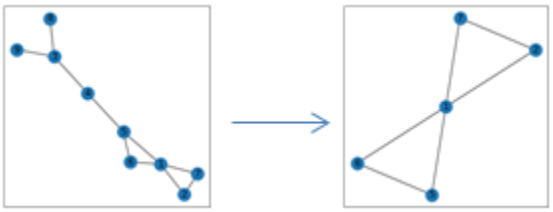

Fig. 3. *(a): Graph G, (b): Graph G'*

Graph G has two triangles, namely (1, 5, 6) and (1, 2, 7), if we delete all node which has d(v) < 2 repeatedly until no node has d(v) < 2 (Fig. 2(b).) the number and the lists of triangles remain the same.

We evaluated these optimization methods for all triangle counting algorithms above, but we got good speedup for the node iterator algorithm, especially for the road networks dataset. See Table VI.

TABLE VI. FACEBOOK NETWORKS

data set	without del (d(v) < 2)	del d(v) < 2
roadNet-PA	1,581	1,379
road-italy	7,172	6,811
roadNet-TX	1,955	1,676
road-belgium	1,479	1,429
socfb-Texas84	554,849	571,382
socfb-Indiana	257,026	247,907
socfb-Berkeley13	175,812	172,956
socfb-Duke14	132,224	137,309

A very sparse graph will have the number of edges close to nodes. So most of the nodes will have degrees less than two. Deleting all nodes with d(v) < 2 will reduce the size and complexity of the graph.

IV. CONCLUSION

There is no efficient algorithm for all type datasets. Old algorithms not always have the worst performance. And the latest algorithm is not necessarily the best.

Running time or computational complexity is a measure of algorithm efficiency, but the execution time may be different in real practice. Node iterator algorithm probably has the worst complexity, but no problem when the networks are very sparse and low maximal degree. Even node iterator performs well for that condition. The distribution and statistical degrees are strongly related to the time execution.

Deleting or removing nodes that have a degree less than two will reduce the size and complexity of the graph, especially for the very sparse graph.

REFERENCES

[1] F. Harary and H. J. Kommel, “Matrix measures for transitivity and balance,” J. Math. Sociol., vol. 6, no. 2, pp. 199–210, 1979, doi: 10.1080/0022250X.1979.9989889.

[2] D. J. Watts and S. H. Strogatz, “Watts_Strogatz_Collective_dynamics_small_world_networks.1998,” Nature, vol. 393, no. June, pp. 440–442, 1998.

[3] M. Rahman and M. Al Hasan, “Sampling triples from restricted networks using MCMC strategy,” CIKM 2014 - Proc. 2014 ACM Int. Conf. Inf. Knowl. Manag., pp. 1519–1528, 2014, doi: 10.1145/2661829.2662075.

[4] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, “Uncovering the overlapping community structure of complex networks in nature and society,” Nature, vol. 435, no. 7043, pp. 814–818, 2005, doi: 10.1038/nature03607.

[5] A. Itai and M. Rodeh, “Finding a minimum circuit in a graph,” Proc. Annu. ACM Symp. Theory Comput., vol. 02-04-May-, no. June, pp. 1–10, 1977, doi: 10.1145/800105.803390.

[6] N. Chiba and T. Nishizeki, “Arboricity and Subgraph Listing Algorithms.,” SIAM J. Comput., vol. 14, no. 1, pp. 210–223, 1985, doi: 10.1137/0214017.

[7] N. Alon, R. Yusfer, and U. Zwick, “Finding and counting given length cycles,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 855 LNCS, pp. 354–364, 1994, doi: 10.1007/bfb0049422.

[8] S. Chu and J. Cheng, “Triangle listing in massive networks,” ACM Trans. Knowl. Discov. Data, vol. 6, no. 4, 2012, doi: 10.1145/2382577.2382581.

[9] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis, “Efficient semi-streaming algorithms for local triangle counting in massive graphs,” Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min., pp. 16–24, 2008, doi: 10.1145/1401890.1401898.

[10] D. Coppersmith and S. Winograd, “Matrix Multiplication via Arithmetic Progressions Department of Mathematical Sciences Y [J ’ Jz /’/ r,” Symb. Comput., no. 9, pp. 251–280, 1990.

[11] T. Schank and D. Wagner, “Finding, counting and listing all triangles in large graphs, an experimental study,” Lect. Notes Comput. Sci., vol. 3503, pp. 606–609, 2005, doi: 10.1007/11427186_54.

[12] M. Latapy, “Main-memory triangle computations for very large (sparse (power-law)) graphs,” Theor. Comput. Sci., vol. 407, no. 1–3, pp. 458–473, 2008, doi: 10.1016/j.tcs.2008.07.017.

[13] M. Al Hasan and V. S. Dave, “Triangle counting in large networks: a review,” Wiley Interdiscip. Rev. Data Min. Knowl. Discov., vol. 8, no. 2, 2018, doi: 10.1002/widm.1226.

[14] T. Schank, “Algorithmic aspects of triangle-based network analysis,” Phd Comput. Sci. Univ. Karlsruhe, 2007, [Online]. Available: http://digbib.ubka.uni-karlsruhe.de/volltexte/documents/4541.

[15] F. Le Gall, “Powers of tensors and fast matrix multiplication,” Proc. Int. Symp. Symb. Algebr. Comput. ISSAC, pp. 296–303, 2014, doi: 10.1145/2608628.2608664.

[16] Ryan A, Rossi and Nesreen K. Ahmed, The Network Data Repository with Interactive Graph Analytics and Visualization, https://networkrepository.com/, 2015