TRIANGLE COUNTING ALGORITHMS

Georgilas Stelios
Dept. of Informatics
Thessaloniki, Greece
sgeorgilas@csd.auth.gr

Charatzoglou Michalis
Dept. of Informatics
Thessaloniki, Greece
mcharatz@csd.auth.gr

ABSTRACT

This project implements various triangle counting algorithms, both exact and approximate, as described in existing literature. It focuses on assessing their scalability characteristics when applied to diverse real-world datasets. The exact algorithms we implement are the brute force algorithm, the node iterator algorithm, and the compact forward algorithm. For approximate triangle counting, we employ the DOULION algorithm. Furthermore, we delve into the application of the streaming algorithm TRIEST to address triangle counting in dynamic graphs. Our experimental analysis evaluates the performance of each algorithm on diverse datasets, examining their efficiency in terms of time and space usage, providing insights on the scalability of triangle counting algorithms for processing large graphs derived from real-world datasets.

Keywords: Triangle counting algorithms, exact algorithms, approximate algorithms, scalability, brute force algorithm, node iterator algorithm, compact forward algorithm, DOULION algorithm, TRIEST algorithm.

1 INTRODUCTION

Counting triangles in graphs is a critical issue in graph theory, providing valuable insights into the structural properties of graphs. It has significant implications across various domains, such as social network analysis (primarily), biological network analysis, recommendation systems, community detection etc. Figure 1 presents the "Triangle Theorem" which states the necessary conditions for the existence of a triangle between three nodes in a graph. To form a triangle, three edges must be present: one connecting node A to node B, another connecting node B to node C, and a third one connecting node C back to node A. These edges are bidirectional, indicating that the nodes are directly linked to each other, creating a closed loop.

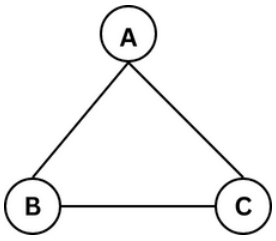

Figure 1: The "Triangle Theorem"

Triangles in a graph may signify clustering or the tendency of nodes to create closely connected groups, providing insights into the graph's overall connectivity and its community organization.

One of the main challenges in triangle counting in large graphs is scalability. Especially for the exact triangle counting algorithms, the task of counting triangles becomes problematic with large graphs, as the computational resources required increase dramatically. The scalability challenge is additionally compounded by the two following issues:

- The sparsity of many real-world large graphs, where there may be a high number of nodes but relatively few edges.
- The dynamic nature of many real-world graphs, which evolve over time, altering their structure and the potential triangles within them.

These challenges make it difficult to apply brute-force and other traditional methods for triangle counting, especially for graphs characteristic of social networks or the internet, which are not only massive but also sparse and constantly changing.

To address the broad scalability issue, there have been developed efficient approximate, randomized, sampling-based and streaming algorithms (Steorts, 2015). These methods aim to provide scalable solutions for triangle counting that can handle the complexities of large, sparse, and dynamic graphs. Additionally, graph sparsifiers (Spielman & Srivastava, 2011) and random walk-based techniques (Riondato & Kornaropoulos, 2016) have been developed to address the sparsity challenge, as well as, sliding windows (Leskovec et al., 2005) and online

algorithms, are among the ones for addressing graph dynamic evolution complexities.

In this project, we contribute to this growing field by benchmarking several exact, approximate and streaming triangle counting algorithms. Utilizing Python, we explore the scalability of the brute force algorithm, the node iterator algorithm, the compact forward algorithm for exact counting, and the Doulion algorithm for approximate counting, along with the TRIEST algorithm for dynamic graph scenarios. Our comprehensive evaluation focuses on the performance of these algorithms across various real-world datasets, assessing their efficiency in terms of time and space complexity. This study aims to offer valuable insights on the scalability of triangle counting algorithms for processing large, sparse and dynamic graphs derived from real-world datasets.

2 DATA

We apply the algorithms to four SNAP datasets (Leskovec & Krevl, 2014). More precisely, we choose the following datasets:

- **G1**. *Social circles: Facebook* (McAuley & J. Leskovec, 2012). This dataset consists of 'circles' (or 'friends lists') from Facebook and includes node features (profiles), circles and ego networks.
- **G2**. *Condense Matter collaboration network* (J. Leskovec et al., 2007). This dataset is an undirected collaboration network, where nodes represent authors and links represent co-authorship of papers submitted to Condense Matter category.
- **G3**. *Amazon product co-purchasing network and ground-truth communities* (Yang and J. Leskovec, 2012). This dataset is crawled from the Amazon website of the "Customers Who Bought This Item Also Bought" feature. If a product *i* is frequently co-purchased with product *j*, the graph contains an undirected edge from *i* to *j*.
- **G4**. *California road network* (J. Leskovec et al., 2009). This dataset represents a road network of California, where nodes represent intersections and undirected edges represent the roads that connect these intersections.

The structural characteristics of the graphs are summarized in Table 1.

Graph	Name	Nodes	Edges	Triangles	Density
G1	Facebook	88234	4039	1612010	0.01
G2	Condensed Matter Collaboration Network	93497	23133	173361	0.0003
G3	Amazon Product Co-purchasing Network	925872	334863	667129	0.000016
G4	California Road Network	2766607	1965206	120676	0.0000014

Table 1: Graphs characteristics.

We categorize the first two datasets as small-scale, the third as medium-scale, and the fourth as large-scale based on their respective sizes. Upon examination, it is apparent that all of the graphs display a low density, a characteristic which aligns with the nature of the datasets. For the graph G1, representing a social network, the sparse connections reflect the realistic scenario where not every individual is connected with one another. Similarly, for graph G2, which maps research collaborations, the data indicates that not all researchers work together, accounting for the lower density of connections. The graph G3 depicts product co-purchases, and the observed sparsity is consistent with consumer behavior where certain products are frequently bought in conjunction with others, but not universally. Lastly, as graph G4 illustrates a road network, the low density is indicative of the actual transportation layout, where direct routes between all locations are not feasible, and travel often requires transiting through various intermediate points.

Finally, as a preprocessing step, we remove all self-loops that may exist in the graphs.

3 EXACT ALGORITHMS

The three basic algorithms used are Brute-Force, Node-Iterator and Compact-Forward. Each of the three is an exact algorithm, and therefore our results in this case are accurate to all extent, thus, our concern is focused on the time complexity of each algorithm.

The **Brute-Force** algorithm for triangle counting finds all triplets, and if all edges appear, then it considers a detected triangle. In more detail, starting with a given node, the algorithm iterates over its neighbors, considering each pair of neighbors to determine if they form a triangle with the starting node. For every pair of neighboring nodes, it checks for an edge that directly connects them. If such an edge exists, the algorithm interprets this as the completion

of a triangle and consequently increments a counter that tracks the number of triangles identified. This process is systematically repeated for every node within the graph. Upon completion of the iteration over all nodes, the final step is to divide the triangle count by six. This adjustment is necessary to correct for the inherent overcounting that occurs due to each triangle being counted multiple times from the perspective of each of its vertices. The time complexity of this algorithm is cubic O(V^3), where V is the number of nodes.

The next algorithm, **Node-Iterator**, works by iterating through all nodes in the graph, while checking if they form a triangle with their neighbours. In more detail, for each pair, the algorithm checks if there is an existing edge between them. The presence of such an edge signifies that a triangle has been formed, since it connects the pair back to the original node. To ensure accuracy in the count of triangles, the total number found is divided by three. This step is crucial as it corrects for the fact that each triangle will be counted three times, once at each of its vertices. The time complexity of this algorithm is O(V$\cdot d^2$), where V is the number of nodes and d tis the maximum degree of a vertex.

The last exact algorithm used is **Compact-Forward**. It is an extension of Forward algorithm, that reduces its space complexity. In more detail, it enhances efficiency by strategically prioritizing nodes based on their degrees, which is the number of connections a node has. For any given node, the process involves specifically considering neighbours that have higher degrees. By comparing the degrees of these neighbours, the algorithm can identify triangles more effectively. This method is crafted to take advantage of the graph's inherent degree structure, thereby streamlining the process of triangle detection. The time complexity of this algorithm is O($V^2 \cdot$ logV), where V is the number of nodes and d tis the maximum degree of a vertex.

Graph	Node Iterator	Compact Forward
G1	26.4 sec.	10.2 sec.
G2	2.3 sec.	2.4 sec.
G3	13.5 sec.	20.3 sec.
G4	8.4 sec.	39.6 sec.

Table 2: The time performance of the exact algorithms.

Table 2 summarizes the performance results of the three above-described triangle counting exact algorithms throughout the four datasets.

Brute-Force is the slowest of all algorithms due to its cubic complexity and it is definitely not a convenient algorithm for counting triangles not even on smaller datasets. Thus, we do not have recorded its performance times due to its complexity.

The Facebook network (G1) has a relatively high density compared to the other graphs, with 88234 nodes and 4039 triangles. The higher density might contribute to the more efficient performance of the Compact Forward algorithm at 10.2 seconds, as it can better leverage the network's degree structure due to the higher likelihood of nodes having common neighbors.

The Condensed Matter Collaboration Network (G2) with 93497 nodes and a significantly lower density than G1, still maintains a substantial number of triangles. The Node Iterator's time (2.3 seconds) was quite close to that of Compact Forward (2.4 seconds), indicating that the Node Iterator is capable of handling sparser graphs with large numbers of nodes effectively, possibly due to its simpler and more direct method of counting triangles.

The Amazon Product Co-purchasing Network (G3) has a vast number of nodes (925872) and edges, but a very low density. The Compact Forward algorithm's performance (20.3 seconds) is less efficient compared to Node Iterator (13.5 seconds), which may be due to the low probability of high-degree neighbors leading to triangles in such a sparse network, thus diminishing the advantages of the Compact Forward algorithm's prioritization strategy.

Lastly, the California Road Network (G4), is the largest in terms of nodes (2766607) and has a very low density, with a relatively small number of triangles. The Node Iterator algorithm completed its task in 8.4 seconds, far outperforming the Compact Forward algorithm at 39.6 seconds. This could be because the Node Iterator does not rely on the degree of the nodes, which becomes an advantage in a road network where the degree may not provide a useful heuristic for triangle formation due to the sparsity and structure of the graph.

In summary, the Node Iterator algorithm shows a consistent performance across graphs of varying sizes and densities, while the Compact Forward algorithm seems to benefit from higher density graphs where its degree-based optimizations can be fully utilized.

4 DOULION

The Doulion algorithm (Tsourakakis et al., 2009) is an approximate triangle counting algorithm in graphs that are

too large for exact algorithms to handle efficiently. It works with graphs that fit entirely in memory as well as those that do not. The algorithm randomly decides to keep each edge in the graph with a certain probability *p* or remove it with a probability of *1-p*. Kept edges are then assigned a new weight of *1/p*, effectively thinning out the graph. This process is known as graph sparsification. Once the graph is sparser, any exact triangle counting algorithm, such as Node Iterator or Compact Forward, can be used to count the triangles in the sparse graph, denoted as G'. In this sparser graph, each triangle is considered to represent $1/p^3$ triangles of the original graph. The number of triangles reported by Doulion is the same as the actual number of triangles in the full graph. The variance is given by the equation:

$$Var(X) = \frac{\Delta(p^3 - p^6) + 2k(p^5 - p^6)}{p^6}$$

where Δ is the total number of triangles of G and k is the number of pairs of triangles that are not edge disjoint. Using Chebysev's inequality it is proved that:

$$Pr(|X - \Delta| \geq \varepsilon\Delta \leq \frac{(p^3 - p^6)}{p^6\varepsilon^2\Delta} + 2k\frac{(p^5 - p^6)}{p^6\varepsilon^2\Delta^2}$$

As we can see, the probability that the algorithm's output approaches the actual number Δ increases as Δ gets larger and as *p* gets closer to 1.

In this project, we apply the Doulion algorithm both with Node Iterator and Compact Forward algorithms. We run the algorithms ten times in order to calculate and present the mean values of accuracy and related error. We select five different values of p: 0.1, 0.3, 0.5, 0.7, and 0.9 to examine the performance of the algorithm in different levels of sparsification.

Figure 2 presents the speedup results in relation to the accuracy of Doulion algorithm combined with the Node Iterator for all the four datasets for different values of *p*.

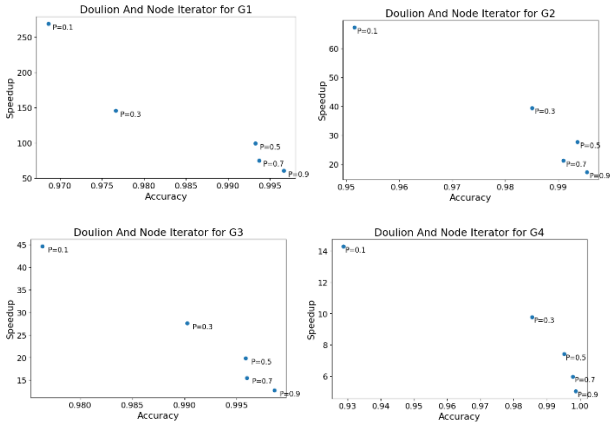

Figure 2: Accuracy results for Doulion & Node Iterator for all datasets.

The G1, with its higher density, shows significant speed improvements with maintained accuracy when using Doulion, particularly at higher probabilities. The G2, despite being less dense, still achieves good speedup without much loss in accuracy, although it's more sensitive to the choice of probability. For the G3, which is large but sparse, the speedup is moderate, and accuracy slightly drops with lower probabilities. The extensive and sparse G4 shows the least improvement in speedup, highlighting the challenges the Doulion algorithm faces with vast, sparse graphs. Overall, while the Doulion algorithm enhances speed, particularly in denser networks, its efficiency gain in sparse networks is more limited, and higher probabilities *p* tend to yield better accuracy.

Figure 3 presents the speedup results in relation to the related error of Doulion algorithm combined with the Node Iterator for all the four datasets for different values of *p*.

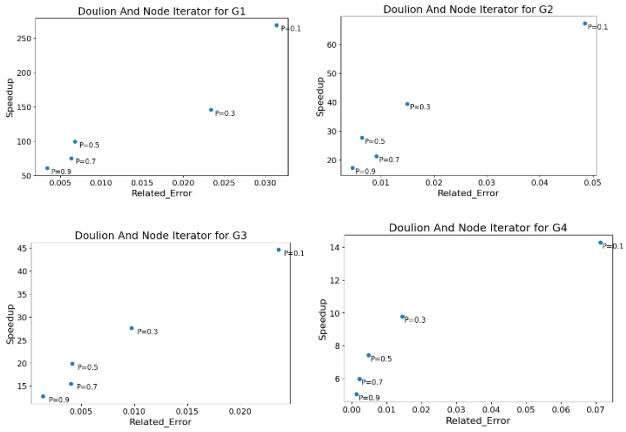

Figure 3: Rel. error results for Doulion & Node Iterator for all datasets.

The G1, with its high density, achieves the greatest speedup, especially at the lowest probability (p=0.1), but this comes with a larger error margin. The speedup is considerable across all probabilities, suggesting density facilitates faster computation but with a cost to accuracy. In contrast, G2 and G3, despite being sparser, exhibit high speedups with more controlled errors across the probabilities. This reflects that while these networks benefit from the algorithm, the sparser the network, the less pronounced the speedup, especially as the probability decreases. The G4 shows a different trend; with the lowest density, the speedup is less dramatic, and the error is higher at p=0.1, indicating that in vast and sparse networks, the algorithm speeds up the process but at a significant cost to the count's accuracy. In summary, the algorithm performs best in terms of speedup on denser networks but at the

expense of accuracy, and this trade-off becomes more critical in sparser networks.

Moving to the Compact Forward triangle counting algorithm, Figure 4 presents the speedup results in relation to the accuracy of Doulion algorithm combined with the Compact Forward for all the four datasets for different values of *p*.

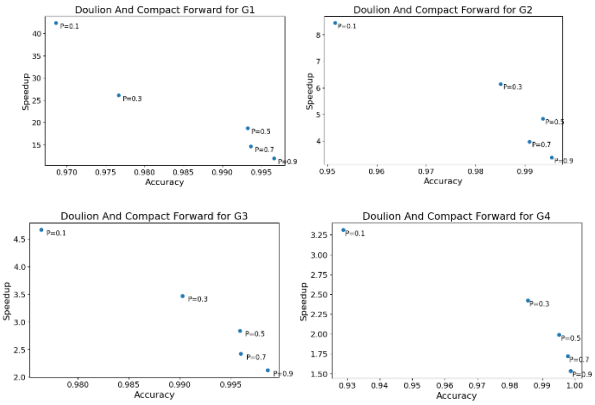

Figure 4: Accuracy results for Doulion & Compact Forward for all the datasets.

In G1 there is a high speedup across all probabilities, with a slight accuracy decrease at lower probabilities. For the sparser G2 the algorithm also achieves good speedup with high accuracy, although the speedup is not as high as in G1. The G3, which is large and sparse, shows moderate speedup and a drop in accuracy as the probability decreases. The G4, the largest and sparsest, shows the least speedup, particularly at higher probabilities, but maintains high accuracy. Overall, while speedup varies with network density, accuracy is consistently high except for a slight decrease in sparser graphs at lower probabilities.

Figure 5 presents the speedup results in relation to the related error of Doulion algorithm combined with the Compact Forward for all the four datasets for different values of *p*.

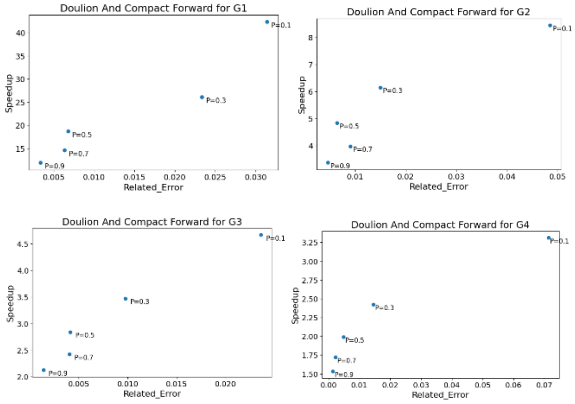

Figure 5: Rel. error results for Doulion & Compact Forward for all the datasets.

For G1 the related error is relatively low across all probabilities, with the lowest error corresponding to the highest probability (p=0.9). As the probability decreases to p=0.1, the error slightly increases, suggesting that more aggressive sparsification leads to less accurate triangle counts. In G2 the error remains low for higher probabilities but starts to increase noticeably as the probability decreases to p=0.1. This indicates that while the algorithm can still perform well at higher probabilities, its accuracy declines when fewer edges are retained. For G3 the trend is similar to G2, with the lowest error observed at the highest probability, and an increase in error as the probability lowers. Finally, G4 shows a related error that increases more significantly as the probability decreases, with the highest error at p=0.1. This demonstrates that in extremely large and sparse graphs, the accuracy of triangle counting is more sensitive to the level of sparsification.

5 STREAMING APPROACH

The importance of streaming algorithms for triangle counting lies in their ability to process data in real time and with limited memory resources, offering a solution that is both efficient and scalable. Streaming algorithms work by processing data in a single pass, with limited memory available for storing the data. The TRIEST algorithm is a popular example of a streaming algorithm for triangle counting, which maintains a reservoir of a fixed size that is used to sample edges from the graph. By sampling edges uniformly at random from the reservoir, the algorithm can estimate the number of triangles in the graph. The TRIEST algorithm has been shown to be effective for large-scale graphs with limited memory resources (Rossi et al., 2014).

For the purposes of this project, we implement the TRIEST algorithm and test it on the four above-described real-world graphs with different memory reservoir sizes (M). In more detail, the algorithm processes edges from an input file sequentially. Upon reading a new edge, designated as (u,v), it updates both global and local counters according to the number of common neighbors shared by the vertices u and v. For every common neighbor, referred to as w, these counters are updated with a probability that is determined by the ratio of M to t. Additionally, each edge (u,v) is probabilistically added to the graph based on the same ratio of M to t. This process is iterated for various values of M (from 10% to 100%) to observe how the algorithm performs under different sampling intensities.

As also describe in the introduction chapter, and similarly to the exact and approximate triangle counting algorithms, the accuracy of the TRIEST algorithm for triangle counting depends on several factors, including the size of the memory reservoir, the sparsity of the graph, and the degree distribution of the nodes in the graph (Tsourakakis 2018).

Figure 6 presents the percentage of number of total edges in relation to the accuracy of the TRIEST algorithm for all the four datasets for different values of M.

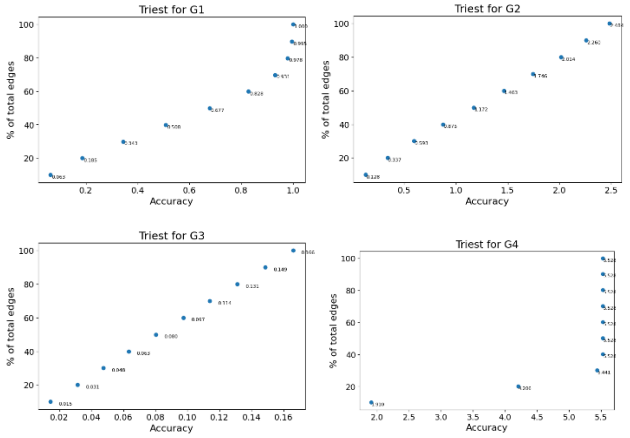

Figure 6: Accuracy results for TRIEST for all the datasets and different M values.

For G1, the Facebook network, which has a relatively high density and a substantial number of triangles given its edges, the TRIEST algorithm shows high accuracy across all percentages of processed edges. This suggests that TRIEST is well-suited to denser networks with a higher potential for triangle formation. In G2, the Condensed Matter Collaboration Network, which has fewer triangles relative to its size and a lower density compared to G1, the accuracy of TRIEST varies more widely. This might be due to the sparser nature of the network, where the probability of triangle formation is less, making the accurate prediction of triangles more challenging as the algorithm processes more edges. For G3, the Amazon Product Co-purchasing Network, which is larger in terms of nodes and edges but has a very low density, the accuracy of the TRIEST algorithm is lower, especially in the initial stages of edge processing. The sparse nature of this network likely contributes to the difficulty in maintaining high accuracy, as there are fewer triangles relative to the graph's size. Lastly, G4, the California Road Network, despite being the largest graph with a significant number of edges, has the lowest density and a smaller number of triangles. The TRIEST algorithm's performance here shows the greatest variation and instances of overcounting. The graph's vast size and extreme sparsity may pose a challenge to the TRIEST algorithm, resulting in lower accuracy, particularly when a smaller percentage of the graph's edges have been processed. Overall, these observations suggest that the TRIEST algorithm's ability to count triangles accurately is influenced by the density and size of the network, with denser networks like G1 resulting in higher accuracy and sparser, larger networks like G3 and G4 showing more variability and challenges in achieving accurate triangle counts.

We also evaluated the TRIEST algorithm on a graph that was computationally not possible to be analyzed with the exact and approximate triangle counting algorithms previously mentioned. This dataset, referred to as G5, represents the SNAP Wikipedia Talk network (Leskovec et al., 2010). Within this dataset, every registered Wikipedia user possesses a talk page which can be modified by themselves or other users for the purpose of dialogue and coordination concerning updates to Wikipedia articles. This network comprises 2,394,385 nodes, 5,021,410 edges, and 9,203,519 triangles. Figure 7 showcases how the accuracy of the TRIEST algorithm correlates with the percentage of the total number of edges processed in the graph tested with varying values of the parameter M.

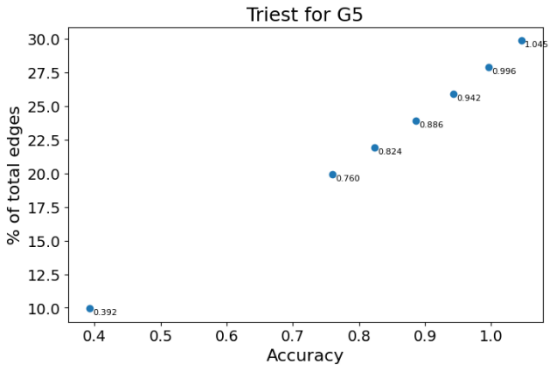

Figure 7: Accuracy results for TRIEST for G5 and different M values

For G5, it appears that as the percentage of processed edges increases, so does the accuracy of the algorithm. The highest accuracy is observed when the algorithm processes the largest portion of edges, which is around 30% of the total edges in the graph. This point corresponds to an accuracy that is very close to 1, indicating that TRIEST is highly accurate when a significant fraction of the edges is considered. The lowest accuracy is seen when the algorithm processes approximately 10% of the edges, with the accuracy at this point being around 0.4. This suggests that when TRIEST operates on a smaller sample of the edges, its estimates are less accurate. This performance trend is consistent with the nature of sampling-based algorithms, where a larger sample typically yields more

accurate results. Considering the substantial number of triangles in G5, the TRIEST algorithm's increasing accuracy with more edge processing is expected. Larger graphs with a significant number of triangles may require a greater percentage of edges to be sampled to attain high accuracy due to the complex interconnections between nodes that form triangles. Overall, the TRIEST algorithm shows improved performance as more of the graph's edges are included in the sampling process. This indicates that for graphs similar to G5, with millions of nodes and triangles, achieving high accuracy in triangle counting would necessitate processing a higher percentage of edges.

Summarizing the benefits of the TRIEST algorithm, it offers a scalable solution for triangle counting in large-scale graphs, which is its primary advantage. It is particularly beneficial in situations where the graph size exceeds the capacity of available memory, and other algorithms fail to be practical. This makes TRIEST a valuable tool for analyzing vast networks that cannot be fully loaded into memory for processing. Another advantage is its feasibility for large graphs. In cases where exact triangle counting algorithms are impractical due to the graph's size, TRIEST steps in as a reliable method to estimate the number of triangles. It allows for the analysis of network connectivity and clustering within graphs that are otherwise too large to handle with traditional methods.

However, there are considerations to bear in mind when using TRIEST. Its performance can be quite sensitive to the choice of M, the parameter that governs the sample size of edges. While a larger M can lead to more accurate triangle counting, finding the right balance is crucial as it directly impacts the algorithm's performance. Another point is the inherent sampling variability. Because TRIEST is based on random sampling, the triangle counts are estimates, and the results may vary with each run. This variability means that while the algorithm can provide a quick estimate, it may not be consistent in its accuracy across multiple iterations. Lastly, TRIEST's accuracy is affected by the edge density of the graph. In networks with low edge density, the algorithm may produce less accurate estimates. This is because sparser graphs have fewer triangles and random sampling might miss these sparse connections, leading to a potential underestimation of the triangle count.

6 CONCLUSIONS

In concluding the project on triangle counting algorithms, it's clear that each algorithm has its strengths relative to the complexity and size of the dataset it addresses. The Node Iterator and Compact Forward algorithms have demonstrated consistent performance across various graph sizes and densities, with Compact Forward excelling in denser networks due to its degree-based optimizations.

The Doulion algorithm, a sparsification approach, shows promising results in speed improvement, especially in denser networks like G1 (Facebook). It provides a practical solution when exact counting is not feasible due to graph size, although its performance is sensitive to the sparsification level set by the probability p.

Finally, the TRIEST algorithm, designed for streaming large-scale graphs, proves to be an efficient tool, achieving high accuracy in denser networks while facing challenges in sparser, larger graphs like G4 (California Road Network). Its utility in handling graphs that cannot be fully loaded into memory highlights its significance in real-world applications where data volume and velocity are considerable. The project's findings contribute valuable insights into the scalability and applicability of these algorithms in different network scenarios.

7 CODE AND DATA AVAILABILITY

The code, the data, and the evaluation results used in the proposed solution are available on GitHub at:

https://github.com/SGeorgilas/Exact-and-approximate-triangle-counting-algorithms

8 REFERENCES

Steorts, R. C. (2015). Entity resolution with empirically motivated priors.

Spielman, D. A., & Srivastava, N. (2008, May). Graph sparsification by effective resistances. In Proceedings of the fortieth annual ACM symposium on Theory of computing (pp. 563-568).

Riondato, M., & Kornaropoulos, E. M. (2014, February). Fast approximation of betweenness centrality through sampling. In Proceedings of the 7th ACM international conference on Web search and data mining (pp. 413-422).

Leskovec, J., Kleinberg, J., & Faloutsos, C. (2005, August). Graphs over time: densification laws, shrinking diameters and possible explanations. In Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining (pp. 177-187).

Leskovec, J., & Krevl, A. (2014). SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

J. McAuley and J. Leskovec (2012). Learning to Discover Social Circles in Ego Networks. NIPS.

J. Leskovec, J. Kleinberg and C. Faloutsos. Graph Evolution: Densification and Shrinking Diameters. ACM Transactions on Knowledge Discovery from Data (ACM TKDD), 1(1), 2007.

Yang and J. Leskovec. Defining and Evaluating Network Communities based on Ground-truth. ICDM, 2012.

J. Leskovec, K. Lang, A. Dasgupta, M. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. Internet Mathematics 6(1) 29--123, 2009.

Tsourakakis, C. E., Kang, U., Miller, G. L., & Faloutsos, C. (2009, June). Doulion: counting triangles in massive graphs with a coin. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 837-846).

Rossi, R. A., Ahmed, N. K., & Neville, J. (2014). Triest: Counting local and global triangles in fully-dynamic streams with fixed memory size. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 1356-1365).

Tsourakakis, C. E., Kapralov, M., & Karypis, G. (2018). Sublinear algorithms for (Δ+ 1)-coloring and triangle counting on sparse graphs. Journal of the ACM (JACM), 65(6), 1-37.

J. Leskovec, D. Huttenlocher, J. Kleinberg. Signed Networks in Social Media. CHI 2010.