Fast and Scalable Triangle Counting in Graph Streams: The Hybrid Approach

Paramvir Singh, Venkatesh Srinivasan, and Alex Thomo

University of Victoria, Victoria BC, Canada
{paramvirsingh,srinivas,thomo}@uvic.ca

Abstract. Triangle counting is a major graph problem with several applications in social network analysis, anomaly detection, etc. One of the most popular triangle computational models considered is Edge Streaming in which the edges arrive in the form of a graph stream. We categorize the existing literature into two categories: Fixed Memory (FM) approach, and Fixed Probability (FP) approach. As the size of the graphs grows, several challenges arise such as memory space limitations, and prohibitively long running time. Therefore, both FM and FP categories exhibit some limitations. FP algorithms fail to scale for massive graphs. We identified a limitation of FM category *i.e*. FM algorithms have higher computational time than their FP variants.
In this work, we present a new category called the Hybrid approach that overcomes the limitations of both FM and FP approaches. We present two new algorithms that belong to the hybrid category: Neighbourhood Hybrid Multisampling (NHMS) and Triest/ThinkD Hybrid Sampling (THS) for estimating the number of global triangles in graphs. These algorithms are highly scalable and have better running time than FM and FP variants. We experimentally show that both NHMS and THS outperform state-of-the-art algorithms in space-efficient environments.

Keywords: Triangle counting · Graph streams · Edge sampling.

1 Introduction

Counting triangles forms a basis for many network analysis, such as social network analysis, anomaly detection, recommendation system, etc. The triangle count is critical for frequently used triangle connectivity, transitivity coefficient, and clustering coefficient, in the analysis. This task is especially challenging when the network is massive with millions of nodes and edges. Several methods had been proposed that are classified into two categories: exact counting and approximate counting. The exact counting, for triangles, is done through enumeration/listing which touches the triangles one by one [7]. The approximate counting is done by sampling the graph and using probabilistic formulae to estimate the total number of triangles in the whole graph based on the number of triangles found in the sample.

Edge streaming is a model where a series of edges arrives in order, one at a time. There are many works published on triangle estimation using this model which we classified into two categories. The first category includes algorithms that have a *Fixed Memory (FM)* Budget. These algorithms sample the edges within the fixed memory

budget and require the user to input the available amount of memory. Once the available memory is full with edges, the next sampled edge randomly replaces an edge in the memory. This is how the sub-graph is maintained within a fixed memory budget.

The second category includes the algorithms that require the user to specify an edge sampling probability p that is fixed for the entire stream, we call them *Fixed Probability (FP)* approach. These algorithms maintain a memory reservoir that doesn't have any size limit. Therefore, if the graph stream arriving is massive, the algorithm often runs out of memory. But, FP algorithms have some benefits over FM algorithms. We analyzed and compared the time complexities of both categories, and show that the FP algorithms are faster than FM algorithms, later in Section 3.

However, both categories have their own limitations detailed in section 3. The sample size always grows in FP algorithms due to which they often run out of memory. On the other hand, FM algorithms have a higher running time than FP algorithms.

The idea of using fixed memory and fixed probability together remains unexplored. We explore this idea and the key intention is to overcome the limitations of FM and FP approaches. We form a new category called the Hybrid category. As the name suggests, the Hybrid category utilizes the power of FM algorithms, that make it more scalable, and the power of FP algorithms, that make it the fastest among all available algorithms.

1.1 Contributions

Our contributions are as follows:

- We prove that FP algorithms are faster than FM algorithms. We provide the running time analysis to prove this claim in Section 3.
- We propose an algorithm called *Neighborhood Hybrid Multisampling (NHMS)*. Not only is our algorithm highly scalable compared to its FP variant *NMS*, but also is significantly faster. The experimental results validate our claims when we compare our algorithm with others on several real-life datasets.
- We propose an algorithm called *Trièst - ThinkD Hybrid Sampling (THS)* that is extremely efficient. We prove that THS is the fastest algorithm available for Triangle Counting in Graph Streams. Our experimental results show that THS is at least 5 times faster than its FM variant.
- We conduct extensive experimentation and prove that our algorithm could execute graphs with billions of edges with in 16 GB RAM and is thus scalable to large graph datasets, whereas many other algorithms fail to execute on the same machine.

2 Prior Work

The extensive literature is available on the approximation of triangle counting in graph streams. Since in this research we propose that the literature presented until now falls in either the *Fixed Memory (FM)* or *Fixed Probability (FP)* category, we will classify the previous work into these categories in this section.

Pavan et al. [6] presented a Neighbourhood Sampling algorithm that needs to execute multiple copies called estimators to approximate triangles. Neighborhood Sampling is an FM algorithm.

Jha et al. [4] applied Birthday Paradox to get the estimate of triangles in graph stream. Similar to Neighborhood Sampling, Birthday Paradox is also FM algorithm.

Stefani et al.[10] presented a suite of algorithms called $Triest$. $Triest$ uses reservoir sampling to sample multiple edges in a fixed memory reservoir. Shin et al. [8] proposed two different algorithms named $ThinkD$ (Think before you discard). The first algorithm is $ThinkD_{Fast}$, which falls in an FP category. The other algorithm is $ThinkD_{Acc}$ which handles a dynamic stream with edges insertion and deletion. As we are considering insertion-only streams, $ThinkD_{Acc}$ is no different from $Triest$. Hence, both $Triest$ and $ThinkD_{Acc}$ belong to the FM category. For brevity, we call this algorithm TS.

Kavassery et al. [5] proposed two algorithms in the paper, Edge-Vertex Multisampling (EVMS), and Neighbourhood Multisampling (NMS). EVMS is an extension to an algorithm by Buriol *et al*. [3] It is categorized as FP algorithm. Kavassery *et al*. [5] presented another algorithm named NMS by modifying the Neighbourhood Sampling [6] using the multisampling approach, similar to EVMS. NMS is also an FP algorithm, it instead uses sampling of edges twice.

None of these works have explored the hybrid approach. We present a new hybrid approach that is both scalable and faster than FM and algorithms. In addition, we present two new hybrid category algorithms ($NHMS$ and THS, more details in section 4), that perform better than all the works mentioned. $NHMS$ is hybrid variant of NMS, and THS is hybrid variant of TS.

3 Fixed Probability vs Fixed Memory Algorithms

3.1 Fixed Probability Algorithms

FP algorithms work as follows: Given a fixed probability p, the algorithm samples the edge from the stream with the probability p and add it to a reservoir, after which the triangle count step is executed for each of the edge in the stream. If a triangle is found, a variable Y is incremented that stores the triangle count in the sample. This continues until the edges in the stream arrive. Whenever the triangle estimates are required, Y is scaled up by some scaling factor and is returned as the final estimate.

Stefani et al.[10] discussed the drawbacks of the algorithms employing FP approach. The limitations of FP algorithms are as follows.

- The input parameter p to be fixed requires in-depth analysis to get the desired approximation quality.
- The reservoir size $|R|$ always grows as there is no limit or restriction to store the number of sampled edges.
- If p is chosen to be large, the algorithm may run out of memory.
- If p is chosen to be small, the algorithms will provide us suboptimal estimates.

3.2 Fixed Memory Algorithms

FM algorithms work as follows: Given a fixed memory budget K, the algorithm samples an edge with probability K/t where t is the time of arrival of an edge, after which the triangle count step is executed for each edge in the stream. Unlike FP algorithms, FM

algorithms scale up the count of triangles found for each edge by some scaling factor η and then add it to Y which is the triangle estimation. This continues until the edges in the stream arrive. Whenever the triangle estimates are required, Y is returned directly as the final estimate. We analyzed and found that there are also some drawbacks for FM algorithms as listed below.

- The selection of input parameter, memory budget K, is not clear (even when there is plenty of memory available), and this is similar to choosing input parameter p for FP algorithms. Namely, If K is small, the triangle estimates we get have low accuracy, and if K is large, the algorithm takes longer to run.
- We further observed that FP algorithms are significantly faster than FM algorithms.

3.3 Analysis

FM algorithms first directly fill the reservoir with the edges streamed. Once the reservoir is full, the edge is sampled by probability K/t and is replaced with a random edge in the reservoir. Hence, the reservoir remains almost full always.

In FP algorithms, there is no such reservoir with fixed size. The fixed probability p specified by the user plays an important role here to store the sampled edges in the memory. Let's say there are $|E|$ edges in graph stream Σ, the number of edges sampled by the algorithm will be $p|E|$.

Time Complexity Comparison Stefani et al. [10] provides a time bound for each edge. For each edge $e = (u,v)$ to compute triangles, TS algorithm requires $O(d(u)+d(v))$ steps where $d(u)$ is the degree of vertex u.

Kavassery et al. [5] does not provide the theoretical time bound proofs for NMS algorithm. We analyzed the NMS algorithm to measure the time complexity that is identified as $(p+q)\sum_{u,v\in R}(d(u)+d(v)+c)$[9].

While comparing the time complexity of NMS and TS, it is clear that they both depend upon the sum of degree of the vertices of an edge arriving in the graph stream i.e. $O(d(u)+d(v))$. We know that the FM algorithms store more number of edges in the reservoir all the time, whereas, the FP algorithms gradually increases the stored edges in the memory. Therefore, the computation of the shared neighbourhood will be more in FM algorithms. This explains the higher computational times in case of FM algorithms.

4 The Hybrid Approach and Algorithms

The key idea behind the hybrid approach is to utilize the benefits of both FM and FP approaches and overcome their limitations. The hybrid approach algorithms have the combination of the fixed memory budget K and the fixed sampling probability p. We present two new algorithms in this category.

Before moving further directly to the new algorithms, let us consider why the hybrid approach is better than FP and FM algorithms. As we have already discussed the limitations of FP and FM approaches in the previous chapter, the questions below would answer how hybrid algorithms overcome them.

Will Hybrid algorithms ever run out of memory like FP algorithms? The hybrid approach limits the size of an edge reservoir by K, similar to FM algorithms, whereas in FP algorithms, there is no such limit and the edge reservoir size always grows. Hence, we overpower FP algorithm's biggest limitation.

Are Hybrid algorithms faster than both FM and FP algorithms? We have already detailed how FP algorithms are faster than FM algorithms in section 3.3. Hybrid algorithms follow the same trend as FP algorithms until the first K edges in the memory are stored. After the first K edges in FP algorithms, the number of edges keep on growing in the memory and will have more number of neighbours stored, whereas in the hybrid approach, the limit on memory reservoir is set, due to which the traversal time becomes almost constant and does not grow much. Hence, hybrid algorithms are faster than both FP and FM algorithms.

How should we select the value of the input p and K in hybrid algorithms together? Considering that the size of K in FM algorithms is chosen to be 1% of the graph stream Σ, it is equivalent to assigning p to be 0.01 for FP algorithms. Similarly, $K = 10\%$ of Σ is equivalent to $p = 0.1$.

Hybrid algorithms expect the input parameters similarly. Considering a user wants to store 10% of the edges in memory, the value of K should be 10% of the total number of edges and p should be 0.1. In case some other values are provided, hybrid algorithms will have two possibilities as detailed below:

- If k is small or p is large, the algorithm will not go beyond the k number of the edges in the memory, similar to FM algorithms.
- If k is large or p is small, the algorithm has enough memory to finish its work.

These characteristics of input parameters for Hybrid algorithms prove to be better than FM and FP approaches, as the incorrect input values for both FM or FP algorithms results in sub-optimal estimations. Considering the advantages of the hybrid approach over FM and FP approaches, We developed the two new algorithms that belong to the hybrid category, and are detailed in the subsequent sections.

4.1 Neighborhood Hybrid Multisampling (NHMS)

Neighborhood Hybrid Multisampling (NHMS) is the hybrid variant of NMS algorithm discussed in Section 2. The intuition behind this algorithm is similar to NMS algorithm, the only difference is the way edges are sampled using a hybrid approach. To understand this better, let's consider an example where we have two triangles, $t_1 = \{a,b,c\}$ and $t_2 = \{a,b,d\}$ that share an edge ab. Assume the order of the edges arriving in the stream is bc, ab, ca, ad, bd. NHMS will sample t_1 if the first edge bc is sampled in L_1, and the edge ab is a neighbour of a sampled edge from L_1 and is sampled in L_2. When the edge ca arrives, the algorithm checks if there's an edge from L_1 $(i.e.\, bc)$ and L_2 $(i.e.\, ab)$ respectively, that forms a triangle. Similarly, t_2 will be counted on arrival of bd, if ad is sampled in L_1, and we already have ab sampled earlier in L_2, therefore, $\langle ab, ad, bd\rangle$ forms a triangle.

The algorithm requires probabilities p and q just like NMS, along with a memory budget K. We maintain two edge reservoirs L_1 and L_2. In our hybrid approach, we limit the size of both the reservoirs to $K/2$.The pseudo-code can be found in Algorithm 1.

Algorithm 1 NHMS

Input: A graph edge stream Σ, memory budget K, probabilities p and q.

$L_1 \leftarrow \emptyset, L_2 \leftarrow \emptyset, Y \leftarrow \emptyset$
for each $e_i = (u,v) \in \Sigma$ **do**
  if coin(p) = "head" **then**
    if $|L_1| < K/2$ **then**
      Add e_i to L_1
    else
      Replace a random edge in L_1 with e_i
  if $e_i \in N(L_1)$ **then**
    if coin(q) = "head" **then**
      if $|L_2| < K/2$ **then**
        Add e_i to L_2
      else
        Replace a random edge in L_2 with e_i
  for every (e_j, e_k) where $e_j \in L_1$, $e_k \in L_2$ such that $time(e_j) < time(e_k)$ and (e_i, e_j, e_k) form a triangle **do**
    Add the triangle (e_i, e_j, e_k) to Y
Return $|Y|/pq$

Theorem 1 (Time Complexity for NHMS). *Let p and q be the fixed probabilities to sample the edges. The time complexity to process an edge $e = (u,v)$ arriving in the stream by Algorithm 1 is $(p+q) \cdot O(d(u)+d(v))$, where d(u) is the degree of vertex u in the reservoir.*

Proof. Suppose that R is the number of edges processed by Algorithm 1, p is the probability to sample L_1 edges, q is the probability to sample L_2 edges, K is the memory budget, and $d(u)$ is the degree of the vertex u in the reservoir.

The running time of NHMS algorithm is dependent on the degree of the vertices of the edges sampled in L_1 and L_2. The triangle count is calculated by traversing the edges from both L_1 and L_2 set, which are stored in the memory. When an edge (u,v) arrives, the common neighbours are searched for each vertex of that edge in L_1 and L_2 set which takes $pd(u)+qd(v)$ and $pd(v)+qd(u)$ steps. This can be further reduced to $(p+q) \cdot O(d(u)+d(v))$.

Theorem 2 (Space Complexity for NHMS). *Let K be the number of edges to be stored on the memory. The space complexity of Algorithm 1 is $O(K)$.*

Proof. Let K be the number of edges to be stored in the memory. The algorithm maintains two edge reservoirs L_1 and L_2 with the maximum size limit of $K/2$. Therefore, the total memory consumed by algorithm will be $O(K)$.

4.2 Trièst / ThinkD Hybrid Sampling (THS)

Trièst/ThinkD Hybrid Sampling (THS) is the hybrid variant of the TS algorithm discussed in Section 2. The algorithm requires a memory budget K just like TS, along with

fixed probability p. We maintain an edge reservoir S which would have a maximum size limit K, the triangle counter variable Y.

Algorithm 2 THS

Input: A graph stream Σ, sampling probability p
1: $S \leftarrow \emptyset, Y \leftarrow 0$
2: **for each** pair $e = (u,v)$ in Σ **do**
3: Update(u,v)
4: Insert(u,v)
5: Estimate(Y)
6: **Function** UPDATE(u,v)
7: **for each** $w \in N(u) \cap N(v)$ **do**
8: $Y \leftarrow Y + 1$
9: **Function** INSERT(u,v)
10: **if** coin(p) = "head" **then**
11: **if** $|S| < K$ **then**
12: $S \leftarrow S \cup \{(u,v)\}$
13: **else**
14: Replace a random edge in S with (u,v)
15: **Function** ESTIMATE(Y)
16: **return** Y/p^2

The intuition behind this algorithm is somewhat similar to NHMS algorithm, the only difference is that we just have one reservoir in this case. Consider an example with triangles $t_1 = \{a,b,c\}$ and $t_2 = \{a,b,d\}$, and the edge stream arrive in an order ab, bc, ca, ad, bd. TS will count t_1 if the edges ab and bc will be stored in S considering it's not full, and ca has neighbouring edges in S $\langle ab, bc \rangle$, that forms a triangle. Now let's consider $S = \{ab, bc, ca\}$ and the reservoir is full, when the new edge arrives, it has to replace an existing edge from S randomly. Therefore, the triangle t_2 will be counted on arrival of bd, if both ad and ab still exists in S. The pseudo-code can be found at Algorithm 2.

Theorem 3 (Time Complexity for THS). *Let p be the fixed probability to sample the edges. The time complexity to process an edge $e = (u,v)$ arriving in the stream by Algorithm 2 is*

$$p \cdot O(d(u) + d(v))$$

where d(u) is the degree of vertex u in the reservoir.

Proof. Suppose R is the number of edges processed by p is the probability to sample S edges, K is memory budget, and $d(u)$ is the degree of vertex u in the reservoir.

The running time of this algorithm is dependent on the degree of the vertices of the edges sampled in memory budget K. The triangle count is calculated by traversing the common neighbours of the vertices of an edge. When an edge (u,v) arrives,

the common neighbours are searched for each vertex of that edge in S set which takes $p \cdot O(d(u)+d(v))$ steps.

If $|R| > K$, the algorithm for the intersection of common neighbours requires $O(d(u)+d(v))$ time, where the degrees are w.r.t. the graph formed by a stream so far.

Theorem 4 (Space Complexity for THS). *Let K be the number of edges to be stored on the memory. The space complexity of Algorithm 2 is* $O(K)$.

Proof. Let K be the number of edges to be stored in the memory. The algorithm maintains an edge reservoir S with the maximum size limit of K. Hence, the total memory consumed by the algorithm will be $O(K)$.

It can be formally verified that both NHMS and THS produce unbiased estimates when the reservoirs used are not full. When the reservoirs are full, we still do not observe any bias in the estimations produced by our algorithms, and in practice, our estimations are equal or better than those produced by FM and FP algorithms. While we are not able to show the unbiasedness of our algorithms formally, based on our extensive experiments with large datasets and reservoir sizes of only 1% of the datasets, we conjecture that our algorithms are unbiased or exhibit negligible bias.

5 Experimental Evaluation

5.1 Experimental Settings

Table 1: Summary of real-world graphs

Dataset	Nodes	Edges	Triangles
enron	69,244	254,449	1,067,993
cnr	325,557	2,738,969	20,977,629
dblp	986,324	3,353,618	7,005,235
dewiki	1,532,354	33,093,029	88,611,129
ljournal	5,363,260	49,514,271	411,155,444
arabic	22,744,080	553,903,073	36,895,360,842
twitter	41,652,230	1,202,513,046	34,824,916,864

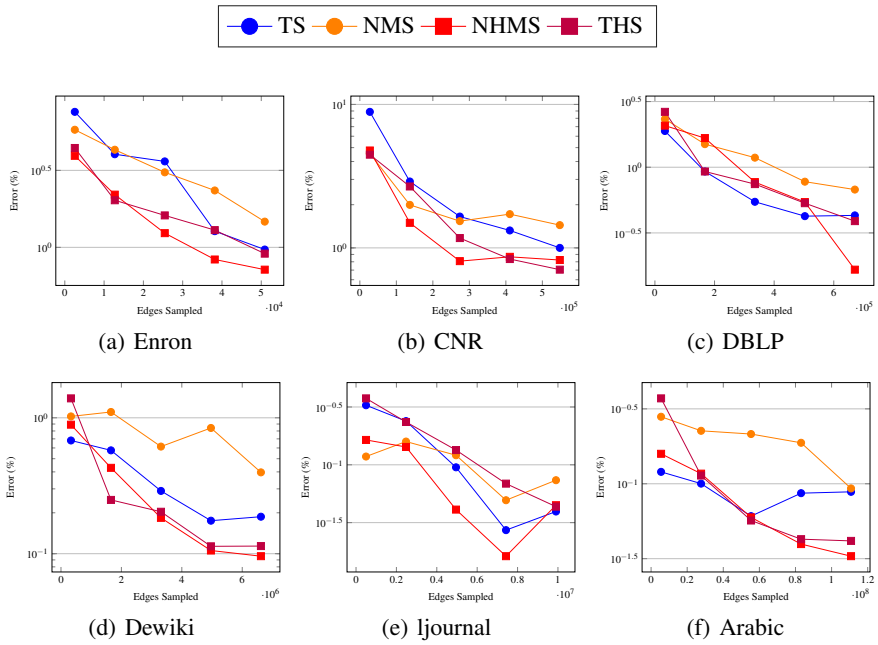
We implemented and executed all algorithms in Java 8. We also used Webgraph framework [2] because of its great compression ratio in saving or loading graphs. Even though the Xeon server had 64GB RAM, we did not change the default memory settings of JVM. Server JVM heap configuration is 1/4 of the total System memory available. Hence, the allocated memory that can be used by the whole implementation of algorithm will be the maximum of 16GB. We had to tweak the JVM heap size for running

the original NMS implementation provided by authors on Large graphs like Arabic and Twitter, as it creates a lot of objects and does not scale up in 16GB RAM.

We perform the evaluation of these algorithms on six real word datasets of varied sizes. All these datasets have been downloaded from the Laboratory of Web Algorithms which provides the compressed form of large datasets using WebGraph [1]. These graphs are then symmetrized and any self-loop is deleted to get the corresponding simple undirected graphs. Table 1 shows the summary of real world graphs used for the experiments.

The comparison of all the algorithms is done considering the total number of edges sampled by the respective algorithms. We followed this criteria to get a fair comparison of all the algorithms (irrespective of FP, FM, or Hybrid approach) as to figure out how the performance of the algorithms looks like when they store the same number of edges. Based on this, our results are discussed in the next section.

5.2 Accuracy



(a) Enron (b) CNR (c) DBLP

(d) Dewiki (e) ljournal (f) Arabic

Fig. 1: Error Rate for algorithms TS, NMS, NHMS and THS

We ran an experiment by sampling 1%, 5%, 10% 15% and 20% of the graph size respectively on THS, NHMS, TS and NMS algorithms and compared their accuracy. The results are plotted in Figure 1 for all the graph datasets we used. Clearly, both THS and NHMS have better accuracy than their counterparts. It is important to note that the

difference in accuracy is not bigger in large graphs as the error rate is below 0.5% for all the test cases. Even if TS and NMS algorithms are highly accurate, NHMS and THS still have even better accuracy.

5.3 Running time

We ran an experiment in similar setting to accuracy *i.e.* by sampling 1%, 5%, 10% 15% and 20% of the graph size respectively on THS, NHMS, TS and NMS algorithms and compared their running time. We ignored the edge arrival time from the input graph stream to accurately measure the run-time performance of the algorithms. The results are plotted in Figure 2 for all the graph datasets that we used.

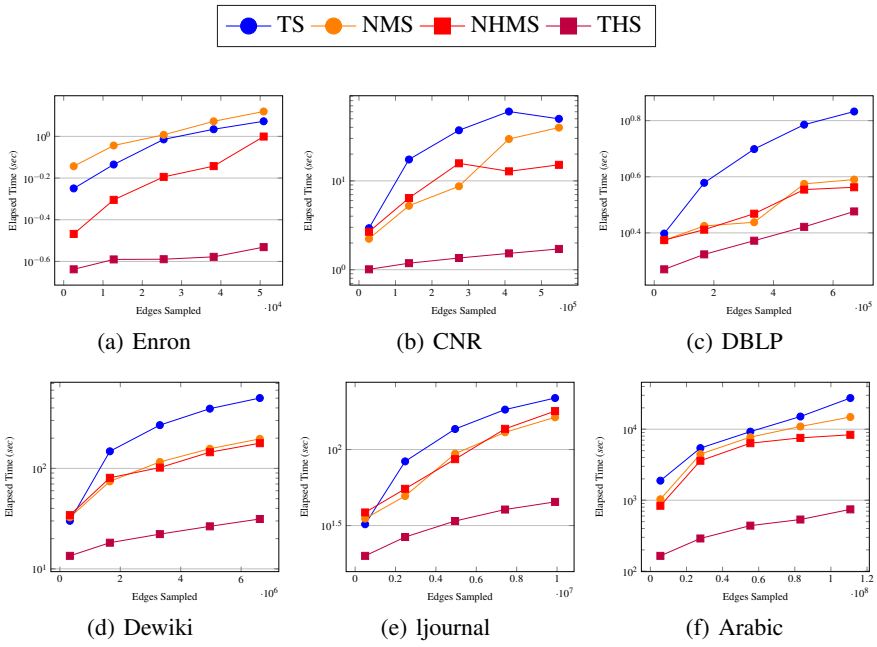


Fig. 2: Run-time for algorithms TS, NMS, NHMS and THS

THS versus TS: THS proves to be **5 - 10X faster** than its counterpart TS. Hence, these results validate our claim that Hybrid algorithms are faster than FM algorithms.

NHMS versus NMS: There is only a slight difference between NHMS and NMS. The reason behind this is that NMS is already an FP algorithm and Hybrid algorithms follow the same execution trend as FP algorithms until the first k edges are processed, after which the increase in time becomes constant. This would be more clear in the next section on scalability. Hence, NHMS is faster than NMS algorithms after the first k edges are processed.

THS versus NHMS: Among our algorithms, THS outperformed NHMS considering the running time, but there's consistently a trade-off among both between run-time and accuracy. NHMS is more accurate whereas THS is much faster than NHMS. **Overall**, both our hybrid algorithms have better running time in comparison to both FM and FP algorithms. THS outperformed all algorithms including NHMS.

5.4 Scalability

To measure scalability, we ran an experiment on twitter that has more than 1.2 billion edges. Data points including time and the number of edges streamed are collected after every 1 million edges in all the algorithms. The results are shown in Figure 3.

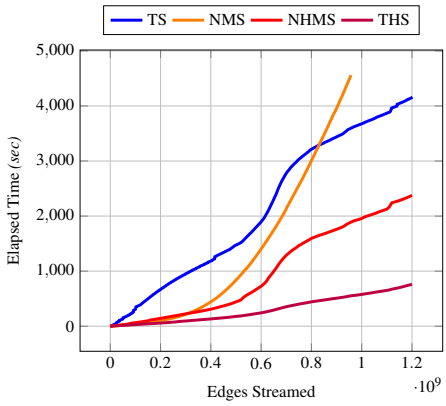


Fig. 3: Scalability Analysis of algorithms on Twitter Graph

Scalability of FP algorithms: FP algorithms are not scalable due to their characteristics. As the edges stored always grows in FP algorithms, elapsed time grows exponentially as the number of edges increases on massive graphs. We can see the results of NMS graph and the algorithm is unable to process whole graph stream and halted even before the billion edges were processed. Hence, NMS is not scalable.

NHMS can scale NMS: As NHMS is the hybrid variant of NMS, it is important to compare both. NMS does not scale on the large graphs, whereas we see NHMS is able to process the whole graph stream. As we have already discussed, NHMS follows the same trend as FP algorithms until the first k edges. We observe that behavior in Figure 3 around 600 million edges, where the curve started to flatten, after a slight exponential rise (similar to NMS).

Scalability of THS and comparison with TS: As seen in Figure 3, THS scales linearly. In comparison to TS that takes around 4200 seconds to execute on Twitter graph, THS just takes 760 seconds that is **5.5X faster**.

Overall, Both hybrid algorithms scale really well for large graphs and are faster in executing the graph stream than both FM and FP algorithms.

6 Conclusion and Future Work

In this study, We prove that FP algorithms are faster than FM algorithms, due to their characteristics that we discussed in Section 3.3. In addition, we present two new algorithms for triangle estimations that belong to the Hybrid category *i.e*. *NHMS* and *THS*. In our evaluation, we observed that NHMS and THS perform a lot better than their counterparts, *i.e*. NMS and TS, in both running time and accuracy. THS proves to be at least **5 times faster** than TS in all the cases with varied graph sizes and different experimental settings. NHMS could not only scale NMS in 16 GB memory, but also is approximately **2 times faster** in most cases.

This research opens up a completely new dimension as we present a new hybrid approach where the community contributions will be proven useful. The theoretical proofs for the accuracy and variance of the estimates of both Hybrid algorithms will provide additional validation of the algorithms. Also, both NHMS and THS can be extended to count sub-graph structures other than triangle.

References

1. Boldi, P., Rosa, M., Santini, M., Vigna, S.: Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In: Srinivasan, S., Ramamritham, K., Kumar, A., Ravindra, M.P., Bertino, E., Kumar, R. (eds.) Proceedings of the 20th international conference on World Wide Web. pp. 587–596. ACM Press (2011)
2. Boldi, P., Vigna, S.: The WebGraph framework I: Compression techniques. In: Proc. of the Thirteenth International World Wide Web Conference (WWW 2004). pp. 595–601. ACM Press, Manhattan, USA (2004)
3. Buriol, L.S., Frahling, G., Leonardi, S., Marchetti-Spaccamela, A., Sohler, C.: Counting triangles in data streams. In: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. pp. 253–262. ACM (2006)
4. Jha, M., Seshadhri, C., Pinar, A.: A space efficient streaming algorithm for triangle counting using the birthday paradox. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 589–597. ACM (2013)
5. Kavassery-Parakkat, N., Hanjani, K.M., Pavan, A.: Improved triangle counting in graph streams: Power of multi-sampling. In: 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM). pp. 33–40. IEEE (2018)
6. Pavan, A., Tangwongsan, K., Tirthapura, S., Wu, K.L.: Counting and sampling triangles from a graph stream. Proc. VLDB Endow. **6**(14), 1870–1881 (Sep 2013). https://doi.org/10.14778/2556549.2556569, http://dx.doi.org/10.14778/2556549.2556569
7. Santoso, Y., Thomo, A., Srinivasan, V., Chester, S.: Triad enumeration at trillion-scale using a single commodity machine. In: Advances in Database Technology-EDBT 2019, 22nd International Conference on Extending Database Technology, Lisboa, Portugal, March 26-29, Proceedings. OpenProceedings. org (2019)
8. Shin, K., Kim, J., Hooi, B., Faloutsos, C.: Think before you discard: Accurate triangle counting in graph streams with deletions. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 141–157. Springer (2018)
9. Singh, P.: Fast and Scalable Triangle Counting in Graph Streams: The Hybrid Approach. Master's thesis, University of Victoria (2020)
10. Stefani, L.D., Epasto, A., Riondato, M., Upfal, E.: Triest: Counting local and global triangles in fully dynamic streams with fixed memory size. ACM Transactions on Knowledge Discovery from Data (TKDD) **11**(4), 43 (2017)