Sampling to Scale: Performance Trade-offs in Approximate Triangle and Square Counting

Shubhashish Kar
Department of Computer Science
University of Nevada, Las Vegas
Las Vegas, USA
kars1@unlv.nevada.edu

Shaikh Arifuzzaman
Department of Computer Science
University of Nevada, Las Vegas
Las Vegas, USA
shaikh.arifuzzaman@unlv.edu

***Abstract*—Exact subgraph pattern matching is a fundamental task in graph mining, but its prohibitive time and memory costs limit large-scale applications. Sampling-based approximation offers a practical alternative by reducing the input graph while providing provable error bounds on the estimated embedding counts.**

In this paper, we conduct a comprehensive, empirical comparison of eight state-of-the-art sampling techniques on large, real-world graphs drawn from diverse application domains. Among local sampling methods, *wedge sampling* consistently delivers the best speed–accuracy balance; within sparsification approaches, *edge sparsification* leads, aside from a few topology-specific outliers. On triangle motifs, wedge sampling attains a $205\times$ speed-up over an exact counter while keeping relative error below 1%. We further quantify how approximation accuracy varies with graph structure, and we report memory footprints to illuminate the trade-offs between speed, accuracy, and resource usage. These insights provide clear guidelines for selecting sampling strategies in large-scale motif analytics.

***Index Terms*—Graph Algorithms; Sampling Methods; Sparsification; Approximation Algorithms; Data Reduction; Performance.**

I. Introduction

Graph, an abstract data structure, is leveraged to formulate and solve numerous real-world problems in the fields of network science, bioinformatics, social networks, and many more [5], [9], [11], [14]. With the increased size of the data, the size of the graph used to formulate the data is growing day by day [1], [17]. Large-scale graph analysis provides a powerful way to extract insights from complex relationships and structures in data and graph analysis can be categorized into graph computation and graph mining. In detail, graph computation problems include the processing of the large-scale graphs leveraging different types of algorithms such as breadth-first search, shortest path, PageRank, to name a few. Whereas, graph mining problems are centered around discovering lower and higher-order structures, specifically finding the complex structural patterns in the large-scale graphs [6], [19]. Of the two classes of graph analysis, graph mining problems are computationally expensive. Graph pattern matching is one of them. In exact graph pattern matching, most of the existing techniques follow subgraph-centric design to match the pattern where the verification of the isomorphism between the subgraph of input graph and pattern plays a crucial role. This problem is considered NP-Hard [21]. On the contrary,

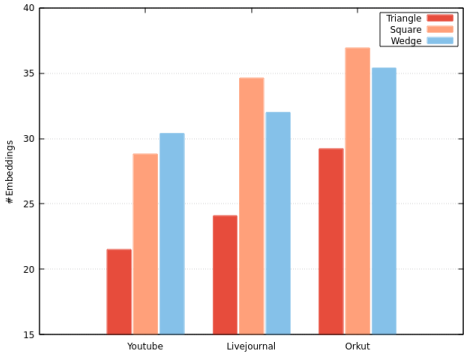


Fig. 1: The number of embeddings for triangle, square, wedges in three real-world datasets. **The number of squares in large graphs surpasses the number of wedges, whereas the number of triangles does not.**

the approximate pattern matching algorithm estimates the number of occurrences of a particular structural pattern in the large real-world datasets providing a specific error limit. Due to the exponential computation complexity of exact pattern matching, various studies cover smaller patterns. Among many smaller patterns, triangle is useful to understand the lower-order structures in the large-scale dataset [4], [18]. To measure the transitivity coefficient and clustering coefficient, triangle counting is heavily used. Likewise, the square motif in the large-scale graphs, helps reveal the higher-order structural patterns in the network. In bipartite graphs, the butterfly structure has a similar adjacency structure to the adjacency structure of squares in the unipartite network. In this study, triangle and square are the representative input patterns and a suite of sampling techniques are utilized to approximate the motif counts in large graphs.

Both triangle and square have symmetry which causes the rapid growth of combinations in exact matching, known as combinatorial explosion, presents a significant difficulty. Fig. 1 demonstrates the increment in the number of occurrences for specific structural patterns including wedge, triangle and square in real-world datasets. The figure is represented in 2-base logarithmic scale to accommodate the large difference in the number of occurrences in a compact fashion.

Furthermore, in recent times, the availability of the entire graph is a point of concern due to the size of the large-networks. If the entire graph is available, then the sparsification

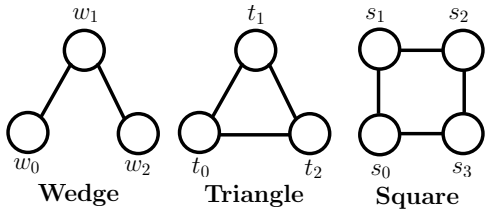


Fig. 2: The **wedge** and the **two templates** (triangle and square) used for evaluating approximation methods.

techniques are applied to get better estimate. In contrast, when the entire graph is unavailable, especially in case of using APIs and gigantic size of the real graphs, the local sampling algorithms are leveraged to get the approximated count. The error metric of the sampling algorithms based on the availability of the entire graph constitutes one of the key objectives of this study.

In summary, the key contributions of this paper are outlined below.

- We conduct comprehensive experimental analysis using a range of sampling techniques on networks from various domains, focusing specifically on triangle and square motifs.
- We analyze the memory-efficiency of the sampling and sparsification solutions along with the parallel-efficiency in the shared-memory systems.
- We assess the error metric of the local sampling solutions in the partitioned graph to measure the approximation scaling for the large-scale graphs.

II. Preliminaries

Given an unweighted unlabeled graph, G, denoted as $G = (V, E)$, where V and E are the set of vertices and the set of edges, respectively. In addition, the graph G has no self-loop and multiple edges. The set of all exact triangles and squares in the entire input graph G are denoted as $\triangle(G)$ and $\square(G)$ respectively. In many scenarios, where the exact number of embeddings is not necessary and there is time constraint, then the approximate count of embeddings with the error bound is sufficient. In sampling-based algorithms, For parameters $\epsilon, \delta \in [0, 1]$ where ϵ is the approximation ratio and δ is the failure probability, an (ϵ, δ) - approximation of a number x is a random variable $\hat{x}$ such that $Pr(|\hat{x} - x| > \epsilon x) \leq \delta$. In this study, the counts of non-induced triangle and square are approximated through different sampling-based techniques. The representative input patterns are triangle and square, specifically, they are 3-cycle and 4-cycle graphs. In the two representative patterns of Fig. 2, wedge is a common building block.

A. Sampling in Massive Graphs

Scale-down sampling goal aims to generate a sampled graph denoted as G', consisting of n' nodes, while the original graph is $G = (V, E)$ consisting of $|V| = n$ nodes. The sampled graph preserves the condition $n' \ll n$. In addition, the sampled graph G' should preserve the structural properties of the original static graph G, such as the degree distribution, the clustering coefficient, the strongly/weakly connected components, and so on. In [13], sampling techniques designed for this scale-down goal broadly fall into three categories: random node sampling, random edge sampling, and random walk-based approaches. For random node selection, the method selects a set of nodes leveraging uniform random sampling and the graph induced by the set of nodes is the sampled graph. This method does not necessarily retain the power-law characteristic of the original graph. In case of random edge selection, the uniform edge sampling is quite similar to the uniform node sampling. In general cases, the resultant graph is sparse and does not retain the community structure of the original graph. In contrast, non-uniform sampling assigns probabilities to nodes and edges according to different attributes, such as in the random walk–based method.

B. Sampling vs Sparsification Algorithms

Real-world graphs are continuously growing in size, in terms of the number of nodes and the relationship among the nodes. Various social networks offer APIs, such as X-API [23] and the Facebook API [7], to facilitate data access. Often these APIs are rate-limited and the gigantic graph runs short of memory. Based on the accessibility of the input graph, there are two approaches to reducing its size for approximation purposes, for instance, local sampling and sparsification. Local sampling algorithms are capable of operating effectively even with restricted access to the input graph. These algorithms leverage the idea of sampling on components such as vertex, edge, or wedge in small subgraphs of the input graph. Later, the count in the small subgraph is scaled to the approximated amount in the entire input graph. In contrast, the sparsification algorithms have the entire view of the graph and they thin down the input graph and computation on the reduced graph. The eligibility of the edges to be in the reduced graph is dictated by randomized procedure. Later, it derives an estimated value based on the exact pattern counting method in the reduced size graph. Sampling techniques are computationally less expensive in terms of runtime and memory consumption compared to their exact and sparsification counterparts. But sparsification is a much better performer in approximation.

III. Related Work

In recent years, the majority of approximation methods in graph mining have been built upon uniform random sampling. These approaches utilize random sampling across various structural components of the graph, effectively minimizing bias by ensuring that each element has an equal probability of selection. Besides, triangle and square are among the most frequently occurring substructures in real-world graphs. As a result, many studies are directed towards the approximation of basic and high-order substructures in the large-scale graphs.

Overall, edge sampling and wedge sampling emerge as the most effective approaches. Wu et al. [22] demonstrated several random sampling algorithms for approximating triangle counts in graphs. Among these, they analyzed the theoretical error bounds of methods such as vertex sampling, edge sampling, and others. Additionally, they evaluated the performance of

these sampling techniques across different data structures, including edge arrays and adjacency lists. Their findings indicate that edge sampling achieves superior performance without the need for prior preprocessing. Sheshadri et al. [18] discussed computing the triangles by T_d wedge sampler which samples the vertices based degree d. Then the wedges hinging at the vertex v are checked whether they are open or closed. Total number of wedges associated with d-degree vertices are used to compute the number of triangles. They claimed that the number of samples needed to meet a given error margin and confidence level does not depend on the size of the graph. In case of edge sampling, the cubic or quadratic functions for estimating triangles, the algorithms suffer from low probability value. Moreover, the uniform-wedge sampling algorithms provide biased estimations. In [20], the deficiencies of the random edge sampling and random wedge sampling are mitigated by introducing combined edge-based wedge sampling approach. To make the estimate precise, they considered the edges forming wedges at lower degree vertices of previously sampled edge. Along with the flavor of sampling in triangles, some researches are directed towards sparsification techniques. In [4], Arifuzzaman et al. implemented the idea of the parallel sparsification for triangle approximation leveraging the distributed-memory setting. In the implementation, different processes approximate the triangle count on their own partitions and afterwards, the approximated values are aggregated. This implementation achieves 3.8% maximum error in triangle approximation.

Specifically, the square is a 4-cycle motif. Most of the approximation algorithms for 4-vertex patterns can't give better estimate in massive-scale graphs. Jha et al. [10] introduced the concept of centered-three-path sampling to get better estimate for 4-cycle-based motifs. In the empirical analysis, they demonstrated that the solution achieves less than 1% relative error.

Sanei-Mehri et al. [16] introduced the concept of the local sampling algorithms and the one-shot sparsification algorithms from the perspective of availability of the graphs. They run a extensive analysis for approximating the count of complete 2×2 biclique in bipartite networks. Moreover, in most of the approximation algorithms, the large template size causes less accurate estimation. Slota et al. [19] proposed the color coding technique to approximate the subgraph count. They covered the experiments with the template size up to 12 vertices and the data graphs up to million vertices. They also introduced shared-memory based parallelism to implement the counting scheme. In addition, the study [8] involved Random-Walk for the approximation in subgraph mining. In some cases, it is computationally expensive to simulate Random-Walk access model.

IV. Sampling Methods for Triangles and Squares

This section explores various sampling and sparsification techniques employed across a range of large-scale graphs in the study.

A. Vertex Sampling

The technique of vertex sampling [22] is initiated with random selection of vertices from the vertex set, V and building the estimate depending on the number of embeddings (triangle or square) containing the random vertex v. For triangle approximation, this is accomplished by counting the exact number of triangles formed by the edges between the sampled vertex v and its neighbors. In case of the square approximation, it determines the two-hop neighbors from the sampled vertex v and afterwards, it ultimately counts the number of wedges formed by the same two endpoints using $\binom{w}{2}$, where w is the wedge count formed by the same two endpoints.

B. Edge Sampling

The idea of edge sampling [22] focuses on the edges $e \in E$ instead of the vertices as in vertex sampling. The uniform random selection of the edges from the edge set E and the containment of the sampled edge into the number of embeddings is the basis of the approximation. For the case of approximate triangle counting, the algorithm determines the number of triangles formed by the intersection of the neighbors of the two endpoints of the edge. Conversely, for approximate square counting, it determines the wedges formed by the sampled edge e and ultimately counts number of squares based on the same endpoints of the wedges as in vertex sampling.

C. Wedge Sampling

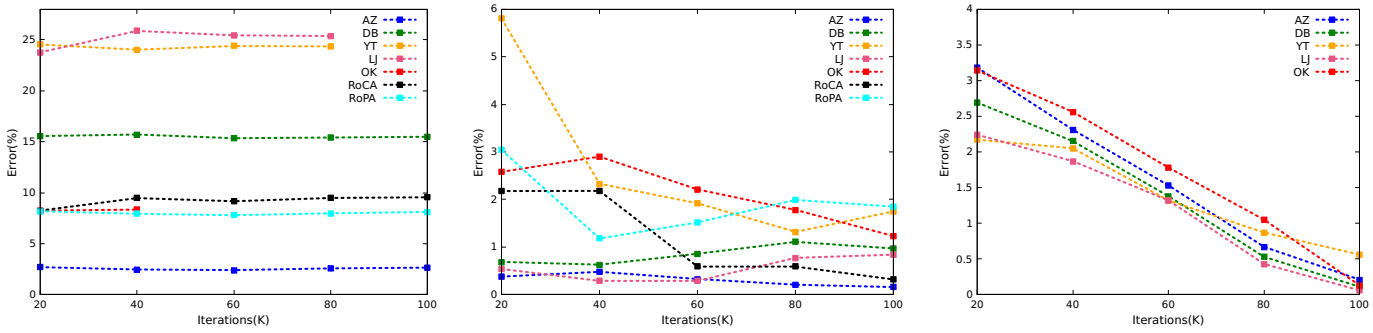
Wedge is a sequence of three vertices and connected by two edges as illustrated in Fig. 2, simultaneously one of the core building blocks both for the triangles and the squares. Unlike sampling only vertices or edges, the wedge sampling [18] is centered around randomly sampling the wedges of the graph. To verify whether the wedges are closed or open is the basis of triangle estimation. In contrast, for square estimation, the number of squares formed by the sampled wedge serves as a basis for extrapolating the square count.

D. Edge-based Wedge Sampling

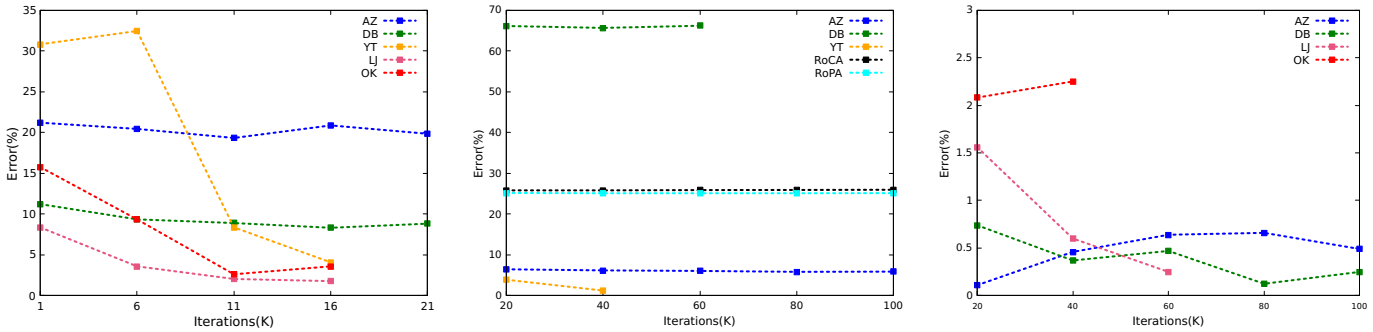
In [20], they proposed a way to combine the approaches of edge and wedge sampling addressing the deficiencies of both algorithms. For triangle approximation in the massive graphs, they sample the edges E' in the first step. Secondly, the lower degree vertex v of the edge $e \in E'$ is considered as the hinging vertex to form the wedges. The last step includes the triangle count increment of $d_v - 1$ (where d_v is the degree of vertex v) provided that the wedge is closed. Whereas, in case of square, the wedges in the second step are leveraged to approximately enumerate the number of squares in the graphs.

E. Neighborhood Sampling

Neighborhood sampling [3] is a basic sampling-based technique for approximating pattern counts on graphs. Firstly, this algorithm adopts an order to match the pattern vertices. This framework starts with uniformly random sampling the edges and then mapping the endpoints of the edges with the ordered pattern vertices. Till the size of the pattern, it randomly



(a) Edge Sampling (b) Wedge Sampling (c) Edge-Wedge Sampling



(d) Neighbor Sampling (e) Edge Sampling (f) Wedge Sampling

Fig. 3: Local Sampling based techniques (a) Triangle Approx. By Edge Sampling - **Maintaining almost constant error**, (b) Triangle Approx. By Wedge Sampling - **Maintaining error** $(0-2)\%$, (c) Triangle Approx. By Edge-Wedge Sampling - **Error close to** 0% (d) Triangle Approx. By Neighbor Sampling - **Varies with Datasets** , (e) Square Approx. By Edge Sampling - **Maintaining almost constant error**, and (f) Square Approx. By Wedge Sampling - **Fluctuating error between** $(0-1)\%$

samples the new edges from the neighborhood of the existing mapped vertices. At last, the increment of the count depends on the condition whether the mapped vertices preserve the adjacency structure of the pattern graph or not.

F. Three-Path Sampling

In [10], they implemented the idea of centered-sampler. This sampling method approximates the square counts in the large-scale graph depending on a construct of three-path. Then, the later phase is the verification that the three-path constitutes a 4-cycle or not. In detail, to enumerate the cycle-based motifs, the algorithm assigns a probabilistic score to each edge based on the degree of the two endpoints and this score is leveraged to sample the edges. Then, the other two edges are selected randomly from the two endpoints from their neighbors, ultimately, forming a three-path. The last step is to make an approximation based on the condition whether the three-path creates a square with an additional edge or not.

G. Sparsification: Edge Based & Color Coding Based

In edge sparsification [16], given the input graph $G = (V, E)$ and the probability of retaining any edge p, the input graph G is sparsified by independently including each of its edges in the sampled graph G' and $V' \subseteq V$, $E' \subseteq E$. In edge-based sparsification, any triangle or square is retained in the sampled graph G' if all the edges of the triangle or the square are retained in G'. In the sparsified graph G', the exact number of embeddings (both for the triangle and the square) provides the estimate for approximated count for triangle or square. The formulation for square is as follows:

$$E[\square(G')] = \sum_{1}^{\square(G)} E[x_i] = \sum_{1}^{\square(G)} Pr(x_i = 1) = p^4 \square(G) \quad (1)$$

In color-based sparsification [16], given the input graph G and number of colors N. The colors are assigned randomly to the vertices. Ultimately, the number of colors dictates the probability p of selecting every edge. Then the edges with the same colored vertices on the two endpoints are retained. Hence, the sparsified graph is achieved and likewise Edge Sparsification, the exact count of the embeddings in the retained graph G' is leveraged to get the approximated number of embeddings in the entire graph G.

$$E[\square(G')] = \sum_{1}^{\square(G)} E[x_i] = (1/N)^4 \square(G) \quad (2)$$

V. Experimental Evaluation

This section is centered around the evaluation of the approximation techniques into a suit of diverse datasets.

A. Graph Datasets

We use nine real-world and synthetic graphs encompassing diverse domains such as social networks, e-commerce, road-networks, and others. These graphs exhibit varying characteristics in terms of the degree and community based properties, thereby offering a robust foundation for conducting comprehensive experiments. The real-world datasets vary in

scale, with vertex counts ranging from 0.3 million to nearly 4 million and more than 34.5 million edges. All datasets were sourced from the Stanford Network Analysis Project (SNAP) repository [1] and the Network Repository [15].

TABLE I: Real-world and Synthetic Graphs Used

Category	Dataset	$\|V\|$	$\|E\|$	d_{avg}
Ecomm.	Amazon(AZ)	0.3M	0.9M	2.03
Social	Youtube(YT)	1.1M	2.9M	5.21
	DBLP(DB)	0.3M	1.0M	4.93
	LiveJournal(LJ)	3.9M	34.6M	17.316
	Orkut(OK)	3M	11.7M	76.28
RoadNet	Road-CA(RoCA)	1.9M	2.7M	2.80
	Road-PA(RoPA)	1.08M	1.54M	2.83
Biological	Human-gene	21.90K	12.30M	1.1K
	Bio-CE-CX	15.2K	0.24M	32
Synthetic	Graph500_18	0.17M	3.8M	43.64
	Graph500_19	0.33M	7.72M	46.10

B. Experimental Environment & Implementation

The code is compiled by **g++ 11.4.0**. We conduct experiments on **Cherry Creek Cluster** with one node consisting of **Intel Xeon E5 - 2697v2 (12 cores)** and **128 GB RAM**.

The entire graph is represented through Compressed Sparse Row (CSR). In some cases, edge list implementation is also used for some sampling techniques. To evaluate the parallel efficiency, we leverage the shared-memory setting and implement in OpenMP. In the experimental evaluation, we adopt time-constrained approach. For approximation accuracy, the median of five runs is selected. The execution times for different sampling solutions are averaged over three runs.

C. Approximation Accuracy

Approximation accuracy or **error** (%) is a crucial metric for any approximation technique in empirical evaluation. In case of both motifs, **vertex sampling** exhibits a higher error rate than the other methods across the majority of social network datasets, road networks, and synthetic datasets. As shown in Figs. 3a and 3e, for both triangle and square patterns, **edge sampling** exhibits an almost constant error (%) in particular datasets as the number of iterations increases. But it performs well in the road networks compared to other techniques. In our empirical analysis, the **wedge sampling** turns out to be one of effective solutions for both motif approximation at a reasonable amount of time. As illustrated in Fig. 3b, the error rate goes less than 1% after $100K$ iterations for social networks. Whereas, for the square motif in Fig. 3f, the error rate fluctuates between $0-1\%$ after $40K$ iterations except for the Orkut dataset. Importantly, wedge sampling demonstrates limited effectiveness in road networks. Although in triangle approximation, **neighborhood sampling** shows lower error rate $(2-3\%)$ in most of the networks, it is not a good option for square approximation in large networks.

Moreover, edge sparsification is a good candidate for triangle and square approximation. The error rate of **edge sparsification**, specifically for square approximation, is demonstrated in Fig. 4b for different networks. In general, all data sets exhibit decent accuracy in the approximation beyond 0.38 probability. Exceptionally, it shows high error(%) for the road networks. In case of **color sparsification**, as demonstrated for triangle approximation in Fig. 4a, all networks show a similar trend of reducing the error rate from a large value to nearly 1% with the decrease in the number of colors, overall, a drastic reduction in the error rate. On the contrary, the scenario for the square motif approximation in Fig. 4c is different compared to the triangle approximation. For most of the social networks in square count estimation, color sparsification exhibits stable error rate after selecting 4 colors(probability 0.25). Likewise **edge sparsification** in road networks, **color sparsification** shows limited accuracy.

D. Parallel-Efficiency

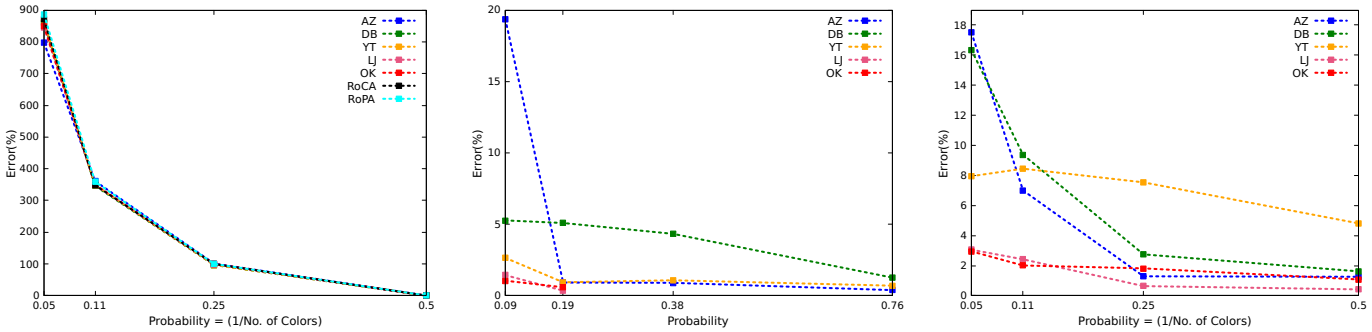
In this study, we introduce parallelism over two sparsification based techniques for triangle and square motifs. We implement these algorithms in shared-memory setting. In edge-sparsification method, parallelism is injected into the sparsification of edges. In case of color sparsification, the colors are assigned to the vertices randomly in parallel fashion. Specifically in both cases of sparsification techniques, the exact counting in the sparsified graph is executed in the parallel manner with dynamic load balancing. In our empirical analysis, performance is evaluated based on the execution time required to achieve the same level of accuracy as the serial implementation. As illustrated in Fig. 5, edge sparsification and color sparsification in case of triangle approximation achieve more than $4\times$ speedup for 16 threads. The scalability varies depending on the adjacency structure and graph properties in the datasets. In shared memory systems, the memory contention problem limits the expected scalability.

E. Memory Analysis

We analyze the memory consumption of wedge sampling and sparsification techniques such as edge sparsification and color sparsification throughout the entire graph computation process. To track the memory usage, we utilize the **Valgrind** [2] tool with **Massif** heap profiler. The Figs. 6 and 7 demonstrate that the local sampling algorithm, **wedge sampling**, consumes far less amount of memory compared to the sparsification techniques. For square approximation, specifically for particular datasets, some processes of sparsification techniques get killed due to the extravagant memory consumption. As illustrated in Figs. 6 and 7, there is a significant disparity in memory usage between local sampling algorithms and sparsification techniques.

F. Approximation Accuracy Based on Random Partition

This metric is specifically employed to assess the approximation error when estimating motif count of entire large-scale graphs using their smaller subgraphs. To the best of our knowledge, this analysis represents a novel aspect of the study. We partition the input graphs with METIS [12] which employs a multilevel scheme that coarsens the graph and partitions the reduced representation, thereby exploiting inherent community structure to minimize inter-partition edges while balancing part sizes. For our social-network datasets, we activate the community-aware options so that the resulting partitions closely follow the networks' natural community organization. Instead of having the global view of the entire input



(a) Color Sparsification (b) Edge Sparsification (c) Color Sparsification

Fig. 4: Sparsification based techniques (a) Triangle Approx. By Color Sparsification - **Drastic reduction in error**, (b) Square Approx. By Edge Sparsification - **Maintaining error** $(0-5)\%$, and (c) Square Approx. By Color Sparsification - **Error reduction with probability increase**

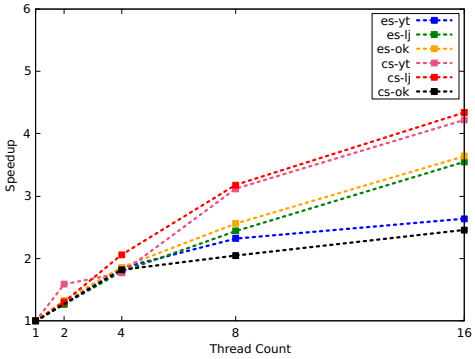


Fig. 5: Scalability for Sparsification Techniques in Triangle Approximation. **Color Sparsification attains more speedup.**

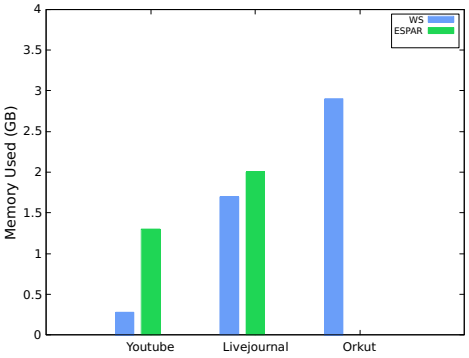


Fig. 6: Memory Consumption for Triangle Approximation. **Consistency in memory consumption among different techniques**

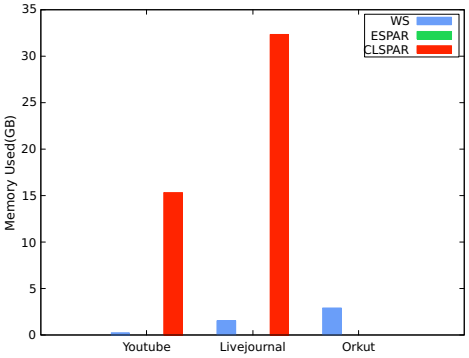


Fig. 7: **Sparsification techniques consume way more memory than local sampling techniques for square approx.**

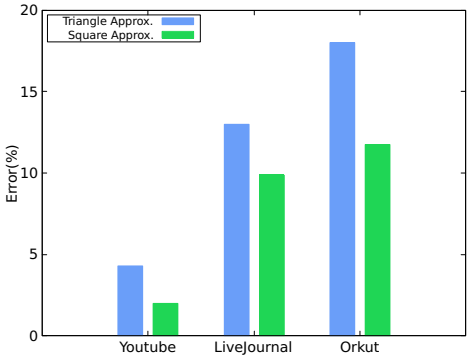


Fig. 8: The error (%) resulting from Approximation in Partition for Wedge Sampling. **Approximation error increases with dataset size**

graph, the local sampling algorithms scale up the pattern count in the partitioned graph to get the approximated amount. We run the best performing local sampling algorithm (considering datasets of diverse domains), **wedge sampling** to the partitions generated by METIS. As illustrated in Fig. 8, for **wedge sampling**, the error (%) exhibits an upward trend with the dataset size for both patterns. In some cases, the error extends up to 18% for triangle and nearly 12% for square.

VI. Conclusion

We present an extensive analysis including error, memory efficiency, and scalability, specifically for counting the triangle and the square in large-scale graphs from various domains. Using a random partitioning strategy, we further present an analysis of the performance of various sampling algorithms with respect to the availability of the entire graph. The **edge-based wedge sampling** outperforms all the other local sampling algorithms in terms of accuracy for triangle approximation in social networks. From the perspective of the local sampling algorithms, **wedge sampling** maintains a decent approximation accuracy with its lower memory overhead for both patterns. **Edge sparsification** demonstrates the superior performance both for approximate triangle and square counting, except in road networks. But it comes with its extravagant memory usage.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant Number 2323533.

REFERENCES

[1] Stanford network analysis project repository. https://snap.stanford.edu/data/.

[2] Valgrind instrumentation framework. https://valgrind.org/.

[3] I Anand, Zaoxing Liu, Xin Jin, Shivaram Venkataraman, Vladimir Braverman, and I Stoic. Asap: Fast, approximate graph pattern mining at scale. In *Proceedings of the USENIX conference*, 2018.

[4] Shaikh Arifuzzaman, Maleq Khan, and Madhav Marathe. Fast parallel algorithms for counting and listing triangles in big graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(1):1–34, 2019.

[5] D.A. Bader and K. Madduri. Parallel algorithms for evaluating centrality indices in real-world networks. In *Proc. of Int. Conf. on Parallel Processing (ICPP)*, 2006.

[6] I. Bordino, D. Donato, A. Gionis, et al. Mining large networks with subgraph counting. In *Proc. of IEEE International Conference on Data Mining (ICDM)*, pages 737–742, 2008.

[7] Facebook. The graph api for the facebook social graph. Online, 2025.

[8] Marco Gori, Marco Maggini, and Lorenzo Sarti. Exact and approximate graph matching using random walks. *IEEE transactions on pattern analysis and machine intelligence*, 27(7):1100–1111, 2005.

[9] Rakibul Hassan and Shaikh Arifuzzaman. An efficient multi-core parallel implementation of SSSP algorithm with decreasing delta-stepping. In *IEEE High Performance Extreme Computing Conference, HPEC 2024*, pages 1–7. IEEE, 2024.

[10] Madhav Jha, C Seshadhri, and Ali Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proceedings of the 24th international conference on world wide web*, pages 495–505, 2015.

[11] Shubhashish Kar and Shaikh Arifuzzaman. Mesm: A query-agnostic and memory-efficient parallel subgraph matching algorithm. In *IEEE High Performance Extreme Computing Conference, HPEC 2024*, pages 1–7, 09 2024.

[12] George Karypis. Metis and parmetis. In *Encyclopedia of parallel computing*, pages 1117–1124. Springer, 2011.

[13] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636, 2006.

[14] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graphs over Time: Densification Laws, Shrinking diameters and Possible Explanations. pages 177–187, 2005.

[15] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[16] Seyed-Vahid Sanei-Mehri, Ahmet Erdem Sariyuce, and Srikanta Tirthapura. Butterfly counting in bipartite networks. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2150–2159, 2018.

[17] Naw Safrin Sattar, Khaled Z Ibrahim, Aydin Buluc, and Shaikh Arifuzzaman. Dyg-dpcd: A distributed parallel community detection algorithm for large-scale dynamic graphs. *International Journal of Parallel Programming*, 53(1):4, 2025.

[18] Comandur Seshadhri, Ali Pinar, and Tamara G Kolda. Triadic measures on graphs: The power of wedge sampling. In *Proceedings of the 2013 SIAM international conference on data mining*, pages 10–18. SIAM, 2013.

[19] George M Slota and Kamesh Madduri. Fast approximate subgraph counting and enumeration. In *2013 42nd International Conference on Parallel Processing*, pages 210–219. IEEE, 2013.

[20] Duru Türkoglu and Ata Turk. Edge-based wedge sampling to estimate triangle counts in very large graphs. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 455–464. IEEE, 2017.

[21] Yihua Wei and Peng Jiang. Stmatch: accelerating graph pattern matching on gpu with stack-based loop optimizations. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13. IEEE, 2022.

[22] Bin Wu, Ke Yi, and Zhenguo Li. Counting triangles in large graphs by random sampling. *IEEE Transactions on Knowledge and Data Engineering*, 28(8):2013–2026, 2016.

[23] X-API. X-API. Online, 2025.