Parallel Local Motif Counting on Large-Scale Dynamic Graphs

Ali Khan
alikhan@my.unt.edu
University of North Texas
Denton, Texas, USA

Sanjukta Bhowmick*
Sanjukta.Bhowmick@unt.edu
University of North Texas
Denton, Texas, USA

Michela Taufer*
mtaufer@utk.edu
University of Tennessee Knoxville
Knoxville, Tennessee, USA

CCS Concepts

• **Mathematics of computing** → **Graph theory**; **Graph algorithms**; **Graph enumeration**; **Graph theory**; **Graph algorithms**; • **Computing methodologies** → **Parallel algorithms**; • **Theory of computation** → **Parallel algorithms**; **Parallel algorithms**.

Keywords

Motif Counting, Graphlets, Dynamic Graphs, Parallel Algorithm

ACM Reference Format:
Ali Khan, Sanjukta Bhowmick, and Michela Taufer. 2025. Parallel Local Motif Counting on Large-Scale Dynamic Graphs. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'25).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Graphlets or motifs are subgraphs of size-k vertices. Structural properties of large networks, such as those arising in diverse areas including biology[9], social network analysis [6] and identifying non-determinism in HPC executions [2], can be determined by computing the frequencies (counts) of the motifs occurring in them [11].

As the network sizes increase it becomes essential to use parallel computing to compute the frequencies of the motifs. Further, many of the real world networks described above change with time, with vertices and edges being added or deleted. This dynamicity exacerbates when the computational challenges. The static motif counts become out of date and recomputing the counts after each batch of update leads to redundant computations.

We build on top of previous work [5] to compute local counts of motifs. Instead of determining the frequency of motifs in a network globally, we compute the frequency of the motifs incident to each edge. Global network-level counts are useful for comparing and characterizing networks, but local network properties allow more fine-grained analysis. For example, classic graph tasks such as node or edge property classification and prediction as well as link prediction both rely on local graph characteristics.

Our contributions are the following: **(1)** we develop a new parallel algorithm to compute $k \leq 4$ local motif counts on dynamic networks, and **(2)** we demonstrate the scalability of our system on shared memory systems with OpenMP as well as GPUs. Results show that our algorithm can be as much as 10 times as fast as the baseline algorithm used for the static graph.

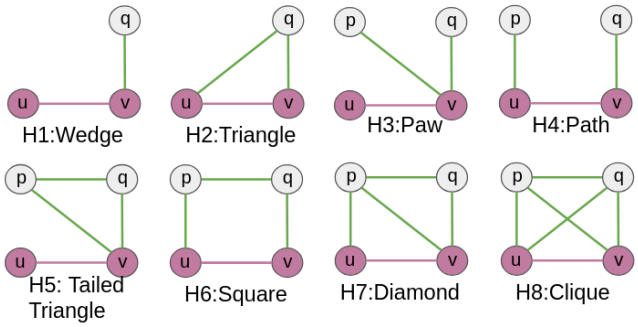

Figure 1: Motifs with $3 \leq k \leq 4$: $k = 3$ motifs: Wedges (H_1) and Triangles (H_2); $k = 4$ motifs: Paw (H_3), Path (H_4), Tailed Triangle (H_5), Square (H_6), Diamond (H_7), and Clique (H_8).

2 Dynamic Subgraph Update

A global count of a motif in a network gives the frequency of the motif in the entire network e.g., the total number of triangles (H2 in Fig. 1) in a network. This information can characterize the entire network. For network alignment and biology, however, a more granular count, called a local count, is sometimes necessary [9]. By computing the frequency of a motif that each vertex or edge in a network participates in, and by exploiting the orientation and symmetries of the motif, called orbits, we can characterize the local topological neighborhood. For example, instead of counting the frequency of H5 in Figure 1 we count the motifs incident to the edges (u, v), (p, v) and (p, q). Edge (q, v) is symmetric to edge (p, v) and can be ignored. Inspired by other static motif counting algorithms [3, 5, 9], by careful enumeration we can compute a set of motifs of size k vertices or less in a single pass. For example, a Paw (H3 in Fig. 1) is composed of two wedges (H1). Likewise adding the edge (h, q) to a Diamond (H7) creates a Clique (H8).

2.1 Overview of Our Method

We handle batched dynamic updates of edges without recomputing the frequencies of the static network. Our algorithm requires only additional space for the storage of the updates. The complexity of computing motifs by similar static algorithms [1, 3], is proportional to the number of edges times the maximum degree of the graph. In contrast, our algorithm only requires time proportional to the number of edges changed (inserted/deleted), and the highest degree of the subgraph induced by these edges and their neighbors. However, if the ratio of changed edges to the total number of edges in a network is sufficiently large, then the static algorithm may be faster.

*advisor

SC'25, November 16–21, 2025, St. Louis, MO

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2.2 Algorithm

We now provide a brief description of our algorithm. Similarly to [5], for each updated edge, we first enumerate all neighbors of distance-1 from the updated edge, marking the type of relationship of the vertices to the edge. Next, we similarly enumerate all distance-2 neighbors of the updated edge. Then, using the information from the previous two steps, we can compute the frequencies of motifs incident to the updated edge in constant time.

2.3 Parallelization

Since each each updated edge can be processed independently, we can parallelize the outer loop of our algorithm. Using Kokkos C++ Performance Portability Ecosystem [12], we can take advantage of the parallelism offered by hardware. For shared-memory systems, we use the OpenMP backend with dynamic scheduling and specify how many CPU cores to use. We use the Nvidia CUDA backend to run our algorithm on the GPU, and take advantage of the GPU's nested parallelism.

3 Experiments

We evaluate our algorithm against the PGD library [1] for different synthetic dynamic networks constructed from the set of real world graphs obtained from NetworkRepository[10]. We construct batch of 1 million synthetic updated edges using different ratios of insertions and deletions: 0% insertions and 100% deletions, 50% insertions and 50% deletions, and 100% insertions and 0% deletions.

Our approach, compared to a state-of-the-art algorithm for static motif counting, performs on average 12x faster with a best-case speedup as much as 124x faster on OpenMP (Fig. 2) on a dataset with 3 million vertices and 106 million edges. We also highlight the performance improvement of the GPU runtime with respect to OpenMP (Fig. 3).

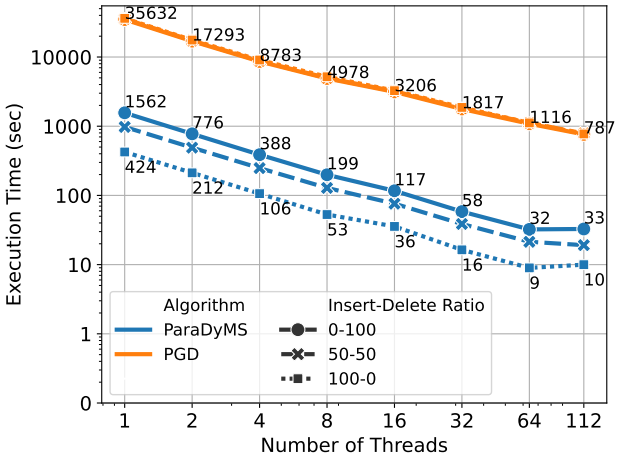

Figure 2: Performance analysis of our algorithm and PGD using OpenMP on Orkut dataset. Execution times are compared across thread counts with ratios of insertions to deletions: 0%-100%, 50%-50%, and 100-0.

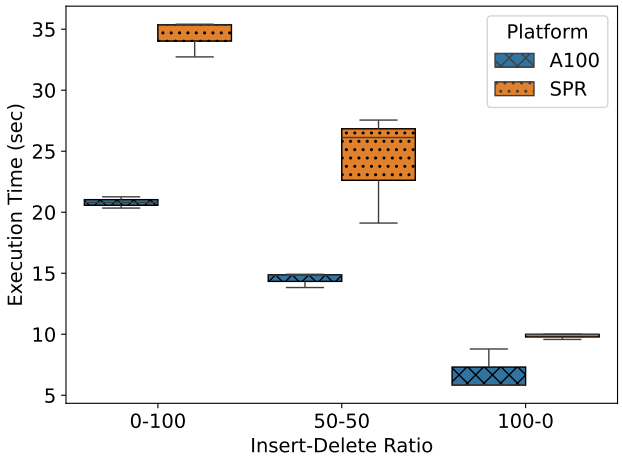

Figure 3: The performance analysis of our algorithm using an Nvidia A100 GPU compared to shared-memory (OpenMP with 112 threads) on Orkut dataset with datasets using Intel Sapphire Rapids (SPR).

4 Related Work

ORCA [4], presents a system of equations that be used to count the local frequencies of motifs of size k using the counts of motifs of size $k-1$. The Parallel Graphlet Decomposition (PGD) library [1, 3], improves the edge-based global and local counts of motifs of $k = 4$ based on the decomposition of the motifs and careful enumeration of wedges and triangles. ESCAPE [8] is the current state of the art method for $k <= 5$ on vertex-based global counts of induced and non-induced size $k <= 5$ graphlets. Among the decomposition based method for graphlet census, few take advantage of GPUs, shared memory systems [1, 3, 7], or distributed memory systems[13], and almost none compute $k <= 4$ motifs for dynamic graphs.

5 Conclusion

We demonstrate a scalable method for counting $k \leq 4$ motifs locally on large dynamic graphs. Our results show a potential speedup of 124 times over a state-of-the-art static motif counting algorithm. While the GPU demonstrates higher performance compared to OpenMP, due to its limited memory constraint, we were not able to fully utilize the GPU's parallelization. We intend to address this by first adapting the data structures used for GPU computation for this memory constraint, and second, offload irregular computation to the CPU (OpenMP) to perform instead of the GPU.

Acknowledgments

We would like to acknowledge Nigel Tan and Jack Marquez for their assistance. This work is supported by NSF grants 1900888, 1900765 and 2341588. Support of XSEDE and IBM through a Shared University Research Award is also acknowledged.

References

[1] Nesreen K Ahmed, Jennifer Neville, Ryan A Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *2015 IEEE international conference on data mining*. IEEE, 1–10.

[2] Sanjukta Bhowmick, Patrick Bell, and Michela Taufer. 2023. A survey of graph comparison methods with applications to nondeterminism in high-performance computing. *The International Journal of High Performance Computing Applications* 37, 3-4 (2023), 306–327.

[3] Vachik S Dave, Nesreen K Ahmed, and Mohammad Al Hasan. 2017. E-CLoG: Counting edge-centric local graphlets. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 586–595.

[4] Tomaž Hočevar and Janez Demšar. 2014. A combinatorial approach to graphlet counting. *Bioinformatics* 30, 4 (2014), 559–565.

[5] Ali Khan, Nigel Tan, Jack Marquez, Michela Taufer, and Sanjukta Bhowmick. 2025. ParaDyMS: Parallel Dynamic Motif Counting at Scale. In *2025 IEEE 25th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 01–10.

[6] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.

[7] Noujan Pashanasangi and C Seshadhri. 2020. Efficiently counting vertex orbits of all 5-vertex subgraphs, by evoke. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 447–455.

[8] Ali Pinar, Comandur Seshadhri, and Vaidyanathan Vishal. 2017. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th international conference on world wide web*. 1431–1440.

[9] Nataša Pržulj. 2007. Biological network comparison using graphlet degree distribution. *Bioinformatics* 23, 2 (2007), e177–e183.

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

[11] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*. PMLR, 488–495.

[12] Christian R. Trott, Damien Lebrun-Grandié, Daniel Arndt, Jan Ciesko, Vinh Dang, Nathan Ellingwood, Rahulkumar Gayatri, Evan Harvey, Daisy S. Hollman, Dan Ibanez, Nevin Liber, Jonathan Madsen, Jeff Miles, David Poliakoff, Amy Powell, Sivasankaran Rajamanickam, Mikael Simberg, Dan Sunderland, Bruno Turcksin, and Jeremiah Wilke. 2022. Kokkos 3: Programming Model Extensions for the Exascale Era. *IEEE Transactions on Parallel and Distributed Systems* 33, 4 (2022), 805–817. https://doi.org/10.1109/TPDS.2021.3097283

[13] Hao Zhang, Jeffrey Xu Yu, Yikai Zhang, Kangfei Zhao, and Hong Cheng. 2020. Distributed subgraph counting: a general approach. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2493–2507.