An Empirical Study of Kernel-Aware Graph Sparsification for Parallel BFS and SSSP

Rakibul Hassan and Shaikh Arifuzzaman
Department of Computer Science
University of Nevada, Las Vegas
Las Vegas, NV, USA
Emails: hassar2@unlv.nevada.edu, shaikh.arifuzzaman@unlv.edu

***Abstract*—Breadth-First Search (BFS) and Single-Source Shortest Path (SSSP) are fundamental graph traversal kernels in large-scale graph analytics. Graph sparsification reduces graph size to improve memory efficiency and traversal performance, making it particularly attractive for applications that repeatedly execute graph queries on the same network. However, the impact of different sparsification strategies on parallel graph traversal performance and traversal quality remains largely unexplored. In this paper, we present a comprehensive performance–quality study of five representative graph sparsification methods for parallel BFS and SSSP kernels on shared-memory multicore systems. Using large real-world and synthetic graphs, we evaluate the tradeoff between traversal acceleration and graph quality through reachability preservation, distance stretch, and graph connectivity. Our results demonstrate kernel-aware sparsification behavior: Common-Neighbor provides the strongest connectivity preservation, whereas Weight-Aware best preserves weighted shortest paths. Sparsification significantly accelerates graph traversal, providing up to $3.30\times$ speedup for parallel BFS and $3.85\times$ speedup for parallel Delta-Stepping (SSSP) with method-dependent tradeoffs between performance, reachability, and distance preservation. These results show that sparsification should be selected according to the target graph kernel rather than using a one-size-fits-all strategy.**

***Index Terms*—Graph Sparsification, Breadth-First Search, Single-Source Shortest Path, Parallel Graph Algorithms, Shared-Memory Computing.**

I. Introduction

Graphs are widely used to model complex relationships in numerous application domains, including social networks [10], [20], transportation systems [5], communication networks [3], biological networks [4], cybersecurity [2], and scientific computing [13]. As graph datasets continue to grow to billions of vertices and edges, efficient graph processing has become a fundamental challenge for high-performance computing (HPC) systems. The massive memory footprint and irregular computation of large-scale graphs often limit scalability [16], motivating techniques that reduce graph size while maintaining useful application characteristics.

Graph sparsification addresses this challenge by reducing the number of edges while attempting to preserve important structural properties of the original graph [9]. A smaller edge set can reduce memory traffic and edge-processing work, making sparsification attractive for accelerating large-scale graph analytics. Existing sparsification methods are typically evaluated using graph-theoretic properties such as connectivity, spectral similarity, or downstream application accuracy.

Among graph analytics kernels, Breadth-First Search (BFS) and Single-Source Shortest Path (SSSP) are two of the most fundamental graph traversal algorithms [1], [11]. They serve as building blocks for numerous applications, including graph exploration, routing, centrality analysis, network optimization, and many higher-level graph algorithms [8]. Although graph sparsification reduces graph size, preserving graph structure alone does not necessarily guarantee that traversal algorithms exhibit similar execution behavior or maintain traversal quality. Different graph kernels rely on different structural characteristics, suggesting that the effectiveness of a sparsification strategy may depend on the target algorithm. Despite the growing interest in graph sparsification, there has been little systematic study of how different sparsification strategies influence both the performance and quality of parallel graph traversal kernels.

In this paper, we present a unified graph sparsification framework for systematically evaluating representative sparsification methods with parallel BFS and Delta-Stepping SSSP on shared-memory systems. Experiments on real-world and synthetic graphs show how different sparsification methods influence graph traversal in terms of speedup, reachability, graph fragmentation, and distance stretch. The primary contributions of this work are summarized as follows:

- We develop a unified shared-memory framework integrating five representative graph sparsification methods with parallel BFS and Delta-Stepping SSSP.
- We conduct a comprehensive experimental evaluation using large real-world and synthetic graph datasets, analyzing traversal speedup, reachability preservation, graph fragmentation, and distance stretch.
- We demonstrate that graph sparsification exhibits *kernel-aware* behavior. Common-Neighbor sparsification consistently provides the strongest preservation of graph connectivity whereas Weight-Aware sparsification achieves superior shortest-path preservation.

II. Related Work

Graph reduction techniques are commonly categorized into graph coarsening, graph condensation, and graph sparsification, each targeting different tradeoffs between graph size and information preservation [14]. Existing graph sparsification

methods can generally be divided into two categories: theoretical spectral sparsifiers and lightweight combinatorial heuristics. Early theoretical work by Benczúr and Karger [7] introduced cut sparsification through randomized edge sampling, while Spielman and Srivastava [21] established the foundation of spectral sparsification using effective resistance sampling to approximately preserve the graph Laplacian. Although these methods provide strong theoretical guarantees, their computational complexity often limits their applicability to very large graphs [12]. Consequently, practical graph analytics frequently rely on lightweight heuristics based on local graph structure, including degree-aware pruning, common-neighbor preservation, bridge-aware edge selection, and distance-preserving graph constructions such as spanners [9], [18].

Recent studies have demonstrated that different sparsification methods preserve different graph properties and downstream application behavior [9]. However, existing evaluations primarily focus on graph-theoretic metrics, spectral similarity, or downstream machine learning tasks, with relatively limited attention given to parallel graph traversal kernels. In particular, a systematic performance–quality comparison of representative sparsification methods for parallel Breadth-First Search (BFS) and Delta-Stepping Single-Source Shortest Path (SSSP) on shared-memory multicore systems remains largely unexplored. This work addresses the limited study of parallel traversal kernels by comparing lightweight edge-based heuristics within a unified framework for BFS and Delta-Stepping SSSP. We evaluate traversal performance, reachability preservation, graph fragmentation, and distance stretch, demonstrating that the effectiveness of graph sparsification depends on the target graph traversal kernel.

III. Methodology

Figure 1 illustrates the overall workflow of our proposed evaluation framework. The input graph is initially constructed from the edge list and represented using adjacency-list and Compressed Sparse Row (CSR) data structures. The original graph is retained as the reference baseline throughout the evaluation. A selected graph sparsification method is then applied to generate a reduced graph by removing a target fraction of edges. Our framework implements five representative sparsification strategies, ranging from random edge removal to graph traversal kernel-aware methods that exploit topological and edge-weight information to guide edge selection. The sparsified graph maintains the original vertex set but contains a reduced edge set according to the selected sparsification strategy. To evaluate the effect of sparsification, both the original and sparsified graphs are processed using the same parallel graph kernels, which are Parallel Breadth-First Search (BFS) [6] and Parallel Delta-Stepping [11]. All executions use identical source vertices, thread configurations, and algorithmic parameters to ensure a fair comparison. The outputs of the traversals are subsequently compared to quantify both graph quality and computational performance.

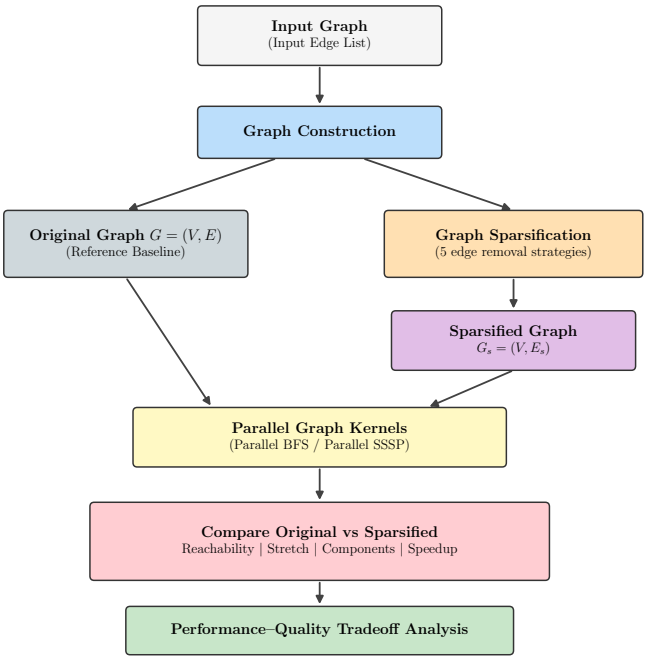


Fig. 1. Overview of the proposed graph sparsification evaluation framework. The original and sparsified graphs are processed using identical parallel graph kernels, and the resulting quality and performance metrics are compared.

A. Graph Sparsification Framework

Let the input graph be represented as $G = (V, E)$, where V denotes the vertex set and $E = \{(u, v, w) \mid u, v \in V,\ w > 0\}$ is the set of weighted edges connecting vertices u and v with edge weight w. Given a target sparsification ratio $\rho \in [0, 1)$, the objective is to construct a sparsified graph $G' = (V, E')$, where $E' \subseteq E$, and $|E'| = (1 - \rho)|E|$. In this study, each sparsification method is evaluated at sparsification ratios of $\rho \in \{0.05, 0.10, 0.20, 0.30, 0.40, 0.50\}$, corresponding to up to 50% edge removal. Figure 2 illustrates the proposed graph sparsification framework. The edge set is first partitioned into two disjoint subsets, $E = P \cup C$, $P \cap C = \emptyset$. Here P denotes the protected edge set and C denotes the candidate edge set. Protected edges are always retained because they satisfy the preservation criterion of the selected sparsification strategy, whereas candidate edges are eligible for removal. Candidate edges are ranked using a method-specific criterion and removed until the target ratio is reached or no eligible edges remain. All methods achieve the requested ratios except Common-Neighbor, whose protected-edge constraint can limit the achievable removal.

B. Graph Sparsification Methods

We implement five representative graph sparsification methods within the unified framework described above. Although all methods follow the same sparsification pipeline, they differ in how protected edges are identified and how candidate edges are prioritized for removal. Table I summarizes the characteristics of the implemented methods.

TABLE I
SUMMARY OF THE IMPLEMENTED GRAPH SPARSIFICATION METHODS.

Method	Protection Criterion	Edge Selection Strategy
Random	No protected edges.	Uniformly shuffle all edges and randomly remove the required number of edges to achieve the target sparsification ratio.
Degree-Protected	Preserve edges incident to low-degree vertices.	Prioritize candidate edges according to the minimum degree of their endpoints, removing edges connecting highly connected regions first.
Common-Neighbor	Preserve edges whose endpoints share no common neighbors.	Rank candidate edges by decreasing common-neighbor count, preferentially removing structurally redundant edges within dense local neighborhoods.
Bridge-Aware	Preserve graph bridge edges identified using Tarjan's algorithm.	Rank remaining candidate edges according to the minimum endpoint degree while ensuring graph bridges are never removed.
Weight-Aware	Preserve edges incident to low-degree vertices.	Prioritize candidate edges by decreasing edge weight, removing larger-weight edges before smaller-weight edges to better preserve weighted shortest-path quality.

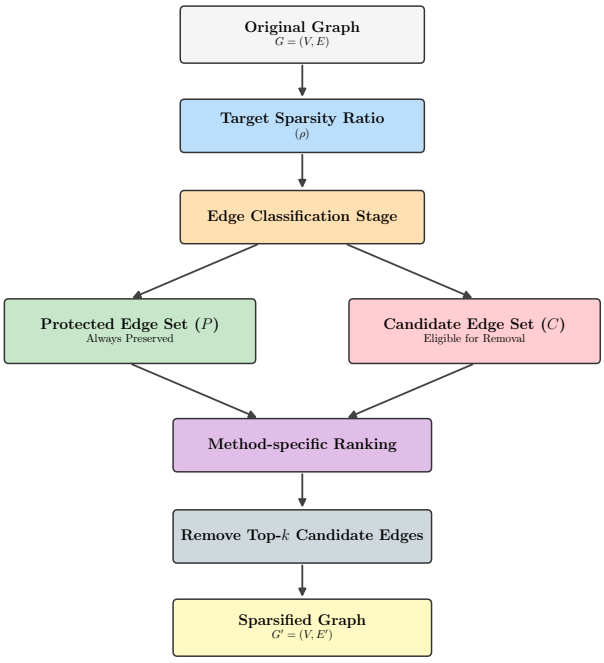


Fig. 2. Unified graph sparsification framework. The original graph is partitioned into protected and candidate edge sets according to the selected sparsification strategy. Candidate edges are ranked using a method-specific criterion and removed until the target ratio is reached or no eligible edges remain.

Random sparsification uniformly removes edges and serves as an inexpensive baseline. **Degree-Protected** preserves edges incident to low-degree vertices and preferentially removes edges in highly connected regions. **Common-Neighbor** protects edges with no common neighbors and prioritizes remaining edges by decreasing neighborhood intersection size, removing structurally redundant edges in dense regions. The protected set can prevent achieving the requested removal ratio. **Bridge-Aware** explicitly preserves bridge edges identified using Tarjan's algorithm and ranks the remaining edges according to endpoint degree to minimize graph fragmentation. Finally, **Weight-Aware** protects edges incident to low-degree vertices and removes larger-weight edges first, aiming to better preserve weighted shortest paths during SSSP traversal.

C. Parallel Graph Traversal Kernels

To evaluate the impact of graph sparsification on graph traversal, we employ two representative parallel graph traversal kernels: Parallel Breadth-First Search (BFS) and Parallel Delta-Stepping Single-Source Shortest Path (SSSP). These kernels are selected because they represent the fundamental traversal primitives for unweighted and weighted graph analytics, respectively, and are widely used as building blocks for many higher-level graph algorithms.

For BFS, we implement a shared-memory parallel top-down traversal approach [6] using OpenMP, where the frontier vertices are processed concurrently at each level. The algorithm computes the shortest hop distance from a given source vertex to all reachable vertices and serves as the basis for evaluating reachability preservation, graph connectivity, and hop-distance stretch after sparsification.

For weighted graphs, we employ the optimized OpenMP-based parallel Delta-Stepping implementation proposed by Hassan *et al.* [11], which builds upon the original Delta-Stepping algorithm of Meyer and Sanders [17]. The implementation combines bucket-based parallel relaxation with decreasing Δ stepping to improve traversal performance on shared-memory multicore systems. To ensure a fair comparison, the original and sparsified graphs are executed using the same source vertex, thread count, and algorithm-specific parameters, including the Δ value for Delta-Stepping.

IV. PERFORMANCE EVALUATION

This section presents a comprehensive evaluation of the proposed graph sparsification framework on a diverse collection of real-world and synthetic graphs. These graphs are collected from widely recognized public archives, specifically the Stanford Large Network Dataset Collection (SNAP) [15] and the Network Repository [19]. The objectives of the evaluation are twofold: (1) to quantify the impact of graph sparsification on the performance of parallel graph traversal kernels and (2) to assess the extent to which different sparsification strategies preserve graph traversal quality.

A. Dataset Description

To comprehensively evaluate the proposed framework, we use a collection of large-scale graph datasets consisting of

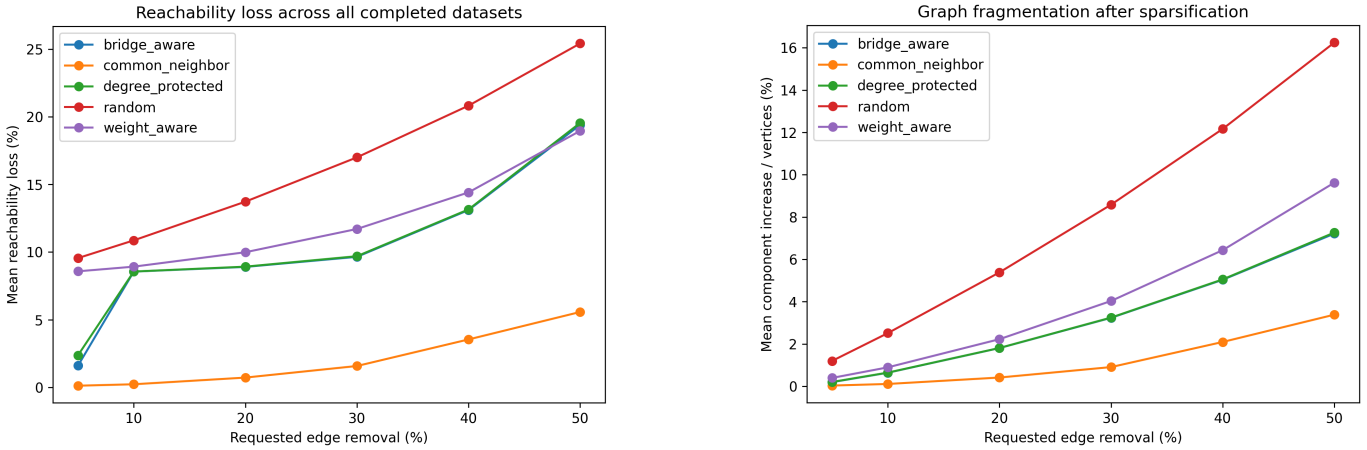


Fig. 3. Structural quality of sparsified graphs measured using average reachability loss (left) and graph fragmentation (right) across all datasets. Lower values indicate better structural preservation. The x-axis denotes requested edge removal; Common-Neighbor may achieve slightly less due to protected edges.

TABLE II
GRAPH DATASETS USED IN THE EXPERIMENTAL EVALUATION.

Dataset	Graph Family	#Vertices	#Edges
Bio-Human-Gene1	Biological	22,283	12,323,680
Com-Amazon	Information Network	548,551	925,872
Com-DBLP	Information Network	425,957	1,049,866
Com-LiveJournal	Social	4,036,538	34,681,189
Com-Orkut	Social	3,072,626	117,185,083
Com-YouTube	Social	1,157,827	2,987,624
Road-Asia	Road Network	11,950,757	12,711,603
SocFB-UCI	Social	58,790,782	92,208,195
Graph500 Kron-20	Synthetic	1,048,576	15,699,588
Graph500 Kron-21	Synthetic	2,097,150	31,767,754
Graph500 Kron-22	Synthetic	4,194,302	64,152,009
Graph500 Kron-23	Synthetic	8,388,608	129,332,875

both real-world and synthetic networks. The real-world graphs represent diverse application domains, including social networks, road networks, web graphs, and biological interaction networks, while the synthetic graphs are generated using the Graph500 Kronecker generator to provide scalable benchmark workloads with different graph sizes.

Table II summarizes the datasets used in this study. The evaluation spans graphs ranging from 22 thousand to nearly 59 million vertices and from 0.9 million to approximately 129 million undirected edges.

B. Experimental Setup

All experiments were performed on a shared-memory Linux workstation equipped with a 13th Gen Intel Core i7-13700 processor featuring 16 CPU cores and 32 GB of main memory. All graph sparsification methods and parallel graph traversal algorithms were implemented in `C++`, with `OpenMP` used to exploit thread-level parallelism. For each dataset, the maximum-degree vertex in the original graph is used as the BFS and SSSP source to favor a large reachable region and substantial traversal workload. The same source is used for the original and all sparsified variants. Random, Degree-Protected, Bridge-Aware, and Weight-Aware achieve

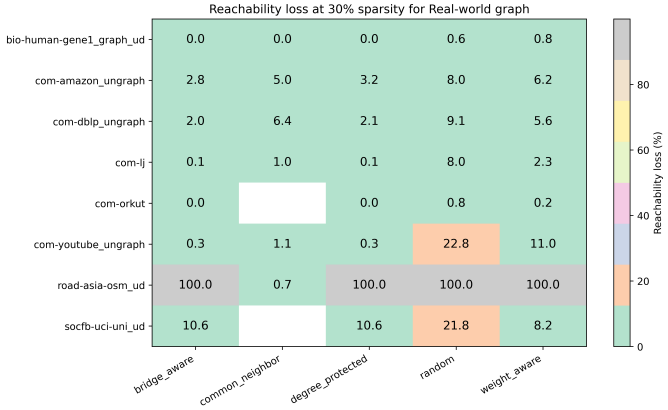


Fig. 4. Reachability loss for individual real-world datasets at 30% sparsification. Road networks are considerably more sensitive to edge removal than social and information networks because their sparse connectivity provides few alternative traversal paths.

the requested removal ratios. Common-Neighbor may achieve less because its protected edges cannot be removed; at 50% requested removal, it achieves approximately 44% on average. Configurations that did not complete within the available memory limit are omitted. Hence, aggregate means may contain slightly different dataset subsets across methods. For originally unweighted datasets, positive edge weights ranging from 1 to 1000 are assigned randomly and kept identical across the original and all sparsified variants.

C. Evaluation Metrics

The proposed graph sparsification framework is evaluated from both performance and graph quality perspectives. Performance metrics quantify the computational benefits of graph sparsification, whereas quality metrics measure the extent to which traversal behavior is preserved after edge removal. We evaluate traversal performance using the runtime of parallel BFS and Delta-Stepping SSSP executed on both the original and sparsified graphs. The speedup achieved through sparsification is computed as $\text{Speedup} = \frac{T_{\text{orig}}}{T_{\text{sparse}}}$, where T_{orig} and T_{sparse} denote the execution times on the original and sparsified graphs, respectively.

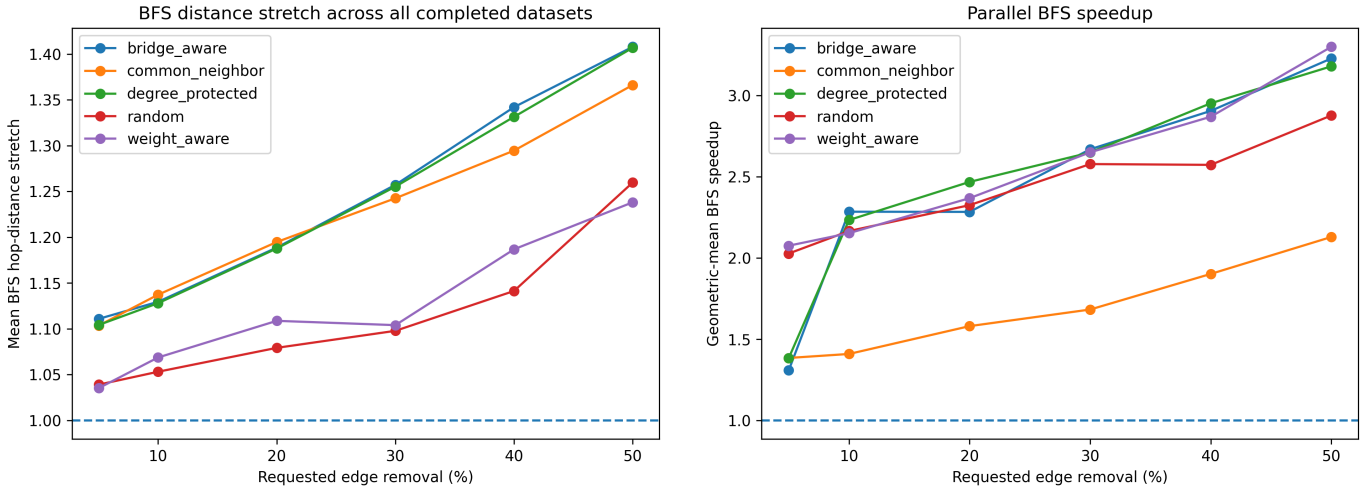


Fig. 5. Performance and quality evaluation of parallel BFS on sparsified graphs. Traversal quality is measured using the average hop-distance stretch, while performance is reported as the geometric-mean speedup.

Graph quality is assessed by comparing the traversal results obtained from the sparsified graph with those of the original graph. We evaluate the following metrics.

Reachability Loss measures the fraction of vertices that become unreachable after sparsification and is computed as $1-R_{\text{sparse}}/R_{\text{orig}}$, where R_{orig} and R_{sparse} denote the numbers of reachable vertices in the original and sparsified graphs, respectively. **Graph Fragmentation** quantifies the structural changes introduced by sparsification using the increase in the number of connected components, computed as $\Delta C = C_{\text{sparse}} - C_{\text{orig}}$. **Distance Stretch** is $d_{\text{sparse}}/d_{\text{orig}}$, where d_{orig} and d_{sparse} denote the distances from the selected source in the original and sparsified graphs, respectively. The metric is averaged over vertices reachable in both graphs; newly unreachable vertices are excluded and captured by Reachability Loss. BFS uses hop distance, whereas SSSP uses weighted distance. Values near one indicate better distance preservation.

D. Performance–Quality Evaluation

This subsection evaluates the impact of graph sparsification from both graph quality and parallel performance perspectives. We first analyze the structural changes introduced by edge removal, followed by their effects on parallel BFS and Delta-Stepping SSSP.

1) Structural Preservation: Since both BFS and SSSP operate on the same sparsified graph, we first evaluate the preservation of graph connectivity before analyzing individual traversal kernels. Reachability loss measures the fraction of vertices that become unreachable after sparsification, while graph fragmentation measures the increase in the number of connected components. Figure 3 summarizes both metrics, averaged across completed datasets. Common-Neighbor consistently provides the strongest structural preservation with below 6% mean reachability loss at the highest requested ratio. Its increase in connected components is also significantly smaller than that of the other methods. In contrast, Random sparsification exhibits the largest degradation in graph connectivity, reaching approximately 25% reachability loss together with the highest graph fragmentation. Degree-Protected, Bridge-Aware, and Weight-Aware preserve connectivity better than Random.

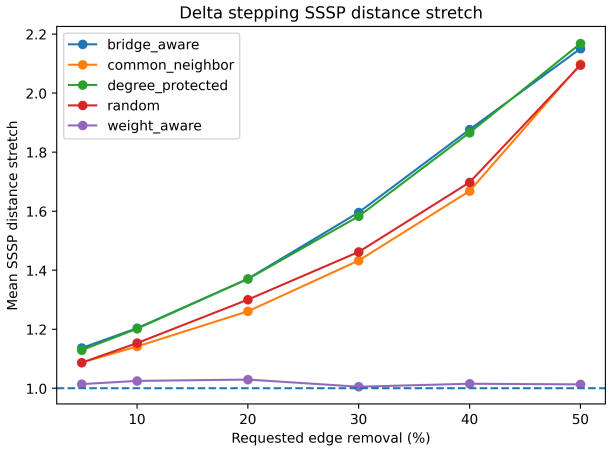
Although the average statistics reveal the overall trend, they conceal important differences among graph families. Figure 4 shows strong dataset dependence. Social and information networks generally retain high reachability, whereas Road-Asia is highly sensitive to removal. Its sparse topology provides fewer alternative paths, so removing critical edges can disconnect the selected source from large portions of the graph.

2) Parallel BFS Evaluation: We next evaluate the impact of graph sparsification on parallel BFS. Since BFS depends solely on graph topology, traversal quality is measured using hop-distance stretch, while traversal performance is evaluated using execution speedup.

Figure 5 summarizes BFS quality and performance. As the sparsification ratio increases, all methods reduce traversal time by decreasing the number of explored edges. Weight-Aware, Degree-Protected, and Bridge-Aware reach up to $3.30\times$ geometric-mean speedup at 50% requested removal. Common-Neighbor achieves lower speedup partly because its protected set slightly limits actual edge removal.

From a traversal-quality perspective, Random sparsification yields the lowest hop-distance stretch among vertices remaining reachable in both graphs, but also the highest reachability loss and fragmentation. This reflects a survivorship effect: disconnected vertices contribute to Reachability Loss rather than stretch. In contrast, Common-Neighbor provides substantially stronger connectivity preservation with moderate hop-distance distortion, revealing a tradeoff between distance and reachability preservation. Overall, these results highlight the tradeoff between preserving shortest traversal paths and maintaining global graph connectivity for parallel BFS.

3) Parallel SSSP Evaluation: Unlike BFS, weighted shortest-path computation depends on both graph topology and



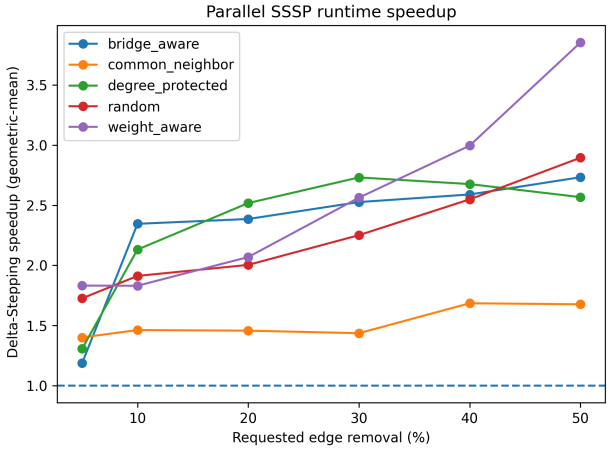


Fig. 6. Performance and quality evaluation of parallel Delta-Stepping SSSP on sparsified graphs. Traversal quality is measured using the average weighted distance stretch, while performance is reported as the geometric-mean speedup across all evaluated datasets. Weight-Aware sparsification consistently provides the best balance between shortest-path preservation and traversal acceleration, particularly at higher sparsification ratios.

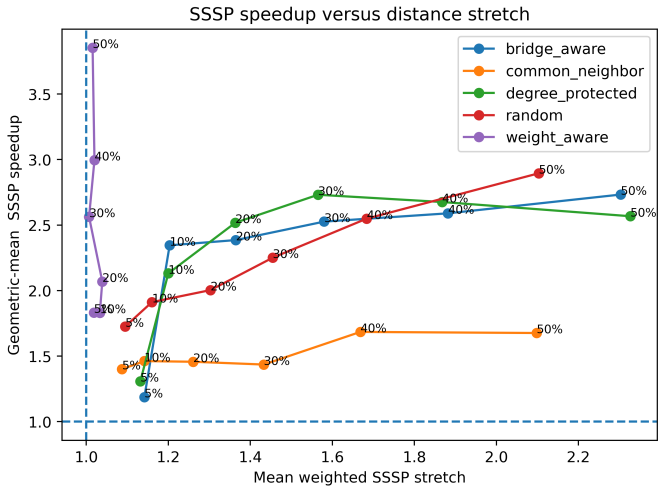


Fig. 7. Performance–quality tradeoff for parallel Delta-Stepping SSSP. Each point corresponds to one sparsification ratio.

edge weights. Consequently, preserving weighted shortest-path distances is as important as reducing graph size.

Figure 6 summarizes the quality and performance of parallel Delta-Stepping SSSP. Weight-Aware consistently achieves the smallest weighted distance stretch, remaining close to the ideal value of one even after removing 50% of the graph edges. Other methods exhibit substantially greater distortion as sparsification increases, highlighting the importance of edge weights for SSSP-oriented sparsification.

The corresponding execution performance demonstrates that graph sparsification substantially accelerates parallel SSSP. Weight-Aware consistently achieves the highest traversal acceleration, reaching up to $3.85\times$ geometric-mean speedup at 50% sparsification while simultaneously providing the best weighted shortest-path preservation. Speedup can exceed the proportional edge reduction because traversal cost also depends on frontier processing, relaxation work, memory behavior, and workload distribution.

The overall performance–quality tradeoff is illustrated in Figure 7. Weight-Aware consistently remains closest to the ideal operating region by combining the lowest weighted distance stretch with the highest traversal speedup. In contrast, Common-Neighbor is more effective in preserving graph connectivity.

V. Conclusions and Future Work

This paper presents a comprehensive performance–quality evaluation of five representative lightweight graph sparsification methods for parallel BFS and Delta-Stepping SSSP on shared-memory multicore systems. Experimental results demonstrated that graph sparsification substantially accelerates graph traversal while exhibiting distinct kernel-aware behavior. Common-Neighbor sparsification provides the strongest preservation of graph connectivity, whereas Weight-Aware sparsification consistently achieves the best weighted shortest-path preservation together with the highest SSSP speedup. BFS results further reveal a tradeoff between distance preservation among reachable vertices and overall reachability. These findings demonstrate that no single sparsification strategy is universally optimal and that the choice of sparsification method should be guided by the target graph traversal kernel.

As future work, adaptive and learning-based graph sparsification techniques can be investigated to automatically select edge-removal strategies according to graph topology and application requirements. The proposed framework can also be extended to distributed-memory systems, GPU-based graph analytics, and additional graph kernels such as PageRank, Connected Components, and Triangle Counting.

Acknowledgment

This material is based upon work supported by the National Science Foundation (NSF) under Award Number 2323533. ChatGPT (GPT-5.6 Sol) was used to generate draft language in Sections I–V, which was reviewed and edited by the authors. The authors take full responsibility for the final content, and all final wording reflects their own edits and judgment.

REFERENCES

[1] Graph500 benchmark. http://www.graph500.org/.

[2] S. Abdelhamid, M. Alam, R. Alo, S. Arifuzzaman, P. Beckman, T. Bhattacharjee, H. Bhuiyan, K. Bisset, S. Eubank, A. C. Esterline, et al. Cinet 2.0: A cyberinfrastructure for network science. In *2014 IEEE 10th International Conference on e-Science*, volume 1, pages 324–331. IEEE, 2014.

[3] R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *nature*, 406(6794):378–382, 2000.

[4] A.-L. Barabási, N. Gulbahce, and J. Loscalzo. Network medicine: a network-based approach to human disease. *Nature reviews genetics*, 12(1):56–68, 2011.

[5] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route planning in transportation networks. In *Algorithm engineering: Selected results and surveys*, pages 19–80. Springer, 2016.

[6] S. Beamer, K. Asanović, and D. Patterson. The gap benchmark suite. *arXiv preprint arXiv:1508.03619*, 2015.

[7] A. A. Benczúr and D. R. Karger. Approximating st minimum cuts in õ (n 2) time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 47–55, 1996.

[8] U. Brandes. A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, 25(2):163–177, 2001.

[9] Y. Chen, H. Ye, S. Vedula, A. Bronstein, R. Dreslinski, T. Mudge, and N. Talati. Demystifying graph sparsification algorithms in graph properties preservation. *Proceedings of the VLDB Endowment (PVLDB)*, 17(3):427–440, 2023.

[10] M. A. M. Faysal, S. Arifuzzaman, C. Chan, M. Bremer, D. Popovici, and J. Shalf. Hypc-map: A hybrid parallel community detection algorithm using information-theoretic approach. In *2021 IEEE High Performance Extreme Computing Conference (HPEC 2021)*, 2021.

[11] R. Hassan and S. Arifuzzaman. An efficient multi-core parallel implementation of sssp algorithm with decreasing delta-stepping. In *2024 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 2024.

[12] K. He, P. Drineas, and R. Khanna. Structure-aware spectral sparsification via uniform edge sampling. *Advances in Neural Information Processing Systems*, 38:49029–49054, 2026.

[13] B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? extended abstract. In *International Symposium on Solving Irregularly Structured Problems in Parallel*, pages 218–225. Springer, 1998.

[14] W. Jin, M. Hashemi, E. Gong, S. Chaterji, J. Deng, X. Cao, and J. Tang. A comprehensive survey on graph reduction: Sparsification, coarsening, and condensation. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2024.

[15] J. Leskovec and R. Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1–20, 2016.

[16] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry. Challenges in parallel graph processing. *Parallel Processing Letters*, 17(01):5–20, 2007.

[17] U. Meyer and P. Sanders. δ-stepping: a parallelizable shortest path algorithm. *Journal of Algorithms*, 49(1):114–152, 2003.

[18] P. Parchas, N. Papailiou, D. Papadias, and F. Bonchi. Uncertain graph sparsification. *IEEE transactions on knowledge and data engineering*, 30(12):2435–2449, 2018.

[19] R. Rossi and N. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

[20] N. S. Sattar and S. Arifuzzaman. Scalable distributed louvain algorithm for community detection in large graphs. *Journal of Supercomputing*, 78, 2022.

[21] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.