Graph Coarsening for Graph Neural Networks

Project Report

Submitted by

[illegible]

Karan Dewangan

12342590

Project Supervisor

Dr. Avijit Pal

Project Co-Supervisor

Dr. Viswesh Jatala

DEPARTMENT OF MATHEMATICS

INDIAN INSTITUTE OF TECHNOLOGY BHILAI

8 MAY 2025

ABSTRACT

Graph Neural Networks (GNN) have become popular for solving complex problems dealing with graph data. Recently, research has gained significant momentum to improve the performance and accuracy of the GNN models. GNN training is computationally expensive, especially when dealing with input graphs that have millions of nodes and billions of edges. To address these challenges, several graph sparsification methods, such as sampling methods and coarsening methods, are proposed in the literature. However, most of them have limitations in several dimensions, such as they do not account for the features of the graph and neighborhood structure.

To address these challenges, we propose a graph coarsening method that takes into account the spectral properties of the graph as well as the node features. The key is to improve the performance of GNN training by reducing the structure of the graph while maintaining essential properties. To achieve this, we first convert the input undirected graph to a weighted graph by assigning weight to an edge based on the neighborhood features and spectral distance. Subsequently, we merge two nodes having a minimum spectral distance into a single node. This helps reduce the overall size of the graph and, hence, improves performance.

We implemented the proposed ideas using Python and evaluated the performance of the GNN models. We observe that GNN training on the coarsened graph achieves significant speedup without negatively affecting the model accuracy when compared to the previous implementations.

Contents

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Most of the real-world data can be represented using the graph data strictures. As a result, for many decades, graph analytics [1–3] has become an active area of research to solve many problems dealing with graph data. Recently, Graph Neural Networks (GNN) [4] have emerged as a machine learning toolbox to solve complex problems on the graph data structure. Subsequently, GNNs have been adopted in various applications [5], such as social networks [6], recommendation systems [7], knowledge graph [8], financial fraud detection [9] and medicine [10] etc.

In GNN, each node/edge in the graph contains initial features. During GNN training, the features are iteratively updated through a message-passing step, which aggregates the features from the neighborhood vertices. This process is repeated until a convergence criteria is achieved. The aggregation operation is computationally expensive as most of the real-world graph datasets often contain billions of edges, far exceeding the limited and expensive memory available on modern GPUs. To solve these issues, many techniques, such as sampling techniques [11–17], graph coarsening [18] were proposed. However, most of them have limitations in several dimensions, such as they do not account for the graph neighborhood structure or node features at the time of the sparsification process.

Graph coarsening has been explored in the context of the graph analytical application; however, research has not matured enough for GNN. In the context of GNN, it is crucial that the coarsened graph retains key structural properties and should not degrade accuracy of the GNN model. Moreover, for large-scale GNN training, the coarsening must be more efficient in reducing the overall training time.

To address these challenges, we propose a novel graph coarsening technique supporting GNN models. The key is to reduce the size of the graph by considering the node features and spectral distance. To achieve this, we first construct a weighted graph from the input undirected graph by giving the weight to each graph based on the node features. Subsequently, we compute the spectral distance between each pair of nodes connected by an edge. Finally, we propose a graph coarsening scheme, which aims to merge the nodes of the graph that have minimal spectral distance to help reduce the size of the graph. We implemented the proposed graph coarsening scheme using Python. We evaluated the effectiveness of our approach by training

GNN models. We observe that our graph coarsening is effective in significantly reducing GNN training time compared to full graph training.

To summarize, this project makes the following contributions.

- We propose a novel graph coarsening algorithm to improve the performance of GNN training. The key is to reduce the size of the graph by considering the spectral properties and node features.
- We implemented the proposed ideas in Python and evaluated the performance of GNN training on large graphs.
- We observe that our graph coarsening achieves a significant speed up on the GNN training without negatively impacting the accuracy when compared to the GNN training on the full graph.

We organize the rest of the report as follows. Chapter 2 discusses the background on the graph neural networks and graph coarsening. Chapter 4 presents the details of our approach. Chapter 3 describes the related studies of the project. Chapter 5 presents the experimental evaluation of the proposed approaches. Finally, we conclude the report in Chapter 6.

Chapter 2

Background

This section gives a brief background on the graph neural networks and graph coarsening.

2.1 Graph Neural Networks

Consider a graph G(V,E) consists of a set of V vertices and E edges. In GNN, each node v in the graph has initial features, which is denoted as a vector h_v^0. For a k layer GNN, the features of v at layer i (denoted as $\mathbf{h}_v^{(i)}$) is computed by aggregating the features from its neighborhood vertices, and updating it through a non linear function (σ_i) as shown in Equations 2.1 and 2.2.

$$\mathbf{h}_{N(v)}^i = AGG_i(\mathbf{h}_u^{i-1}, \forall u \in N(v)) \tag{2.1}$$

$$\mathbf{h}_v^i = \sigma_i(\mathbf{W}^i.CONCAT(\mathbf{h}_{N(v)}^i, \mathbf{h}_v^{i-1})) \tag{2.2}$$

where $N(v)$ denotes the set of neighboring nodes of v. The AGG_i function denotes the aggregation function, such as min, max, or mean. $\mathbf{W_i}$ denotes the weight matrix at layer i.

A GNN model is trained for a specified number of epochs. The final features obtained after training are used for the down-streaming tasks, such as node classification.

2.2 Graph Coarsening

Consider a graph $G = (V, E, A)$, where V denotes a set of nodes such that $|V| = N$, E denotes the a set of edges such that $|E| = M$, and $A \in \mathbb{R}^{N \times N}$ denotes the adjacency matrix corresponding to G. We denote the coarsened graph using $G' = (V', E', A')$ where V' denotes a set of nodes, E' denotes the set of the edges in the coarsened graph G'. The adjacency matrix corresponding to G' using denoted using $A' \in \mathbb{R}^{N' \times N'}$, where $N' = |V'|$ and $M' = |E'|$.

We define the graph coarsening with a respect to the set of partitions $\mathscr{P} = \{C_1, C_2, \ldots, C_{N'}\} \subset V$

where $N' = |V'|$. Where each partition C_i corresponding to a node in the coarsened graph. We define a partition matrix $P \in \{0,1\}^{N \times N'}$, where $P_{ij} = 1$ if $v_i \in C_j$. Now for a given set of partitions P, then adjacency matrix A' corresponding to the coarsened graph G' is computed using $A' = P^T A P$.

Chapter 3

Related Work

This section briefly describes the related studies on graph neural networks and graph reduction techniques, such as graph sampling, graph coarsening, and graph condensation.

3.1 Graph Neural Networks

With the emergence of the GNN models, several techniques were proposed to improve their accuracy. The models differ in the way the aggregation is performed. For instance, GAT [12] gives specific attention to its neighbors, whereas GCN [4] uses convolution operation for aggregation. GraphSAGE [11] samples the neighbors and aggregates with their own features. A survey of techniques to improve the performance and efficiency of GNN models are explored [5].

Recently, many studies have focused on improving the performance of GNN by exploiting the recent trends in the architectures, such as GPU platforms [19–22]. Similarly, GNN was also adopted to distributed platforms [23–25].

3.2 Graph Reduction Techniques

Graph reduction techniques aim to improve performance by reducing the size of the graph without degrading the correctness of the application.

In the context of GNN, several graph sampling techniques [21, 25, 26] are proposed, which sample the neighbors of the node to reduce the computation overhead of the aggregation operation and model training. These techniques also help in reducing the memory consumption of the model. For instance, Zhang et al. [26] samples the neighbors based on the similarity of the node features. GraphSAGE [11] uses random sampling methods to randomly sample the neighbors. ClusterGCN [27] aims to restrict the neighborhood using the induced subgraph constructed from the batch nodes.

Graph Coarsening is another alternative method to reduce the size of a graph while maintaining its essential properties. Graph coarsening was majorly studied in the context of the graph analytical applications [28–30]. Most of these techniques aim to coarsen graphs while maintaining their spectral characteristics, as captured by matrix representations [31] [32] [28].

Recently, graph coarsening has been explored in the context of GNN training as well. For instance, Graph Skeleton [33] proposes alpha, beta, and gamma strategies to reduce the size of the graph by retaining the structural properties and showing their effectiveness on GNN training. Convolution Matching [18] technique uses the node features to merge the nodes in the graph. Subsequently, they train the model on the coarsened graph to improve the performance. Coreset techniques focus on identifying a representative subset of the training data such that models trained on these subsets achieve performance comparable to those trained on the full dataset [34] [35] [36].

Machine learning techniques are explored even to reduce the size of the graph. For instance, graph condensation [37] proposes a machine-learning model to shrink a graph's size while ensuring the accuracy of the model. This is useful for faster training and inference in tasks like node classification or link prediction. Distillation and condensation [38] [39] [40] [41] [42] aim to create a small synthetic dataset. Models trained on this smaller set should perform similarly to those trained on the large dataset.

Chapter 4

Approach

The aim is to improve the performance of GNN training by reducing the size of graph. To achieve this, we transform G into a G_c, such that $G_c < G$ and the spectral distance between G and G_c is minimized. This ensures that the G_c have similar spectral properties as that of original graph G and training on the G_c requires less computation time that G.

For this, we first convert the given unweighted graph G to a weighted graph G', where the weight of edge is computed based on the nodes features of the end points of the edge. Section 4.1 provides the details of the weight computation for each edge. Subsequently, we transform the graph G' to another weighted graph G'' such that the weight of each edge computed as the normalized weights of the end points of the edge. Finally, we coarsen the graph G'' to G_c such that $G_c < G''$ and G_c consume less training time than G. We present these details in Section 4.2 and Section 4.3.

4.1 Weight Graph Construction

For a given graph $G(V,E)$, the weight of edge (u,v) is computed based on the feature matrix X. Let H denotes the feature matrix obtained after applying layer-1 computation in the GNN training, then H can be computed using Equation 4.1.

$$H = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}XI \tag{4.1}$$

Here D is the degree diagonal matrix and A denotes the adjacency matrix corresponding to G, X denotes the initial feature matrix, and I is the identity matrix.

The weight of the edge (u,v) is calculated as the difference in the feature vectors before and after merging the nodes the nodes u and v. For each edge (u,v) in the original graph create a merge graph $G' = (V',E',A')$ by merging the nodes u and v. Subsequently, we compute the weight of (u,v) as follows.

$$\text{weight}(u,v) \equiv \left\| P_{(u,v)} H'_{P_{(u,v)}} - H \right\| \tag{4.2}$$

$$\text{weight}(u,v) \equiv ||P_{(u,v)}(D'^{-\frac{1}{2}}A'D'^{-\frac{1}{2}})X' - (D'^{-\frac{1}{2}}A'D'^{-\frac{1}{2}})X|| \tag{4.3}$$

Where the partition matrix $P(u,v)$ defines this merging operation of u and v. $H'_P(u,v)$ represents the result of a coarse graph convolution performed on the corresponding coarse graph. This weight function evaluates the effect of merging the two nodes.

Since calculating the total weight of all possible node combinations becomes computationally impractical, as the number of combinations increases exponentially with the number of nodes. Therefore, we consider only the weights of the edges.

4.2 Spectral Distance in Graph Coarsening

Once we compute the weighted graph, we aim to convert the original graph G to G_c such that are G and G_c are spectrally similar [30]. Jin et al. [30] have shown that the graph G_{uv} be a coarse version of the graph G, obtained by merging a pair of nodes v and u and if the edge weights of the merged nodes satisfy certain conditions, then

$$\left|\left|\frac{w(v)}{d(u)} - \frac{w(v)}{d(u)}\right|\right| \leq \varepsilon, \tag{4.4}$$

then the spectral distance between coarse and the original graphs are bounded by

$$SD_{\text{full}}(G, G_{uv}) \leq N\varepsilon \quad \text{and} \quad SD_{\text{part}}(G, G_{uv}) \leq N\varepsilon. \tag{4.5}$$

The proposition states that the spectral distance is bounded by the discrepancy in edge weights of the merged nodes. This means that the difference in edge weights affects how much the spectral properties are preserved. Therefore, minimizing the edge weights among nodes within the same partition leads to better preservation of spectral properties.

4.3 Graph Coarsening for GNN

Section 4.2 shows that if two nodes in the graph satisfy Equations 4.4 and 4.5, then it is beneficial to merge them. So we aim to transform G' to another graph G'' where the weight of edge u,v is obtained using Equation 4.4.

$$\left|\left|\frac{w(u)}{d(u)} - \frac{w(v)}{d(v)}\right|\right|, \tag{4.6}$$

Finally, we coarsen G'' to G_c by applying least edge matching using METIS [43], where merges

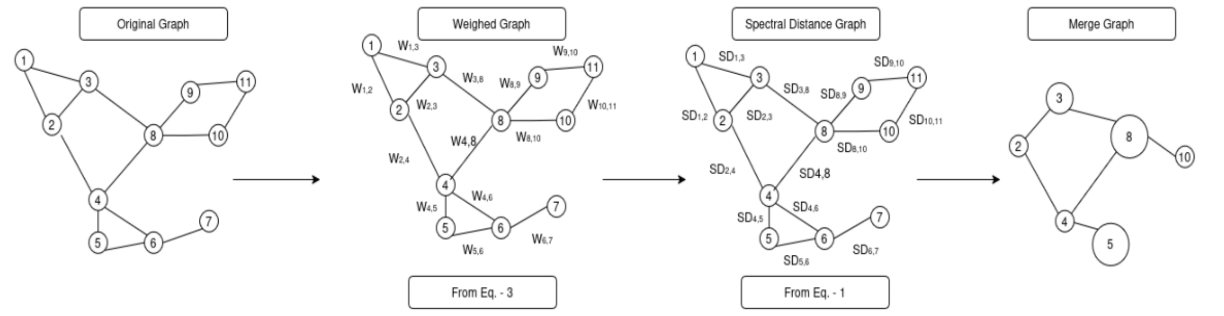

Figure 4.1: Graph Coarsening for Graph Neural Networks

u and *v* have lower weight are merged into a single entity called a supernode. When two nodes *u* and *v* are merge into a single supernode, this new supernode inherits all the neighbors of both *u* and *v*. In the coarsened graph, each edge that connects to the supernode is assigned a weight that represents the average number of edges that previously connected the individual nodes *u* and *v* to the corresponding neighbor.

Figure 4.1 shows the example of the graph coarsening for GNN. In the first step, the original graph is converted to weighted graph G' based on the weight computation in Section 4.1. Subsequently, the graph is G' to G'' by using the spectral distance as shown in Equation 4.6. Finally, we obtained the merged graph G_c by merging the nodes in G''

Once the graph is coarsened, we train the GNN model the final graph G_c and apply the inference on the original G to show the effectiveness of application. Since the graph G_c is much smaller than G, the training G_c can help in improving the performance without losing the accuracy.

Chapter 5

Experimental Results

5.1 Experimental Setup

We conducted our experiments on a machine having Intel(R) Xeon(R) Gold 5218 CPU with 16 cores, clocked at 2.30 GHz, and 512 GB RAM. The machine has a single NVIDIA RTX A6000 GPU with 48 GB of memory. We compile the graph coarsening method with Python. We used DGL v2.0.0 and PyTorch v1.13.0 for training the GNN application.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	**Feature Dim**	**No. of Classes**	**Train/Val/Test**
Cora	2708	10556	1433	7	140\500\1000
Citeseer	3327	9104	3703	6	120\500\1000
Actor	7600	53318	932	5	3648\2432\1520
Chameleon	2277	36101	2325	5	1092\729\456

Table 5.1: Datasets and their key properties

We performed experiments on a wide range of graph datasets, whose properties are shown in Table 5.1. The Cora dataset consists of a set of scientific publications classified into one of seven classes. The Citeseer datasets also used for the classification of paper into 6 classes. The Actor dataset consists of a graph where nodes represent actors, and edges indicate co-occurrence on the same Wikipedia page. Each node is associated with a feature vector based on keywords extracted from the corresponding Wikipedia page. The Chameleon dataset is a graph where nodes represent english wikipedia articles, and edges denote mutual hyperlinks between them. Each node is characterized by features indicating the presence of specific nouns in the article text. The nodes are categorized into five classes based on their average monthly web traffic.

We compared the performance of graph coarsening by training the GraphSAGE [11] model using GCN aggregation from the DGL framework [23]. The GraphSAGE model uses random sampling method to improve the performance of GNN training. We train GNN with three layers, a learning rate of 0.001, and 50 epochs. We present the results for various batch sizes ranging from 1024.

Dataset	**$\|V\|$(Org)**	**$\|V\|$(Coarse)**	**$\|E\|$(Org)**	**$\|E\|$(Coarse)**	**Time(S)**
Cora	2708	1722	10556	5150	0.0087
Citeseer	3327	2080	9104	4240	0.0089
Actor	7600	5282	53318	16408	0.0092
Chameleon	2277	1207	36101	14162	0.0087

Table 5.2: Evaluation of Graph Datasets After Coarsening

Dataset	**Memory(MB)**	**Memory Coarse(MB)**
Cora	29.69	18.87
Citeseer	94.06	58.80
Actor	83.50	57.87
Chameleon	64.60	34.21

Table 5.3: Memory-Efficient Graph Processing through Coarsening Techniques

5.2 Results

The Table 5.2 presents the results of applying a coarsening algorithm to four well-known graph datasets: Cora, Citeseer, Actor, and Chameleon. These datasets are commonly used benchmarks in the field of node classification and graph learning. For each dataset, the number of original nodes, coarsened nodes, original edges, and coarsened edges are reported, along with the time taken for coarsening. The results indicate a substantial reduction in graph size. For example, in the Cora dataset, the number of nodes decreased from 2708 to 1722, and the number of edges fell from 10556 to just 5150. This represents an over 25% reduction in graph size, achieved in just 0.0053 seconds. Similar trends are observed in the other datasets. Citeseer saw a reduction from 3327 to 2080 nodes and from 9104 to 4240 edges. Actor, a much larger graph, experienced a drop from 7600 to 5282 nodes and from 53318 to 16408 edges. Chameleon was reduced from 2277 to 1207 nodes and from 36101 to 14162 edges.

Furthermore, the coarsening process is very efficient, with times ranging from 0.0087 to 0.0092 seconds, even for graphs with tens of thousands of edges. This efficiency makes it feasible to integrate coarsening as a preprocessing step in large-scale machine learning pipelines.

In conclusion, the graph coarsening technique demonstrates strong potential in improving the scalability of graph learning methods. By reducing the number of nodes and edges significantly with minimal processing time, it enables faster computation, lower memory usage, and maintains sufficient structural fidelity to support downstream tasks. As graph-based models continue to grow in popularity, especially with the rise of GNNs, coarsening will likely play a vital role in optimizing performance on large and complex datasets.

The Table 5.3 comparison reveals the significant memory-saving advantages of applying graph coarsening techniques to standard graph benchmarks. The memory consumption before and after coarsening is recorded in megabytes (MB) for four datasets: Cora, Citeseer, Actor, and Chameleon. Graph coarsening consistently reduces the memory footprint across all datasets. For instance, Cora's memory usage drops from 29.69 MB to 18.87 MB, a reduction of approximately 36.5%. Similarly, Citeseer shows a reduction from 94.06 MB to 58.80 MB, saving more than 35 MB of memory, which is particularly valuable for large-scale or resource-constrained environments. In Table 5.3 , the Actor dataset benefits from a decrease from 83.50 MB to 57.87 MB, and Chameleon sees a reduction from 64.60 MB to 34.21 MB, saving almost 47% of the original memory usage. These reductions make graph coarsening especially appealing for training graph neural networks (GNNs) on edge devices or when deploying models in memory-sensitive applications.

In summary, graph coarsening significantly lowers memory requirements, enabling more scalable and efficient graph learning workflows, particularly beneficial for handling large datasets and complex models.

Dataset	**Training time org(sec)**	**Accuracy org**	**Training time coarse(sec)**	**Accuracy Coarse**
Cora	3.67	0.805	3.53	0.685
Citeseer	6.68	0.686	6.43	0.724

Table 5.4: GNN Training

From Table 5.3 we can observe that graph coarsening has a mixed impact on performance. In the case of the Cora dataset, the coarsened graph slightly reduces the training time from 3.67 to 3.53 seconds, showing a modest improvement in efficiency. However, this comes at the cost of reduced classification accuracy, dropping from 80.5% to 68.5%. This suggests that for Cora, the coarsening process may have removed important structural or feature-based information necessary for effective node classification.

In contrast, the Citeseer dataset shows a more favorable outcome. The training time decreases from 6.68 to 6.43 seconds, and the accuracy improves from 68.6% to 72.4%. This indicates that, for Citeseer, coarsening not only speeds up training but may also enhance generalization by reducing overfitting or eliminating noisy connections in the original graph.

Chapter 6

Conclusion

This project aims to improve the performance of GNN training by graph coarsening. The key is to reduce the graph size by preserving essential properties. Our graph coarsening scheme takes into account both the spectral properties of the graph and the features of the nodes. First, we convert the input undirected graph into a weighted graph, where the edge weights are assigned based on the spectral properties and node features. Subsequently, we merge two nodes that have minimal spectral distance.

The proposed methodology was implemented in Python and evaluated it on several GNN models. We observe that our approach reduces the GNN training time significantly without negatively affecting the accuracy of the models.

In the future, we plan to extend our work to other GNN models and also show its effectiveness for distributed platforms.

Bibliography

[1] J. Shun and G. E. Blelloch, "Ligra: a lightweight graph processing framework for shared memory," in *Proceedings ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, ser. PPoPP '13, 2013, pp. 135–146. [Online]. Available: http://doi.acm.org/10.1145/2442516.2442530

[2] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings ACM SIGMOD Intl Conf. on Management of Data*, ser. SIGMOD '10, 2010, pp. 135–146. [Online]. Available: http://doi.acm.org/10.1145/1807167.1807184

[3] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed Graph-parallel Computation on Natural Graphs," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 17–30. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387880.2387883

[4] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017. [Online]. Available: https://arxiv.org/abs/1609.02907

[5] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[6] W. Wu, B. Li, C. Luo, and W. Nejdl, "Hashing-accelerated graph neural networks for link prediction," in *Proceedings of the Web Conference 2021*, ser. WWW '21. ACM, Apr. 2021, pp. 2910–2920. [Online]. Available: http://dx.doi.org/10.1145/3442381.3449884

[7] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, ser. KDD '18. ACM, Jul. 2018, pp. 974–983. [Online]. Available: http://dx.doi.org/10.1145/3219819.3219890

[8] N. Park, A. Kan, X. L. Dong, T. Zhao, and C. Faloutsos, "Estimating node importance in knowledge graphs using graph neural networks," in *Proceedings of the 25th ACM*

SIGKDD International Conference on Knowledge Discovery Data Mining, ser. KDD '19. ACM, Jul. 2019. [Online]. Available: http://dx.doi.org/10.1145/3292500.3330855

[9] Z. Liu, C. Chen, X. Yang, J. Zhou, X. Li, and L. Song, "Heterogeneous graph neural networks for malicious account detection," 2020. [Online]. Available: https://arxiv.org/abs/2002.12307

[10] S. Rhee, S. Seo, and S. Kim, "Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification," 2018. [Online]. Available: https://arxiv.org/abs/1711.05859

[11] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 1025–1035.

[12] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018. [Online]. Available: https://arxiv.org/abs/1710.10903

[13] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," 2018. [Online]. Available: https://arxiv.org/abs/1801.10247

[14] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," 2018. [Online]. Available: https://arxiv.org/abs/1710.10568

[15] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," 2019.

[16] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," 2020. [Online]. Available: https://arxiv.org/abs/1907.04931

[17] P. Gong, R. Liu, Z. Mao, Z. Cai, X. Yan, C. Li, M. Wang, and Z. Li, "gsampler: General and efficient gpu-based graph sampling for graph learning," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 562–578.

[18] C. Dickens, E. Huang, A. Reganti, J. Zhu, K. Subbian, and D. Koutra, "Graph coarsening via convolution matching for scalable graph neural network training," in *Companion Proceedings of the ACM Web Conference 2024*, ser. WWW '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1502–1510. [Online]. Available: https://doi.org/10.1145/3589335.3651920

[19] L. Ma, Z. Yang, Y. Miao, J. Xue, M. Wu, L. Zhou, and Y. Dai, “Neugraph: Parallel deep neural network computation on large graphs,” 2019.

[20] Y. Wang, B. Feng, G. Li, S. Li, L. Deng, Y. Xie, and Y. Ding, “Gnnadvisor: An adaptive and efficient runtime system for GNN acceleration on gpus,” in *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*, A. D. Brown and J. R. Lorch, Eds. USENIX Association, 2021, pp. 515–531. [Online]. Available: https://www.usenix.org/conference/osdi21/presentation/wang-yuke

[21] W. Wu, X. Shi, L. He, and H. Jin, “Turbognn: Improving the end-to-end performance for sampling-based gnn training on gpus,” *IEEE Transactions on Computers*, vol. 72, no. 9, pp. 2571–2584, 2023.

[22] Q. Fu, Y. Ji, and H. H. Huang, “Tlpgnn: A lightweight two-level parallelism paradigm for graph neural network computation on gpu,” in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC ’22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 122–134.

[23] “DGL,” https://www.dgl.ai/, 2019. [Online]. Available: https://www.dgl.ai/

[24] M. L. Das, V. Jatala, and G. R. Gupta, “Joint partitioning and sampling algorithm for scaling graph neural network,” in *2022 IEEE 29th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2022, pp. 42–47.

[25] D. Deshmukh, G. R. Gupta, M. Chawla, V. Jatala, and A. Haldar, “Entropy aware training for fast and accurate distributed gnn,” in *2023 IEEE International Conference on Data Mining (ICDM)*, 2023, pp. 986–991.

[26] X. Zhang, Y. Shen, and L. Chen, “Feature-oriented sampling for fast and scalable gnn training,” in *IEEE International Conference on Data Mining*, 2022.

[27] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, “Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, ser. KDD ’19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 257–266. [Online]. Available: https://doi.org/10.1145/3292500.3330925

[28] G. Bravo-Hermsdorff and L. M. Gunderson, “A unifying framework for spectrum-preserving graph sparsification and coarsening,” 2020. [Online]. Available: https://arxiv.org/abs/1902.09702

[29] A. Loukas, “Graph reduction with spectral and cut guarantees,” 2018. [Online]. Available: https://arxiv.org/abs/1808.10650

[30] Y. Jin, A. Loukas, and J. F. JaJa, "Graph coarsening with preserved spectral properties," 2019. [Online]. Available: https://arxiv.org/abs/1802.04447

[31] A. Loukas and P. Vandergheynst, "Spectrally approximating large graphs with smaller graphs," 2018. [Online]. Available: https://arxiv.org/abs/1802.07510

[32] D. Durfee, Y. Gao, G. Goranci, and R. Peng, "Fully dynamic spectral vertex sparsifiers and applications," 2019. [Online]. Available: https://arxiv.org/abs/1906.10530

[33] L. Cao, H. Deng, Y. Yang, C. Wang, and L. Chen, "Graph-skeleton: 1% nodes are sufficient to represent billion-scale graph," in *Proceedings of the ACM Web Conference 2024*, ser. WWW '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 570–581. [Online]. Available: https://doi.org/10.1145/3589334.3645452

[34] S. Har-Peled and S. Mazumdar, "Coresets for *k*-means and *k*-median clustering and their applications," 2018. [Online]. Available: https://arxiv.org/abs/1810.12826

[35] M. Welling, "Herding dynamic weights for partially observed random field models," 2012. [Online]. Available: https://arxiv.org/abs/1205.2605

[36] O. Sener and S. Savarese, "Active learning for convolutional neural networks: A core-set approach," 2018. [Online]. Available: https://arxiv.org/abs/1708.00489

[37] W. Jin, L. Zhao, S. Zhang, Y. Liu, J. Tang, and N. Shah, "Graph condensation for graph neural networks," 2022. [Online]. Available: https://arxiv.org/abs/2110.07580

[38] B. Zhao, K. R. Mopuri, and H. Bilen, "Dataset condensation with gradient matching," 2021. [Online]. Available: https://arxiv.org/abs/2006.05929

[39] W. Jin, X. Tang, H. Jiang, Z. Li, D. Zhang, J. Tang, and B. Yin, "Condensing graphs via one-step gradient matching," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '22. ACM, Aug. 2022, p. 720–730. [Online]. Available: http://dx.doi.org/10.1145/3534678.3539429

[40] W. Jin, L. Zhao, S. Zhang, Y. Liu, J. Tang, and N. Shah, "Graph condensation for graph neural networks," 2022. [Online]. Available: https://arxiv.org/abs/2110.07580

[41] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros, "Dataset distillation," 2020. [Online]. Available: https://arxiv.org/abs/1811.10959

[42] O. Bohdal, Y. Yang, and T. Hospedales, "Flexible dataset distillation: Learn labels instead of images," 2020. [Online]. Available: https://arxiv.org/abs/2006.08572

[43] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, 1998.