DSpar: An Embarrassingly Simple Strategy for Efficient GNN Training and Inference via Degree-Based Sparsification

Zirui Liu\textsuperscript{1}, Kaixiong Zhou\textsuperscript{1}, Zhimeng Jiang\textsuperscript{2}, Li Li\textsuperscript{3}, Rui Chen\textsuperscript{3}, Soo-Hyun Choi\textsuperscript{3}, Xia Hu\textsuperscript{1}

\{Zirui.Liu, Kaixiong.Zhou, Xia.Hu\}@rice.edu, \{zhimengj\}@tamu.edu, \{li.li, rui.chen1, sh9.choi\}@samsung.com

\textsuperscript{1}Rice University, \textsuperscript{2}Texas A\&M University, \textsuperscript{3}Samsung Electronics America

Reviewed on OpenReview: \url{https://openreview.net/forum?id=SaVEXFuozg}

Abstract

Running graph neural networks (GNNs) on large graphs suffers from notoriously inefficiency. This is attributed to the sparse graph-based operations, which is hard to be accelerated by community hardware, e.g., GPUs and CPUs. One potential solution is to “sketch” the original graph by removing unimportant edges, then both the training and inference processes are executed on the sparsified graph with improved efficiency. Traditional graph sparsification work calculates the edge importance score, i.e., effective resistance, from graph topology with theoretical guarantee. However, estimating effective resistance is even more expensive than training GNNs itself. Later, learning-based sparsification methods propose to learn the edge importance from data, but with significant overhead due to the extra learning process. Thus, both of them introduce significant ahead-of-training overhead. In this paper, we experimentally and theoretically prove that effective resistance can be approximated using only the node degree information and achieve similar node presentations on graphs with/without sparsification. Based on this finding, we propose DSpar to sparsify the graph once before training based on only the node degree information with negligible ahead-of-training overhead. In practice, in the training phase, DSpar is $1.1 \sim 5.9 \times$ faster than the baselines on different datasets and models with almost no accuracy drop. For the inference phase, DSpar reduces $30\% \sim 90\%$ latency. The code is available at \url{https://github.com/warai-0toko/DSpar_tmlr}.

1 Introduction

Graph neural networks (GNNs) have achieved great success in representation learning on graphs from various domains, including social networks [Hamilton et al., 2017; Liu et al., 2023a; Jiang et al., 2022], biology [Hu et al., 2020], and recommendation system [Ying et al., 2018; Liu et al., 2023b; Zha et al., 2023; Wang et al., 2020]. Despite their effectiveness, GNNs are notoriously known for being inefficient. Specifically, GNNs are characterized by their alternating sequence of aggregation and update phases. During the aggregation phase, each node gathers information from its neighboring nodes in a layer-by-layer manner, using sparse matrix-based operations as described in [Fey & Lenssen, 2019; Wang et al., 2019]. Then, in the update phase, each node updates its representation based on the aggregated messages via dense matrix-based operations [Fey & Lenssen, 2019; Wang et al., 2019]. In Figure \ref{fig:1}, \texttt{SpMM} and \texttt{MatMul} are the sparse and dense operations in the aggregation and update phases, respectively. Our profiling results indicate that the aggregation phase may take up to 90\% of the total training time, and a similar pattern can be observed in the inference process.

The execution time of sparse operation is proportional to the number of edges in the graph. So intuitively, one straightforward way to accelerate the process is to sparsify the graph by removing unimportant edges. Thus, previous work tries to produce a “sketch” of the input graph by removing unimportant edges once before training, then using the sparsified graph for both the training and inference process. These works can be
roughly divided into two streams of works. **The first stream of work** proposes to remove edges according to their **theoretical importance score**, which is calculated based on the graph topology [Spielman & Srivastava, 2011]. Albeit the good theoretical properties, obtaining the theoretical importance score (i.e., effective resistance) is even more expensive than training GNNs itself. **The Second stream of work** tries to learn the edge importance from data [Zheng et al., 2020; Chen et al., 2021; Li et al., 2020]. However, learning to drop edges introduces another training process for identifying redundant edges, which introduces significant ahead-of-training overhead. Thus up to our knowledge, no prior discussion was placed on how to accurately sparsify the graph with little or no time overhead for fast GNN training and/or inference. In view of such, this paper raises the question:

**Can we sparsify the graph once before training with little overhead, while achieving similar model performance with less training and inference time?**

This paper makes an attempt in providing a positive answer to the above question. We experimentally and theoretically prove that the traditional edge importance score, i.e., effective resistance, can be efficiently approximated by using only the node degree information. Based on our theoretical analysis, we propose Degree-based Sparsification (DSpar). Specifically, We first down-sampling edges based on only the node degree before training. Then we use the sparsified graph for both training and inference. As a result, the training and inference processes are both accelerated since computations are executed on a sparsified graph. We theoretically show that the node embeddings learned on the graph sparsified by DSpar are good approximations of those learned on the original graphs. Our main contributions are outlined below:

- We design DSpar, a highly-efficient algorithm to sub-sample the edges based on only the degree information before training. We theoretically prove that GNNs could learn expressive node representations on graphs sparsified by DSpar.

- DSpar sparsifies the graph once before training, and the sparsified graph can be used for both training and inference with improved efficiency. DSpar allows to run GNNs faster in wall-clock time. For the training phase, DSpar achieves $1.1 \sim 5.9 \times$ faster than baseline with almost no accuracy drop. For the inference phase, DSpar reduces $30\% \sim 90\%$ latency compared to the baseline.

- We implement DSpar as a ready-to-use extension for Pytorch Geometric and Pytorch, which supports parallel sampling of edges from large graphs (Appendix A).

## 2 Preliminary Analysis

Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph with $\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}$ and $\mathcal{E} = \{e_1, \ldots, e_{|\mathcal{E}|}\}$ being the set of nodes and edges, respectively. Let $X \in \mathbb{R}^{|\mathcal{V}| \times d}$ be the node feature matrix. $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ is the graph adjacency matrix, where $A_{i,j} = 1$ if $(v_i, v_j) \in \mathcal{E}$, else $A_{i,j} = 0$. $\tilde{A} = \tilde{D}^{-\frac{1}{2}}(A + I)\tilde{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix, where $\tilde{D}$ is the degree matrix of $A + I$. GNNs recursively update the embedding of a node by aggregating embeddings of its neighbors. For example, the forward pass of the $l$th Graph Convolutional Network (GCN) layer [Kipf & Welling, 2017] can be defined as:

$$H^{(l+1)} = \text{ReLU}(\tilde{A}H^{(l)}\Theta^{(l)}),$$

where $H^{(i)} \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the node embedding matrix consisting of node embedding $h_v^{(i)}$ for all $v \in \mathcal{V}$ at the $l$th layer and $H^{(0)} = X$. $\Theta^{(l)}$ is the weight matrix of the $l$th GCN layer. In practice, $\tilde{A}$ is often stored in the sparse matrix format, e.g., compressed sparse row (CSR) [Fey & Lenssen, 2019]. From the implementation aspect, the computation of Equation (1) can be described as:

![Figure 1: The time profiling of a two-layer GCNs on different datasets. SpMM in the aggregation phase may take 70% ~ 90% of the total time.](image)
$H^{(l+1)} = \text{ReLU} \left( \text{SpMM} \left( \tilde{A} \cdot \text{MatMul} \left( H^{(l)}, \Theta^{(l)} \right) \right) \right)$.

where $\text{SpMM}(\cdot, \cdot)$ is Sparse-Dense Matrix Multiplication and $\text{MatMul}(\cdot, \cdot)$ is the Dense Matrix Multiplication. Unlike normal dense matrix, the elements is randomly distributed in the sparse matrix. Thus sparse operations, such as $\text{SpMM}$, have many random memory accesses and are much slower than the dense counter-part [Han et al., 2016]. To get a sense of the scale, we show in Figure 1 that for GCNs, $\text{SpMM}$ may take roughly $70\% \sim 90\%$ of the total training time.

3 Fast Graph Sparsification

As we analyzed, the sparse operations are the main efficiency bottleneck for running GNNs. For sparse operations, the computation is only executed on non-zero entries. To improve the efficiency, we propose to reduce the number of non-zero entries in the adjacency matrix by removing unimportant edges.

3.1 Sampling-base graph sparsification

To improve the training efficiency while minimizing the impact of compression, we sparsify the graphs by removing unimportant edges. We sparsify the graph once before training. Thus, the training and inference processes are both accelerated since the computation is done on sparsified graphs. Below we introduce how to sparsify the graph in an unbiased and efficient way.

**Algorithm 1:** Sampling-based Graph Sparsification [Spielman & Srivastava, 2011]

```
Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, sampling probability $\{p_e\}_{e \in \mathcal{E}}$, number of samples to draw $Q$.
Output: the sparsified weighted graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$ with edge weights $\{w_e\}_{e \in \mathcal{E}'}$

1. $\mathcal{E}' \leftarrow \emptyset$
2. for $j = 1, \cdots, Q$ do
3.   Sample an edge $e \sim \mathcal{E}$ with replacement according to $p_e$
4.   if $e \notin \mathcal{E}'$ then
5.     Add $e$ to $\mathcal{E}'$ with weight $w_e = \frac{A_{e}}{Qp_e}$
6.   end
7.   else
8.     $w_e \leftarrow w_e + \frac{A_{e}}{Qp_e}$
9.   end
10. end
11. return $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$ with edge weights $\{w_e\}_{e \in \mathcal{E}'}$
```

The process of graph sparsification is shown in Algorithm 1. First, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, for each edge $e = (v_i, v_j) \in \mathcal{E}$, we need to decide the probability $p_e$ that it will be sampled ($\sum_{e \in \mathcal{E}} p_e = 1$). Then we need to decide the number of samples to draw $Q$, which implicitly controls the sparsity of the sparsified graph $\mathcal{G}'$. The sparsified adjacency matrix $A' \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ of $\mathcal{G}'$ can be constructed as $A'_{i,j} = w_e$ (line 5 and line 8) if $e = (v_i, v_j) \in \mathcal{E}'$ else $A'_{i,j} = 0$. We note that $A'$ is an unbiased estimation of $A$, i.e., $\mathbb{E}[A'] = A$. To see this,

$$\mathbb{E}[A']_{i,j} = \mathbb{E} \left( \sum_{k=1}^{Q} \mathbb{1}_e \frac{A_{i,j}}{Qp_e} \right) = \sum_{k=1}^{Q} \mathbb{E}[\mathbb{1}_e] \frac{A_{i,j}}{Qp_e} = A_{i,j},$$

where $\mathbb{1}_e$ represents the event that the edge $e = (v_i, v_j)$ being sampled and we have $\mathbb{E}[\mathbb{1}_e] = p_e$.

It is straight-forward to see that $|\mathcal{E}'| = \mathcal{O}(Q)$. Intuitively, the approximation error will diminish if $Q$ approaches infinity, resulting in a denser $\mathcal{G}'$. The challenge is how to choose a good $\{p_e\}_{e \in \mathcal{E}}$ such that we can set $Q$ as small as possible under the given error.
In theory, Spielman & Srivastava (2011) shows that \( G' \) is an accurate approximation of the input graph \( G \) if we set \( p_e \) in proportional to the effective resistance \( R_e \) for each edge \( e \). In spectral graph theory, the effective resistance \( R_e \) is often used as the distance measure between two nodes by encoding the global topology of the graph (e.g., the cluster structure) Lovász (1993). Specifically, for each edge \( e = (u, v) \), the effective resistance \( R_e \) is defined as

\[
R_e = (X_u - X_v)\top L^+(X_u - X_v),
\]

where \( L = I - D^{-\frac{1}{2}} AD^{-\frac{1}{2}} \) is the normalized graph laplacian matrix and \( L^+ \) is the psudo-inverse of \( L \). \( X_u \in \mathbb{R}^{|V|} \) is the elementary unit vector with a coordinate 1 at position \( u \). When the sampling probability \( p_e \) is in proportional to \( R_e \), Spielman & Srivastava (2011) shows that the sample complexity of \( Q \) is \( \mathcal{O}(|V| \log |V|) \), where \( \epsilon \) is a constant which controls the approximation error.

Despite the good theoretical properties, estimating the effective resistance \( R_e \) is non-trivial because (1) it requires approximating the pseudoinverse of the graph laplacian matrix, which is very time consuming for large graphs. (2) it is hard to parallelize the calculation for each edge \( e \).

Albeit Algorithm 1 is well-established, up to our knowledge, there is no previous work investigating its usage for GNNs. This is because even estimating \( R_e \) is extremely time consuming, which counteracts the acceleration effects of graph sparsification.

In the next subsection, we will discuss how to approximate the effective resistance efficiently.

### 3.2 Efficiently approximating effective resistance

Here we introduce how we approximate the effective resistance of each edge using only its local information (e.g., the node degree). Specifically, the following Theorem shows that for any edge \( e = (u, v) \), its effective resistance \( R_e \) is bounded by \( \frac{1}{d_u} + \frac{1}{d_v} \), which can be easily calculated since it only involves the node degrees.

**Theorem 1** (Corollary 3.3 in Lovász (1993)). For all \( e = (u, v) \in E \), we have \( \frac{1}{2}(\frac{1}{d_u} + \frac{1}{d_v}) \leq R_e \leq \frac{1}{\alpha}(\frac{1}{d_u} + \frac{1}{d_v}) \), where \( \alpha (\alpha \leq 2) \) is the smallest non-zero eigenvalue of \( L = I - D^{-\frac{1}{2}} AD^{-\frac{1}{2}} \).

In Spectral graph theory, \( \alpha \) indicates the connectivity of a graph and the bound is tight for well-connected graphs. Informally, the intuition is that for large graphs with cluster structures, the random walk is hard to escape from the local cluster since most of the edges are pointing to nodes within the same cluster. Thus, the local information dominates the information flow on the whole graph. From Theorem 1 instead of setting \( p_e \propto R_e \), we propose to set \( p_e' \propto \frac{1}{d_u} + \frac{1}{d_v} \) for edge \( e = (u, v) \).

**Experimental Analysis.** In Figure 2 we display the distribution of \( \frac{|p_e' - p_e|}{p_e} \) for Reddit dataset, where \( p_e \propto R_e \) and \( p_e' \propto \frac{1}{d_u} + \frac{1}{d_v} \). We note that obtaining exact \( R_e \) is impractical on large graphs since it is very time consuming. To get a sense, even the previous state-of-the-art approximation method still need 248 seconds for estimating \( R_e \). In contrast, calculating \( \frac{1}{d_u} + \frac{1}{d_v} \) for all edge \( e = (u, v) \) consumes only 0.6 second on the same hardware. Experimentally, by setting \( p_e \propto \frac{1}{d_u} + \frac{1}{d_v} \) for each edge \( e = (u, v) \), applying Algorithm 1 to GNNs has negligible effects on the model accuracy (see Table 1), but significantly accelerates both the training and inference processes.

**Theoretical Analysis.** Here we theoretically analyze why our degree-based sampling prorduces a good approximation of the original graph. Formally, we have the following Theorem:

**Theorem 2** (Proof in Appendix 3). Given an input graph \( G = (V, E) \), let \( A \) be the associated adjacency matrix and \( \alpha \) be the smallest non-zero eigenvalue of \( L \). Given an error parameter \( \epsilon \), If we set \( Q = \mathcal{O}(\frac{|V| \log |V|}{\epsilon^2}) \),

\[
\frac{|p_e - p_e'|}{p_e} < \epsilon
\]

for all edges in Reddit dataset.
and for each edge \( e = (u, v) \), we set \( p_e \propto \frac{1}{d_u} + \frac{1}{d_v} \), Algorithm 1 produces a sparsified graph \( G' = (V', E') \) with \( A' \), for any vector \( x \in \mathbb{R}^{|V|} \), we have

\[
(1 - \frac{\epsilon}{\alpha}) \sum_{(u, v) \in E} (x_u - x_v)^2 A_{u,v} \leq \sum_{(u, v) \in E'} (x_u - x_v)^2 A'_{u,v} \leq (1 + \frac{\epsilon}{\alpha}) \sum_{(u, v) \in E} (x_u - x_v)^2 A_{u,v}. \tag{4}
\]

Informally, Equation (4) suggests that the degree-based sampling preserves the graph spectral information, i.e., the eigenvalues of the normalized Laplacian matrix. Specifically, let \( \lambda_1 \geq \lambda_2 \cdots \lambda_{|V|} = 0 \) be the eigenvalue of \( L \) associated with \( G \), and \( \lambda'_1 \geq \lambda'_2 \cdots \lambda'_{|V|} = 0 \) be the eigenvalues of \( L' \) associated with the sparsified \( G' \) given by Algorithm 1, we have:

\[
(1 - \frac{\epsilon}{\alpha}) \lambda_i \leq \lambda'_i \leq (1 + \frac{\epsilon}{\alpha}) \lambda_i. \tag{5}
\]

Here we defer the mathematical details of Equation (5) to Appendix B. According to the graph theory, small (large) eigenvalues indicate the global clustering (local smoothness) structure of the graphs \cite{zhang2019fastgcn}. Thus if the eigenvalues are similar, then most of the graph properties will be preserved, e.g., cluster structures and Cheeger constant \cite{west2001introduction}. Later we experimentally show that both the largest and smallest eigenvalues are preserved by DSpar (Section 5.2.2). In the next Section, we show that preserving spectral information is crucial for maintaining the quality of node representations.

### 3.3 Why degree-based sparsification works for GNNs?

Here we analyze why the sparsification works for GNNs theoretically. High-levely speaking, the reason there is a term “Convolutional” in the name of GCN is that GCN learn node representation by extracting the spectral information of the graph. According to Theorem 2, degree-based sampling preserves the spectrum (e.g., eigenvalues) of the input graph. Thus, the learned node representation should be similar.

Here we analyze the behaviour of GCNs with sparsified graph for simplicity. We note that our analysis can be applied to other GNNs. Let \( G \) be the original graph and \( G' \) be the graph sparsified by degree-based sampling. Specifically, let \( H^{(l)} \) and \( H'^{(l)} \) be the node embeddings learned on the original graph \( G \) and sparsified graph \( G' \) at \( l \)-th layer of GCNs, respectively. Let \( \lambda_1 \) and \( \alpha \) be the largest and the smallest non-zero eigenvalue of the original \( L \), respectively. We have:

**Theorem 3** (Proof in Appendix B).

\[
\|H^{(l+1)} - H'^{(l+1)}\|_F < \frac{\lambda_1}{\alpha} \|H^{(l)} \Theta^{(l)}\|_F. \tag{6}
\]

The above Theorem shows that our proposed degree-based spectral sparsification leads to good approximations of node embeddings learned on the original graphs, especially for well-connected graph.

**Experimental Analysis:** Here we experimentally validate whether our proposed degree-based spectral sparsification leads to good approximations of node embeddings learned on the original graphs. Specifically, we evaluate the same GNN on two graphs, the original and the sparsified graph. To get a sense on scale, in Figure 3 we plot the relative Frobenius norm error \( \frac{\|H^{(l+1)} - H'^{(l+1)}\|_F}{\|H^{(l+1)}\|_F} \) against the error term \( \epsilon \) for Cora dataset \cite{sen2008collective}, where the model is a two-layer GCN with hidden size 32. In summary, we observe for small \( \epsilon \) (\( \leq 0.35 \)), the relative Frobenius norm error is about 5% ~ 10%. Then the error will grow rapidly to 30% when \( \epsilon = 1.0 \). Thus the error is acceptable when a proper \( \epsilon \) is selected. Later we experimentally show that the model accuracy drop is negligible even with a very aggressive \( \epsilon \).
4 Related Works and Discussion

Subgraph-based GNN training. The key idea of this line of work is to train GNNs with sampled subgraphs to reduce the number of nodes retained in memory. Based on this idea, various sampling techniques have been proposed, including the node-wise sampling [Hamilton et al. (2017); Chen et al. (2017)], layer-wise sampling [Huang et al. (2018); Zou et al. (2019)], and subgraph sampling [Chiang et al. (2019); Zeng et al. (2020)]. Similar to DSpar, subgraph-based methods also introduce the error during the training process. However, the key difference is that subgraph-based methods generate several subgraphs before training, and use different subgraphs at each training step. In contrast, DSpar only outputs one sparsified graph and uses it for both training and inference. Thus, methods in this category are orthogonal to DSpar. We experimentally show that DSpar can be integrated with subgraph-sampling based methods to achieve a better efficiency without loss of accuracy.

System-level Acceleration. This research line can be roughly divided into two categories. First, some works propose distributed GNNs training systems, which focus on minimizing the communication cost among hardware [Zheng et al., 2020b; Wan et al., 2022a,b]. Second, another research line optimizes the memory access pattern of sparse operations via coalescing the memory access and fusing consecutive operations [Zhang et al., 2022; Huang et al., 2020; Rahman et al., 2021; Wang et al., 2021]. We note that our work is orthogonal to system-level acceleration methods.

Approximation. Some works try to replace the expensive sparse operations with their cheaper approximated versions [Narayanan et al., 2022; Liu et al., 2022a,b,2023c; Wang et al., 2022]. For example, Liu et al. (2022a) replaces the original SpMM operations with their faster-but-inaccurate versions.

Graph Sparsification. Graph sparsification provides a “sketch” of the input graph by removing redundant edges. Traditional graph sparsification uses efficient resistance as the importance score for removing the edges [Spielman & Srivastava, 2011]. However, as we analyzed, these methods are not practical on large graphs. Then another research line tries to learn the edge importance score. Specifically, Zheng et al. (2020a) proposes a learning-based graph sparsification method which removes potentially task-irrelevant edges from input graphs [Li et al., 2020] formulates the graph sparsification problem as an optimization objective which can be solved by alternating direction method of multipliers (ADMM). Chen et al. (2021) proposes to co-simplify the input graph and GNN model by extending the iterative magnitude pruning to graph areas. However, we note that these learning-based sparsification methods have extra training process, and thus introduces significant ahead-of-training overhead. Moreover, learning-based methods are not scalable since it need to assign each edge an extra trainable mask variable, which is extremely expensive for large graphs.

4.1 Comparison to other sampler using node degree information

GraphSAINT edge sampler. Both the edge sampler in GraphSAINT and DSpar assign $p_e \propto \frac{1}{d_u} + \frac{1}{d_v}$. Here we would like to highlight three key difference between them.

- First, they generate subgraphs differently. The edge sampler in GraphSAINT is used to build node-induced subgraph [Zeng et al., 2020]. Namely, it first selects a subset of anchor nodes using the edge sampler and including all the edges that connect those nodes. In other words, the induced subgraph may contain edges that are not sampled. In contrast, the graph sparsified by DSpar can be viewed as edge-induced subgraph. Namely, DSpar first selects a subset of edges from the original graph and includes only those nodes that are endpoints of the selected edges.

- Second, they are executed differently. One key difference between DSpar and the graph sampler (e.g., edge sampler and FastGCN sampler [Chen et al., 2018]) is that DSpar only sparsify the graph once before training. In contrast, the graph sampler generates different subgraphs at each training step.

- Third, they are derived differently and thus are used differently. $\frac{1}{d_u} + \frac{1}{d_v}$ in GraphSAINT edge sampler is obtained by debiasing node embeddings in the sampled subgraph [Zeng et al., 2020]. We note that this bias term stems from using the node-induced subgraph. In contrast, DSpar is derived following the traditional spectral graph theory, which mainly focuses on providing one light-weight sketch of the original graph.
The FastGCN sampler [Chen et al., 2018] shares some similarities with the DSpar method, as both techniques employ node degree information. However, they utilize this information in significantly different ways. Specifically, for a given batch of nodes, FastGCN samples neighbors for each in-batch node with a probability proportional to the square of the node degree. In contrast, DSpar uses node degree to subsample edges with a probability proportional to $\frac{1}{d_u} + \frac{1}{d_v}$ for any edge $e$ in the graph.

This difference in sampling strategies leads to distinct outcomes. In FastGCN, a neighbor with a higher degree has a greater chance of being sampled for a given node. Conversely, in the DSpar method, an edge is less likely to be sampled if the degrees of its endpoints are large. Moreover, FastGCN sampler is developed for selecting neighbors for a given node, which cannot be directly extended to the area of graph sparsification.

4.2 Limitations

Here we would like to briefly discuss the limitation of our work. First, unlike previous theoretical pioneer on GNN generalization (Li et al., 2022; Zhang et al., 2023), Theorem 3 is not directly related to the generalization of GNNs trained on sparsified graphs. However, we would like to note that our main theoretical result (Theorem 2) is derived without making any assumptions. Obtaining a meaningful generalization bound for GNNs without any assumptions is highly challenging. We leave it as a future work. Thus, the generalization of GNNs trained on large graphs is only experimentally verified.

5 Experiment

We verify the effectiveness of our proposed framework through answering the following research questions:

- **Q1**: How effective is the proposed DSpar in terms of model accuracy and efficiency compared to the random sparsification? What is the sampling time overhead of DSpar?
- **Q2**: How effective is DSpar in terms of the preserved spectral information, i.e., eigenvalues?
- **Q3**: How efficient is DSpar compared to the baseline during inference?
- **Q4**: How sensitive is DSpar to its key hyperparameters?

5.1 Experimental Settings

We first introduce the applied baselines and datasets. Then we introduce the evaluation metrics for measuring the speed, and accuracy, respectively. Finally, we introduce the hyperparameter settings for DSpar. Following previous works Fey et al. (2021); Hu et al. (2020); Duan et al. (2022), we focus on the transductive node classification, which is also the most common task in large-scale graph benchmarks.

5.1.1 Datasets and Baselines

To evaluate DSpar, we adopt four common large scale graph benchmark datasets from different domains, namely, Reddit Hamilton et al. (2017), Yelp Zeng et al. (2020), ogbn-arxiv Hu et al. (2020), ogbn-proteins Hu et al. (2020) and ogbn-products Hu et al. (2020). We evaluate DSpar under both the mini-batch training and full-batch training settings. For the mini-batch training setting, we integrate DSpar with the state-of-the-art subgraph sampling methods, i.e., GraphSAINT Zeng et al. (2020). When integrating DSpar with subgraph sampling methods, we first sparsify the input graph by Algorithm 1 and then sample subgraphs from the sparsified graph. For the full-batch training setting, we integrate DSpar with three popular models: two commonly used shallow models, namely, GCN Kipf & Welling (2017) and GraphSAGE Hamilton et al. (2017), and one deep model GCNII Chen et al. (2020). To avoid creating confusion, GCN, GraphSAGE, and GCNII are all trained with the whole graph at each step. For a fair comparison, we use the mean aggregator for GraphSAGE and GraphSAINT throughout the paper. Details about the hyperparameters are in Appendix C.

We compare DSpar with random sparsification (“random” in Table 1), where each edge has equal probability to be removed. We note that we only compare to the random sparsification mainly because (1) learning-based
Table 1: Comparison of test accuracy (↑) and training throughput (↑) on five datasets. Gray cells indicate the accuracy drop is negligible (≈ 0.3%) or the result is better compared to the baseline. The hardware here is a single NVIDIA A40 (48GB). All reported results are averaged over ten random trials.

<table>
<thead>
<tr>
<th>Model</th>
<th>Methods</th>
<th># nodes</th>
<th># edges</th>
<th>Acc. (epoch/s)</th>
<th>Throughput (epoch/s)</th>
<th>F1-micro</th>
<th>Throughput (epoch/s)</th>
<th>Acc. (epoch/s)</th>
<th>Throughput (epoch/s)</th>
<th>ROC-AUC (epoch/s)</th>
<th>Acc. (epoch/s)</th>
<th>Throughput (epoch/s)</th>
<th>Δ Acc.</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphSAINT</td>
<td>Baseline</td>
<td>230K</td>
<td>11.6M</td>
<td>95.39</td>
<td>9.39</td>
<td>64870.61</td>
<td>0.92</td>
<td>47140.10</td>
<td>12.94</td>
<td>95.410.80</td>
<td>0.30</td>
<td>79.740.23</td>
<td>0.19</td>
<td>0.9 1.6x</td>
</tr>
<tr>
<td></td>
<td>+random</td>
<td>95.66</td>
<td>95.33</td>
<td>95.66</td>
<td>95.66</td>
<td>95.66</td>
<td>95.66</td>
<td>95.66</td>
<td>95.66</td>
<td>95.66</td>
<td>95.66</td>
<td>95.66</td>
<td>95.66</td>
<td>95.66</td>
</tr>
<tr>
<td></td>
<td>+DSpar</td>
<td>95.41</td>
<td>95.41</td>
<td>95.41</td>
<td>95.41</td>
<td>95.41</td>
<td>95.41</td>
<td>95.41</td>
<td>95.41</td>
<td>95.41</td>
<td>95.41</td>
<td>95.41</td>
<td>95.41</td>
<td>95.41</td>
</tr>
<tr>
<td>GCN</td>
<td>Baseline</td>
<td>230K</td>
<td>7.0M</td>
<td>95.39</td>
<td>9.39</td>
<td>64870.61</td>
<td>0.92</td>
<td>47140.10</td>
<td>12.94</td>
<td>95.410.80</td>
<td>0.30</td>
<td>79.740.23</td>
<td>0.19</td>
<td>0.9 1.6x</td>
</tr>
<tr>
<td></td>
<td>+random</td>
<td>94.97</td>
<td>10.29</td>
<td>95.33</td>
<td>95.33</td>
<td>95.33</td>
<td>95.33</td>
<td>95.33</td>
<td>95.33</td>
<td>95.33</td>
<td>95.33</td>
<td>95.33</td>
<td>95.33</td>
<td>95.33</td>
</tr>
<tr>
<td></td>
<td>+DSpar</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
</tr>
<tr>
<td>SAGE (full batch)</td>
<td>Baseline</td>
<td>230K</td>
<td>9.57 (2.5x)</td>
<td>95.39</td>
<td>9.39</td>
<td>64870.61</td>
<td>0.92</td>
<td>47140.10</td>
<td>12.94</td>
<td>95.410.80</td>
<td>0.30</td>
<td>79.740.23</td>
<td>0.19</td>
<td>0.9 1.6x</td>
</tr>
<tr>
<td></td>
<td>+random</td>
<td>95.66</td>
<td>9.57</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
</tr>
<tr>
<td></td>
<td>+DSpar</td>
<td>95.39</td>
<td>9.57</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
</tr>
<tr>
<td>GCNII</td>
<td>Baseline</td>
<td>230K</td>
<td>3.99 (1.6x)</td>
<td>95.39</td>
<td>9.39</td>
<td>64870.61</td>
<td>0.92</td>
<td>47140.10</td>
<td>12.94</td>
<td>95.410.80</td>
<td>0.30</td>
<td>79.740.23</td>
<td>0.19</td>
<td>0.9 1.6x</td>
</tr>
<tr>
<td></td>
<td>+random</td>
<td>95.66</td>
<td>4.06</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
</tr>
<tr>
<td></td>
<td>+DSpar</td>
<td>95.39</td>
<td>4.06</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
<td>95.39</td>
</tr>
</tbody>
</table>

Sparsification introduces significant ahead-of-training overhead because it requires extra learning processes to identify unimportant edges. Moreover, learning-based methods are not scalable since it needs to assign each edge an extra trainable mask variable, which is extremely expensive for large graphs. Chen et al. (2021) estimating effective resistance for all edges in large graphs is not practical.

5.1.2 Evaluation metrics

We comprehensively investigate the practical usefulness of our proposed method by evaluating the trade-off between the speed and accuracy. Specifically,

**Speed**: Based on FLOPs, SpMM is theoretically much faster than Matmul. However, this is not true in practice due to the random memory access pattern of SpMM, which cannot be efficiently accelerated on CPUs and GPUs (Han et al. 2016). To evaluate the practical usage of our method, we measure the actual running speed on the off-the-shelf hardware. For **Training speed**, we measure the hardware throughput on GPUs (epoch/s). For **Inference speed**, we measure the latency on GPUs (ms). For **Ahead-of-training overhead**, we measure the wall clock time (s).

**Accuracy**: Following Hu et al. (2020), Zeng et al. (2020), we use the test accuracy for evaluating performance on Reddit, ogbn-arxiv, and ogbn-products datasets. The F1-micro is used for Yelp dataset. The ROC-AUC is used for ogbn-proteins dataset.

5.1.3 Hyperparameter Settings

For graph sparsification, the hyperparameter is the number of trials Q in Algorithm 1. According to Theorem 2, we set \( Q = \frac{|V| \log |V|}{\epsilon^2} \), where \( \epsilon \) controls the sparsity and the approximation errors. We alter the value of \( \epsilon \) to see how it affects the accuracy drop. Specifically, we vary \( \epsilon \) from 0.3 to 1.5.

5.2 Accuracy versus Training Efficiency (Table 1)

5.2.1 The training efficiency

To answer Q1, we summarize the training throughput and the accuracy of different methods in Table 1. We also report the model performance under random sparsification, i.e., “random” in Table 1. For a fair comparison, we control the number of removed edges roughly the same for both DSpar and random sparsification. In this way, they should have the same acceleration effect. We show the sparsification effect of DSpar in Table 2. The ahead-of-training overhead of DSpar is given in Table 3. We also show how the optimization process affected by DSpar in Figure 3. We observe:

- Regardless of subgraph training (i.e., GraphSAINT) or full graph training (i.e., GCN, GraphSAGE, and GCNII), the accuracy drop of applying DSpar over baselines is negligible (≈ 0.3%) across
Table 2: The effect of DSpar (Algorithm 1), which removes \( \approx 25 \sim 95\% \) edges for different datasets, according to graph statistics. When applying DSpar over subgraph sampling methods, the sparsity may need to be decreased to guarantee the accuracy since subgraph sampling methods also introduce extra error.

| Dataset       | Graph Sparsification Rate \((= 1 - \frac{|E'|}{|E|})\) | Sampling time (s) |
|---------------|----------------------------------------------------|-------------------|
| Reddit        | 82.1%                                               | 2.05              |
| Yelp          | 73.5%                                               | 1.62              |
| ogbn-arxiv    | 26.7%                                               | 1.4               |
| ogbn-proteins | 95.0%                                               | 1.5               |
| ogbn-products | 79.4%                                               | 11.0              |

Table 3: Ahead-of-training overhead (sampling time) of DSpar

different models and datasets. This can be explained by Theorem 3 that GNNs still learn expressive node representation on graphs sparsified by DSpar. As we mentioned at the beginning of Section 5.2, we compare the model accuracy drop of DSpar and random sparsification under the same speedup. Interestingly, for Yelp and ogbn-proteins, random sparsification performs almost the same compared to DSpar. However, the accuracy drop of random sparsification is much larger on other datasets, especially on ogbn-products. We hypothesize this might because for Yelp and ogbn-proteins, the node features are more important to the model accuracy compared to the edge connectivity.

- DSpar can significantly reduce the training time of GNNs with even better accuracy. As shown in Table 1, we summarize the training throughput (epoch/s) and speedup over baseline on a single NVIDIA A40 GPU. Notably, DSpar achieves non-trivial trade-off between accuracy and efficiency. Namely, in average DSpar brings \( \approx 2.5 \times \) speedup for full-graph training and \( 1.4 \times \) speedup for mini-batch training, with almost no accuracy drop or even better accuracy. The better accuracy can be explained by the fact that removing edges may impose regularization effect during training Rong et al. (2019).

- DSpar removes \( \approx 25\% \sim 95\% \) edges across different datasets, with negligible ahead-of-time overhead. As shown in Table 2, we summarize the sparsification effect of DSpar. According to Theorem 2 and Theorem 3, \( Q \) in Algorithm 1 should be scale with the graph connectivity, measure by the ratio of largest eigenvalue \( \lambda_1 \) to the smallest non-zero eigenvalue \( \alpha \) of the graph Laplacian matrix. In practice, we absorb the term of \( \frac{\lambda_1}{\alpha} \) into \( \epsilon \). Specifically, we control the sparsify through directly tuning \( \epsilon \) with \( Q = \frac{|V| \log |V|}{\epsilon^2} \). When applying DSpar over subgraph sampling methods, the sparsity may need to be decreased to guarantee the accuracy since subgraph sampling methods also introduce extra error. For the ahead-of-training overhead, i.e., the running time of Algorithm 1, is reported in Table 3. We note that we provide an optimized sampler for sampling edges in large graphs, details are elaborated in Appendix A. We summarize that the overhead is negligible compared to the acceleration effect of graph sparsification.

- DSpar almost has the same convergence behaviour compared to the baseline. We investigate the convergence speed of DSpar, where the convergence speed is measured by the gap in training loss between consecutive epochs. Figure 4 summarizes the training curves of GNNs trained with different
methods on Reddit and ogbn-arxiv dataset. We observe that DSpar almost has the same convergence behaviour compared to the baseline. This is consistent with our theoretical analysis that DSpar leads to good approximations of node embeddings learned on the original graphs.

5.2.2 Can DSpar preserves the graph spectral information, i.e., eigenvalues?

As we analyzed, preserving graph spectral information is crucial for learning meaningful representations on graphs. To answer Q2, we calculate the relative error of the eigenvalue of the graph Laplacian matrix \( \mathcal{L} \) on ogbn-arxiv and Reddit dataset, respectively. **We note that it is almost impossible to calculate all of the eigenvalues of a large graph.** As we mentioned, small (large) eigenvalues indicate the global clustering (local smoothness) structure of the graphs. Thus in Figure 5, we instead calculate the top-200 and bottom-200 eigenvalues of \( \mathcal{L} \) corresponding to the original graph and sparsified graph, respectively. We observe:

- **DSpar preserves most of the key eigenvalues.** Specifically, we observe that for both ogbn-arxiv and Reddit dataset, DSpar significantly outperforms the random sparsification in terms of the relative error of eigenvalues. For the top 200 eigenvalues, which indicate the global clustering structure, DSpar provides an accurate approximation. That means the cluster structure is well-preserved by DSpar. However for the bottom 200 eigenvalues, which indicate the local smoothness, DSpar has \( \approx 5 \sim 40\% \) relative errors for these bottom eigenvalues. In contrast, random sparsification almost has 100% relative error, i.e., failing to preserve this part of information. Our analysis also partially explains why GNNs learned on graph with random sparsification have much worse performance on these two datasets.
Inference Latency (ms)

<table>
<thead>
<tr>
<th>Graph Sparsity (%)</th>
<th>Test Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td>92.8</td>
</tr>
<tr>
<td>93</td>
<td>95.1</td>
</tr>
<tr>
<td>94</td>
<td>116.5</td>
</tr>
<tr>
<td>95</td>
<td>164.1</td>
</tr>
<tr>
<td>96</td>
<td>226.06</td>
</tr>
<tr>
<td>97</td>
<td>655.74</td>
</tr>
</tbody>
</table>

Figure 6: Inference latency comparison on a single NVIDIA A40 (48GB) GPU (lower is better). “OOM” means out-of-memory. Compared to the baseline, DSpar reduce $\approx 30\% \sim 90\%$ inference latency.

5.2.3 The Inference efficiency

For GraphSAINT, the subgraph sampling is only applied to the training process. Thus, they do not affect the inference latency of GNNs. To answer Q3, Figure 6 compares the inference latency of different models on the original graph and the graph sparsified by DSpar. We observe that

- \boxed{\text{According to Figure 6, DSpar reduces $\approx 30\% \sim 90\%$ inference latency. Notably, on Yelp dataset, DSpar reduces only $30\%$ inference latency, although it removes more than $70\%$ edges (Table 2). This is mainly because for Yelp dataset, graph-based operations account for a relatively small percentage of the total time compared to other datasets. For other datasets, we observe that DSpar significantly reduces up to $\approx 90\%$ inference latency.}}

Figure 7: Accuracy versus the graph sparsity on Reddit dataset. Here the graph sparsity equals the percentage of removed edges. All results are averaged over ten random trials.

5.3 Hyperparameter sensitivity analysis

As we analyzed, DSpar has only one hyperparameter, namely, the number of trial $Q$ for controlling the graph sparsity. In this subsection, to answer Q4, we explore the sensitivity of hyperparameters $Q$ for DSpar. As we mentioned, we set $Q = |V| \log |V|$ according to Theorem 3, where $\epsilon$ controls the approximation error. We alter the value of $\epsilon$ from 0.3 to 1.5 to check the relationship between the graph sparsity and the accuracy drop. As shown in Figure 7, the accuracy drop becomes larger when edges are being removed. This is consistent with the theoretical analysis in Section 3.3 that we have larger approximation errors when the graph sparsity is high. In addition to studying dense graphs like Reddit, we also performed a sparsification experiment on a large sparse graph, specifically ogbn-products, using GraphSAGE. We observe that in this case, the accuracy drop of random sparsification and DSpar is more significant. This is mainly because ogbn-products is much sparser than Reddit, making it more sensitive to edge removal. We also observe that DSpar significantly outperforms the random baseline, which is consistent.
with the theoretical analysis. Furthermore, when we remove roughly 95% of edges in Reddit dataset, the accuracy drop is roughly 0.8 ∼ 1.2% for different models, which is still acceptable when considering a 95% sparsity. In practice, we suggest selecting $Q = \frac{|V| \log |V|}{\epsilon^2}$ according to the accuracy drop and the efficiency constraint by adjusting $\epsilon$.

6 Conclusions and Future work

We propose DSpar, a simple-yet-effective framework for training GNNs with compressed tensors, which can be plugged into most of the existing solutions to save memory. We demonstrate the potential of DSpar for the practical usage by systematically evaluating the trade-off among the memory-saving, time overhead, and accuracy drop. Future work includes (1) evaluating DSpar under the distributed training setting; (2) exploring other types of graph sparsification methods.

References


A Implementations

We use CPUs for sparsifying graph using Algorithm 1, which because the whole graph may exceed the GPU memory. We emphasize that Algorithm 1 sparsifies the graph by sampling with replacement (line 3). Thus, the sampling process can be easily paralleled. In practice, to avoid the numerical precision problem for graphs with more than $2^{24}$ edges \(^1\) we use \texttt{torch.double} as the precision for the probability tensor \(\{p_e\}_{e \in E}\), which is not supported for most of the built-in sampling methods in Pytorch. Thus, we implement Algorithm 1 in C++ and parallel it base on OpenMP. We build it as an extension for Pytorch, which can be called using Pytorch API.

B Theory

B.1 Proof of Theorem 2

Theorem 2 (Proof in Appendix B). Given an input graph \(G = (V, E)\), let \(A\) be the associated adjacency matrix and \(\alpha\) be the smallest non-zero eigenvalue of \(L\). Given an error parameter \(\epsilon\), If we set \(Q = O\left(\frac{|V| \log |V|}{\epsilon^2}\right)\) and for each edge \(e = (u, v)\), we set \(p_e \propto \frac{1}{d_u} + \frac{1}{d_v}\), Algorithm 1 produces a sparsified graph \(G' = (V', E')\) with \(A'\), for any vector \(x \in \mathbb{R}^{|V|}\), we have

\[
(1 - \frac{\epsilon}{\alpha}) \sum_{(u,v) \in E} (x_u - x_v)^2 A_{u,v} \leq \sum_{(u,v) \in E'} (x_u - x_v)^2 A'_{u,v} \leq (1 + \frac{\epsilon}{\alpha}) \sum_{(u,v) \in E} (x_u - x_v)^2 A_{u,v}. \tag{4}
\]

Proof. Below we first introduce the necessary concepts and tools for deriving Theorem 2. Let \(B \in \mathbb{R}^{|E| \times |V|}\) be the signed edge-vertex incidence matrix, given by

\[
B_{(e,v)} = \begin{cases} 
1 & \text{if } v \text{ is edge } e \text{'s head}, \\
-1 & \text{if } v \text{ is edge } e \text{'s tail}, \\
0 & \text{otherwise}.
\end{cases} \tag{7}
\]

Let \(W \in \mathbb{R}^{|E| \times |E|}\) be a diagonal matrix such that \(W_{e,e} = A_e\). Then we define the projection matrix \(\Pi\) as

\[
\Pi = \frac{1}{2} B L^+ B^T \frac{1}{2}. \tag{8}
\]

We note that \(\Pi\) have the following properties \cite{Spielman2011}:

\begin{itemize}
  \item \(\Pi\) is a diagonal projection matrix,
  \item \(\|\Pi_{e,e}\|^2_2 = \Pi_{e,e},\)
  \item \(\Pi^2 = \Pi,
  \item \(\Pi_{e,e} = A_e R_e.\)
\end{itemize}

Let \(S \in \mathbb{R}^{|E|}\) be the diagonal random matrix representing the sampling process in Algorithm 1 such that

\[
S_{e,e} = \frac{\left(\text{# of times } e \text{ is sampled}\right)}{Q p_e}. \tag{8}
\]

Then we have the following Lemma:

Lemma 1 (Lemma 4 in \cite{Spielman2011}). Let \(S\) be the diagonal random matrix representing the sampling process such that

\[
\|\Pi S \Pi - \Pi\Pi\| \leq \epsilon,
\]

Then

\[
\forall x \in \mathbb{R}^{|V|}, (1 - \epsilon) x^T \mathcal{L} x \leq x^T \mathcal{L}' x \leq (1 + \epsilon) x^T \mathcal{L} x, \tag{9}
\]

where \(\mathcal{L} = B^T W B\) is the normalized Laplacian matrix and \(\mathcal{L}' = B^T W \frac{1}{2} S W \frac{1}{2} B\) is the sparsified Laplacian matrix.

---

\(^1\)https://www.mathworks.com/help/matlab/ref/flintmax.html
Sampling $Q$ edges from $G$ corresponds to sampling $Q$ columns from $\Pi$, so we can write

$$
\Pi S \Pi = \sum_e S_{e,e} \Pi_{e,e} \Pi_{e,e}^T
$$

(10)

$$
= \sum_e \left( \frac{\text{# of times } e \text{ is sampled}}{Q p_e} \right) \Pi_{e,e} \Pi_{e,e}^T
$$

(11)

$$
= \frac{1}{Q} \sum_e \left( \frac{\text{# of times } e \text{ is sampled}}{p_e \sqrt{p_e}} \right) \Pi_{e,e} \Pi_{e,e}^T
$$

(12)

$$
= \frac{1}{Q} \sum_{i=1}^Q y_i y_i^T,
$$

(13)

where $y_1, \cdots, y_Q$ are drawn independently with replacement from the distribution

$$
y = \frac{\Pi_{e,e}}{\sqrt{p_e}} \text{ with probability } p_e.
$$

(14)

Recall that according to Theorem 1, we know that for any edge $e = (u, v)$, we have $\frac{1}{2} (\frac{1}{d_u} + \frac{1}{d_v}) \leq R_e \leq \frac{1}{\alpha} (\frac{1}{d_u} + \frac{1}{d_v})$. Let $p_e$ and $p_e'$ be the sampling probability for edge $e$ in proportional to $R_e$ and $\frac{1}{d_u} + \frac{1}{d_v}$, respectively. Let $S$ and $S'$ be the random matrix with $p_e$ and $p_e'$, respectively. Let $y$ and $y'$ be the distribution defined above associated with $p_e$ and $p_e'$, respectively. Then

$$
p_e' = \frac{\frac{1}{d_u} + \frac{1}{d_v}}{\sum_{e=(u,v) \in E} \frac{1}{d_u} + \frac{1}{d_v}} \geq \frac{\alpha R_e}{2 \sum_{e=(u,v) \in E} R_e} = \frac{\alpha}{2} p_e.
$$

(15)

For the norm of $y$, we have:

$$
\|y\|_2^2 = \frac{1}{p_e} \|\Pi_{e,e}\|_2^2 = \frac{1}{p_e} \Pi_{e,e}^2 = \sum_{e=(u,v) \in E} \frac{R_e}{R_e} = \sum_e R_e = |V| - 1.
$$

We note that we have $\sum_e R_e = |V| - 1$ from the definition of effective resistance Spielman & Srivastava (2011).

Then we have

$$
\|y\|_2^2 = \frac{\|\Pi_{e,e}\|_2^2}{p_e'} \leq \frac{2}{\alpha} \frac{\|\Pi_{e,e}\|_2^2}{p_e} = \frac{2}{\alpha} \|y\|_2^2 \leq \frac{2}{\alpha} (|V| - 1)
$$

(16)

Then

$$
\E[||S'\Pi - \Pi\Pi||_2^2] = \E\left[\frac{1}{Q} \sum_{i=1}^Q y_i y_i^T - \E y y^T \right]
$$

(17)

**Lemma 2** (Lemma 5 in Spielman & Srivastava (2011)). Let $p$ be the probability distribution over $\Omega \in \mathbb{R}^{|V|}$ such that $\|y\|_2 \leq M$ and $\||Ey y^T\|| \leq 1$, we have

$$
\E\left[\frac{1}{Q} \sum_{i=1}^Q y_i y_i^T - \E y y^T \right]_2 \leq \min(CM \sqrt{\frac{\log Q}{Q}}, 1),
$$

(18)

where $C$ is an absolute constant.

By Taking $Q = 9C^2 |V| \log |V| / \epsilon^2$, we have
By Lemma 1, this completes the proof of the theorem.

B.2 Derive Equation (5)

By Courant-Fischer Theorem, we know that the $i$-th eigenvalue of $\mathcal{L}$ satisfies

$$\lambda_i = \min_{K: \text{dim}(K) = i} \max_{x \in K} \frac{x^\top \mathcal{L} x}{x^\top x},$$

where $\text{dim}(K)$ is the number of linearly independent vectors that form a basis for $K$.

Similarly, for $\mathcal{L}'$, the $i$-th eigenvalue $\lambda'_i$ can be expressed as:

$$\lambda'_i = \min_{K: \text{dim}(K) = i} \max_{x \in K} \frac{x^\top \mathcal{L}' x}{x^\top x}.$$  

(21)

Now, by Theorem 2, we know that for any $x$, we have

$$\left(1 - \frac{\epsilon}{\alpha}\right) x^\top \mathcal{L} x \leq x^\top \mathcal{L}' x \leq \left(1 + \frac{\epsilon}{\alpha}\right) x^\top \mathcal{L} x,$$

(22)

Let $K_i$ be any subspace with dimension $i$. Then, for any vector $x \in K_i$, we have:

$$\left(1 - \frac{\epsilon}{\alpha}\right) \frac{x^\top \mathcal{L} x}{x^\top x} \leq \frac{x^\top \mathcal{L}' x}{x^\top x} \leq \left(1 + \frac{\epsilon}{\alpha}\right) \frac{x^\top \mathcal{L} x}{x^\top x}.$$  

(23)

Suppose $\frac{x^\top \mathcal{L}' x}{x^\top x}$ achieves maximum at $x'$, now we have

$$\max_{x \in K_i} \frac{x^\top \mathcal{L}' x}{x^\top x} = \frac{x'^\top \mathcal{L}' x'}{x'^\top x'},$$

(24)

$$\leq \left(1 + \frac{\epsilon}{\alpha}\right) \frac{x'^\top \mathcal{L} x'}{x'^\top x'},$$

(25)

$$\leq \left(1 + \frac{\epsilon}{\alpha}\right) \max_{x \in K_i} \frac{x^\top \mathcal{L} x}{x^\top x}.$$  

(26)

By applying the similar technique on the left side of the inequality, we have

$$\left(1 - \frac{\epsilon}{\alpha}\right) \max_{x \in K_i} \frac{x^\top \mathcal{L} x}{x^\top x} \leq \max_{x \in K_i} \frac{x^\top \mathcal{L}' x}{x^\top x}.  

(27)

Combine the left side and right side inequality together, we have:

$$\left(1 - \frac{\epsilon}{\alpha}\right) \max_{x \in K_i} \frac{x^\top \mathcal{L} x}{x^\top x} \leq \max_{x \in K_i} \frac{x^\top \mathcal{L}' x}{x^\top x} \leq \left(1 + \frac{\epsilon}{\alpha}\right) \max_{x \in K_i} \frac{x^\top \mathcal{L} x}{x^\top x}.$$  

(28)

Please note that the above inequality holds for any subspace $K_i$ with dimension $i$. Let $\hat{K}_i$ be the subspace that achieves the minimum in the Courant-Fischer theorem for eigenvalue $\lambda_i$ in Equation (20). We have

$$\max_{x \in \hat{K}_i} \frac{x^\top \mathcal{L}' x}{x^\top x} \leq \left(1 + \frac{\epsilon}{\alpha}\right) \max_{x \in K_i} \frac{x^\top \mathcal{L} x}{x^\top x} = \left(1 + \frac{\epsilon}{\alpha}\right) \lambda_i.$$  

(29)
Now by definition of the minimum, we know that
\[
\max_{x \in \mathbf{K}_i} \frac{x^\top \mathbf{L}' x}{x^\top x} \geq \min_{\mathbf{K}_i: \dim(\mathbf{K}_i) = i} \max_{x \in \mathbf{K}_i} \frac{x^\top \mathbf{L}' x}{x^\top x} = \lambda'_i
\]

By connecting the above two inequality together, we have
\[
\lambda'_i \leq (1 + \frac{\epsilon}{\alpha}) \lambda_i. \tag{30}
\]

Similarly, by applying the same technique on the left side, we can obtain
\[
(1 - \frac{\epsilon}{\alpha}) \lambda_i \leq \lambda'_i \tag{31}
\]

By connecting the above inequality together, we obtain
\[
(1 - \frac{\epsilon}{\alpha}) \lambda_i \leq \lambda'_i \leq (1 + \frac{\epsilon}{\alpha}) \lambda_i. \tag{32}
\]

### B.3 Proof of Theorem 3

Theorem 3 is the direct extension of Theorem 2 to GNN area. In this proof, we aim to establish a bound on the difference between the activations of two GCNs with different Laplacian matrices. We compute the Frobenius norm of the difference between these activations and derive an upper bound by applying the triangle inequality and the properties of the matrix 2-norm.

**Theorem 3** (Proof in Appendix B).

\[
\|H^{(l+1)} - H'^{(l+1)}\|_F < \epsilon \frac{\lambda_1}{\alpha} \|H^{(l)}\Theta^{(l)}\|_F. \tag{6}
\]

**Proof.** First, by the definition of GCN, we have
\[
H^{(l+1)} = \text{ReLU}\left((2I - \mathcal{L})H^{(l)}\Theta^{(l)}\right), \tag{33}
\]
\[
H'^{(l+1)} = \text{ReLU}\left((2I - \mathcal{L}')H^{(l)}\Theta^{(l)}\right). \tag{34}
\]

Then we have
\[
\|H^{(l+1)} - H'^{(l+1)}\|_F \leq \|\mathcal{L} - \mathcal{L}'\|_2 \|H^{(l)}\Theta^{(l)}\|_F \leq \|\mathcal{L} - \mathcal{L}'\|_2 \|H^{(l)}\Theta^{(l)}\|_F \tag{35}
\]

The above inequality directly from the fact that for any two matrix \(X\) and \(Y\), we have \(\|XY\|_F \leq \|X\|_2\|Y\|_F \leq \|X\|_F\|Y\|_F\).

By definition of 2-norm, we have
\[
\|\mathcal{L} - \mathcal{L}'\|_2 = \sup_{x: \|x\| = 1} x^\top (\mathcal{L} - \mathcal{L}') x \tag{36}
\]

Suppose the above supremum is achieved at \(x_0\), then
\[
\| (\mathcal{L} - \mathcal{L}') \|_2 = x_0^\top \mathcal{L} x_0 - x_0^\top \mathcal{L}' x_0 \\
\leq x_0^\top \mathcal{L} x_0 - (1 - \frac{\epsilon}{\alpha}) x_0^\top \mathcal{L} x_0 \quad (\text{From Theorem 2}) \tag{37}
\]
\[
= \frac{\epsilon}{\alpha} x_0^\top \mathcal{L} x_0 \\
\leq \sup_{x: \|x\| = 1} \frac{\epsilon}{\alpha} x^\top \mathcal{L} x \\
= \frac{\epsilon}{\alpha} \lambda_1 \tag{38}
\]
By combining the above inequality and Equation (35), we have
\[
\| H^{(l+1)} - H'^{(l+1)} \|_F = \leq \frac{\epsilon}{\alpha} \lambda_1 \| H^{(l)} \Theta^{(l)} \|_F \tag{39}
\]

C Experimental Settings

C.1 Software and Hardware Descriptions

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA</td>
<td>11.1</td>
</tr>
<tr>
<td>pytorch_sparse</td>
<td>0.6.12</td>
</tr>
<tr>
<td>pytorch_scatter</td>
<td>2.0.8</td>
</tr>
<tr>
<td>pytorch_geometric</td>
<td>1.7.2</td>
</tr>
<tr>
<td>pytorch</td>
<td>1.9.0</td>
</tr>
<tr>
<td>OGB</td>
<td>1.3.1</td>
</tr>
</tbody>
</table>

All experiments are conducted on a server with four NVIDIA 3090 GPUs, four AMD EPYC 7282 CPUs, and 252GB host memory. We implement all models based on Pytorch and Pytorch Geometric. During our experiments, we found that the version of Pytorch, Pytorch Sparse, and Pytorch Scatter can significantly impact the running speed of the baseline. Here we list the details of our used packages in all experiments in Table 4.

C.2 Statistics of benchmark datasets

We give the detailed statistics and URLs for all datasets used in our experiments in Table 5. We follow the standard data splits and all datasets are directly downloaded from Pytorch Geometric or the protocol of OGB [Hu et al. (2020)].

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Edges</th>
<th>Features</th>
<th>Classes</th>
<th>Label Rates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reddit</td>
<td>232,965</td>
<td>11,606,919</td>
<td>602</td>
<td>41</td>
<td>65.86%</td>
</tr>
<tr>
<td>Yelp</td>
<td>716,847</td>
<td>6,977,409</td>
<td>300</td>
<td>100</td>
<td>75.00%</td>
</tr>
<tr>
<td>ogbn-arxiv</td>
<td>169,343</td>
<td>1,157,799</td>
<td>128</td>
<td>40</td>
<td>53.70%</td>
</tr>
<tr>
<td>ogbn-proteins</td>
<td>169,343</td>
<td>1,157,799</td>
<td>128</td>
<td>40</td>
<td>53.70%</td>
</tr>
<tr>
<td>ogbn-products</td>
<td>2,449,029</td>
<td>61,859,076</td>
<td>100</td>
<td>47</td>
<td>8.03%</td>
</tr>
</tbody>
</table>
C.3 Hyperparameter Settings

Regarding Reddit, and Yelp dataset, we follow the hyperparameter configurations reported in the respective papers as closely as possible. Following [Fey et al.] (2021), we clips the gradient during training. The “Gradient Clipping” in below tables indicate the maximum norm for gradients. “Gradient Clipping= 0.0” means we do not clip the gradients in that experiment. Regarding ogbn-arxiv and ogbn-products dataset, we follow the hyperparameter configurations and codebases provided on the OGB [Hu et al.] (2020) leader-board. Please refer to the OGB website for more details. Table 9 summarizes the hyperparameter configuration of GraphSAINT. Table 6, Table 7, and Table 8 summarize the hyperparameter configuration of full-Batch GCN, full-Batch GraphSAGE, and full-batch GCNII, respectively.

Table 6: Configuration of Full-Batch GCN.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Learning Rates</th>
<th>Epochs</th>
<th>Dropout</th>
<th>Gradient Clipping</th>
<th>BatchNorm</th>
<th>Layers</th>
<th>Hidden Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reddit</td>
<td>0.01</td>
<td>400</td>
<td>0.5</td>
<td>0.5</td>
<td>Yes</td>
<td>2</td>
<td>256</td>
</tr>
<tr>
<td>Yelp</td>
<td>0.01</td>
<td>500</td>
<td>0.1</td>
<td>0.5</td>
<td>Yes</td>
<td>2</td>
<td>512</td>
</tr>
<tr>
<td>ogbn-proteins</td>
<td>0.01</td>
<td>1000</td>
<td>0.5</td>
<td>0.0</td>
<td>No</td>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>ogbn-arxiv</td>
<td>0.01</td>
<td>500</td>
<td>0.5</td>
<td>0.5</td>
<td>Yes</td>
<td>3</td>
<td>128</td>
</tr>
</tbody>
</table>

Table 7: Configuration of Full-Batch GraphSAGE.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Learning Rates</th>
<th>Epochs</th>
<th>Dropout</th>
<th>Gradient Clipping</th>
<th>BatchNorm</th>
<th>Layers</th>
<th>Hidden Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reddit</td>
<td>0.01</td>
<td>400</td>
<td>0.5</td>
<td>0.5</td>
<td>Yes</td>
<td>2</td>
<td>256</td>
</tr>
<tr>
<td>Yelp</td>
<td>0.01</td>
<td>500</td>
<td>0.1</td>
<td>0.5</td>
<td>Yes</td>
<td>2</td>
<td>512</td>
</tr>
<tr>
<td>ogbn-arxiv</td>
<td>0.01</td>
<td>500</td>
<td>0.5</td>
<td>0.5</td>
<td>Yes</td>
<td>3</td>
<td>128</td>
</tr>
<tr>
<td>ogbn-proteins</td>
<td>0.002</td>
<td>1000</td>
<td>0.5</td>
<td>0.0</td>
<td>No</td>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>ogbn-products</td>
<td>0.002</td>
<td>500</td>
<td>0.5</td>
<td>0.5</td>
<td>No</td>
<td>3</td>
<td>256</td>
</tr>
</tbody>
</table>

Table 8: Configuration of Full-Batch GCNII.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Learning Rates</th>
<th>Epochs</th>
<th>Dropout</th>
<th>Gradient Clipping</th>
<th>BatchNorm</th>
<th>Layers</th>
<th>Hidden Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reddit</td>
<td>0.01</td>
<td>400</td>
<td>0.5</td>
<td>0.5</td>
<td>Yes</td>
<td>4</td>
<td>256</td>
</tr>
<tr>
<td>Yelp</td>
<td>0.01</td>
<td>500</td>
<td>0.1</td>
<td>0.5</td>
<td>Yes</td>
<td>4</td>
<td>512</td>
</tr>
<tr>
<td>ogbn-proteins</td>
<td>0.01</td>
<td>1000</td>
<td>0.5</td>
<td>0.0</td>
<td>No</td>
<td>4</td>
<td>256</td>
</tr>
<tr>
<td>ogbn-arxiv</td>
<td>0.001</td>
<td>1000</td>
<td>0.1</td>
<td>0.1</td>
<td>Yes</td>
<td>16</td>
<td>256</td>
</tr>
</tbody>
</table>
Table 9: Training configuration of GraphSAINT.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>RandomWalk Sampler</th>
<th>Training</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Walk length</td>
<td>Roots</td>
<td>Learning Rates</td>
</tr>
<tr>
<td>Reddit</td>
<td>4</td>
<td>2000</td>
<td>0.01</td>
</tr>
<tr>
<td>Yelp</td>
<td>2</td>
<td>1250</td>
<td>0.01</td>
</tr>
<tr>
<td>ogbn-arxiv</td>
<td>3</td>
<td>10000</td>
<td>0.01</td>
</tr>
<tr>
<td>ogbn-proteins</td>
<td>3</td>
<td>10000</td>
<td>0.01</td>
</tr>
<tr>
<td>ogbn-products</td>
<td>3</td>
<td>20000</td>
<td>0.01</td>
</tr>
</tbody>
</table>