GSD-GNN: Generalizable and Scalable Algorithms for Decoupled Graph Neural Networks

Anonymous Author(s)

ABSTRACT

Graph Neural Networks (GNNs) have achieved remarkable performance in various applications, including social media analysis, computer vision, and natural language processing. Decoupled GNNs are a ubiquitous framework because of their high efficiency. However, existing decoupled GNNs suffer from the following several defects. (1) Their studies on GNN feature propagation are isolated, with each study emphasizing a user-specified propagation matrix. (2) They still have high computation costs to achieve provable performance on massive graphs with millions of nodes and billions of edges. (3) Their feature propagation steps are uniform, which makes it difficult for them to escape the dilemmas of over-smoothing. In this paper, we propose GSD-GNN, a **G**eneralized and **S**calable **D**ecoupled **GNN** framework based on the spectral graph theory, which offers the following advantages. Firstly, through minor parameter adjustments, it can degenerate into most existing Decoupled GNNs, such as APPNP, GDC, SGC, etc. Secondly, it efficiently computes an arbitrary propagation matrix with near-linear time complexity and theoretical guarantees. Thirdly, it customizes the adaptive feature propagation mechanism for each node to mitigate the over-smoothing dilemma. Finally, extensive experiments on massive graphs (up to billions of edges) demonstrate that the proposed GSD-GNN indeed is effective, scalable, and flexible.

ACM Reference Format:
Anonymous Author(s). 2018. GSD-GNN: Generalizable and Scalable Algorithms for Decoupled Graph Neural Networks. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Recently, the noteworthy success of Graph Neural Networks (GNNs) has propelled numerous tasks, including social media analysis [22, 29], recommender system [10, 33], natural language processing [23], and computer vision [14, 38]. Vanilla GNNs [9, 15] follow a seminal message-passing framework. Namely, each node maintains a representation vector (the initial representation vector is the corresponding feature vector) that iteratively updates through aggregating the representations of its neighbors. However, such a framework is full-batch training (i.e., all node representation vectors have to be stored in GPU), needing huge GPU memory space and high time overheads, resulting in poor scalability. Therefore,

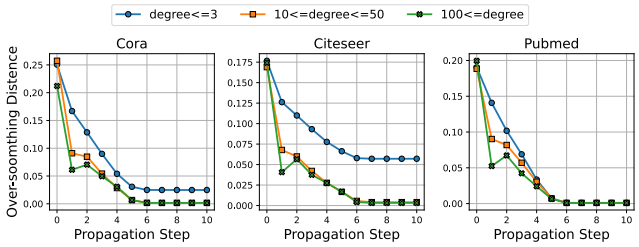

Figure 1: Nodes with higher degrees tend to faster over-smoothing compared to those with lower degrees.

how to apply vanilla GNNs to massive graphs (e.g., a graph with millions of nodes and billions of edges) has become a central research topic. There are two major lines of research have been proposed for improving scalability (Section 2): (1) Sampling-based approaches implement the message-passing only between the neighbors within a sampled mini-batch [5, 8, 9, 34, 35, 40]. (2) Decoupled approaches decouple the feature propagation and feature transformation by removing the intermediate activation function [3, 6, 16, 20, 30].

Although the above methods have substantially improved training time, they still suffer from several notable limitations. Specifically, sampling-based methods often need to discard a large number of neighboring nodes, leading to potential accuracy diminishment. For instance, GraphSAGE [9], a well-known method that aggregates a limited subset of neighboring nodes, usually between 10 to 25, per target node. As a result, GraphSAGE demonstrates an average accuracy that is 2% lower than the proposed GSD-GNN in our empirical results (Section 5). Decoupled approaches can effectively capture the overall structure of a graph, which alleviates the limitations of sampling-based methods. However, decoupled approaches often encounter poor scalability for achieving provable performance on massive graphs with millions of nodes and billions of edges. For example, PPRgo [3] uses the seminal Local Push sub-procedure [2] to accelerate the computation of *a single* Personalized PageRank vector for capturing higher-order neighborhood information. According to the latest reports [19], the fastest Local Push algorithm takes 2 seconds on a million-node graph. Thus, computing the whole PPR matrix (i.e., n Personalized PageRank vectors) with this algorithm requires over 23 days, making it infeasible even for powerful computing clusters. Besides, our empirical observations indicate that distinct nodes exhibit different convergence rates for over-smoothing (over-smoothing refers to nodes reaching an indistinguishable state). As shown in Figure 1, nodes with higher degrees tend to achieve over-smoothing at a faster rate, i.e., considering $\tilde{A}^l X$ is l-step feature propagation, the number of propagation steps l to reach over-smoothing is inversely proportional to the node's degree. The detailed experimental setup is deferred to Section 5.5. However, existing decoupled models maintain a *uniform* l for each node, which suffers from the following dilemma. A small number of propagation steps restricts access to information from distant

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/10.1145/1122445.1122456

nodes, while a larger number of steps results in convergence among different nodes' features, leading to indistinguishability. Thus, there is potential to enhance both the efficiency and flexibility of GNNs.

In this paper, we introduce GSD-GNN, a **G**eneralized and **S**calable **D**ecoupled **GNN** with low time complexity, high accuracy and flexible. GSD-GNN adopts the well-known decoupled GNN architecture, ensuring exceptional scalability. To address the issues of existing decoupled GNNs, GSD-GNN employs non-trivially the theories of the spectral sparsifier and random-walk matrix polynomials [7, 25] to present a generalized framework, which can approximate the feature propagation matrices of most well-known decoupled GNNs with minor parameter adjustments (Table 1). In particular, we derive an appropriate proximity-based generalized propagation matrix that maintains the influence of relevant nodes located multiple hops away. It is important to emphasize that this approach not only presents a generalized framework for computing arbitrary propagation matrices but also provides theoretical guarantees regarding approximation errors. On top of that, we propose path-sampling strategies to obtain the generalized propagation matrix with near-linear time complexity, which can greatly reduce the computational cost of traditional decoupled GNNs. On the other hand, to address the propagation step or over-smoothing dilemma mentioned above, we design an adaptive feature propagation mechanism to allow different nodes to adopt different propagation steps. Finally, we employ a multi-layer perception (MLP) to acquire predicted soft labels for unlabeled nodes. Thanks to the efficient and scalable training of the MLP, scaling GSD-GNN to massive graphs becomes feasible. In a nutshell, we highlight our main contributions as follows.

- We introduce a novel model of **G**eneralized and **S**calable **D**ecoupled **GNN** (GSD-GNN), which can generalize the feature propagation matrices of most decoupled GNNs. Besides, with the support of the theories of spectral sparsifier and random-walk matrix polynomials, we design near-linear time algorithms to obtain quality-guaranteed results.
- We present an adaptive feature propagation strategy that enables nodes to selectively aggregate multi-hop information from other nodes, which can alleviate the over-smoothing dilemma of exiting decoupled GNNs.
- Empirical results on seven real-life graphs and eleven competitors showcase that our GSD-GNN indeed is effective, scalable, and flexible on the semi-supervised node classification task. Our datasets and codes are obtained at [1].

2 RELATED WORK

Decoupled GNNs. Decoupled GNNs separate the feature propagation from the feature transformation, enabling deep feature propagation with shallow feature transformations. SGC [30] introduces a decoupling scheme that removes non-linearity from feature transformation. It directly propagates the features of neighbors within K hops, in which K is a hyper-parameter. After SGC, a wide array of models with decoupled architectures emerged. The Approximated Personalized Propagation of Neural Predictions (APPNP) [16] integrates multi-hop information by leveraging predefined weights that decay exponentially with hops. To bolster scalability, PPRGo [3] mitigates the computational burden by curtailing the number of aggregated neighbors. The Deep and Adaptive Graph Neural Network (DAGNN) [20] uses a learning mechanism to learn propagation adaptively. However, computing an effective propagation matrix on massive graphs is still a challenge for PPRGo and DAGNN. Chen et al. [6] introduce GBP, which integrates reverse push and random walk strategies within a generalized PageRank model to approximate feature propagation. However, GBP is to improve scalability at the expense of memory overheads. AGP [27] devises a unified graph propagation model, and employs a forward push-based algorithm and random sampling strategy to select subsets of unimportant neighborhoods. However, AGP introduces randomness and has no theoretical guarantee, resulting in inaccurate and even poor empirical performance in most cases, as stated in our experiments. So, compared to the original GNNs, which use a full-batch training strategy (i.e., each node's representation has to be stored in GPU memory), decoupled GNNs separate the propagation and transformation. Thus, decoupled GNNs allow for mini-batch training and improve the models' scalability.

Sampling-based GNNs. For the node-wise sampling methods, GraphSAGE [9] randomly selects a fixed-size set of neighbors within each mini-batch. VR-GCN [5] leverages historical activations to restrict the number of sampled nodes and reduce the variance of sampling. For the layer-wise sampling methods, FastGCN [4] employs independent node sampling in each layer according to the node's degree, and maintains the same sample size across all layers. LADIES [40] considers the layer constraint and introduces a layer-wise, neighbor-dependent, and importance sampling approach. For the graph sampling methods, Cluster-GCN [8] utilizes graph clustering techniques to divide the original graph into multiple sub-graphs. In each mini-batch, one sub-graph is sampled to perform feature propagation. Similarly, GraphSAINT [35] samples a certain amount of nodes and uses the induced sub-graph to perform feature propagation in each mini-batch.

3 PRELIMINARIES

3.1 Notations of GNNs

Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = n$ nodes and $|\mathcal{E}| = m$ edges, graph connectivity is given by the adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$, and node features are represented by $X \in \mathbb{R}^{n\times d}$ where d is the feature dimension. Let $\mathbf{D} = \text{diag}\,(d_1, d_2, \cdots, d_n) \in \mathbb{R}^{n\times n}$ be the degree matrix associated with $\mathbf{A}$, where $d_i = \sum_{v_j\in\mathcal{V}} \mathbf{A}_{ij}$ is the degree of node v_i. $L = D - A$ is the Laplacian matrix of G. Let $\mathcal{V}_l$ be the set of labeled vertices, our goal is to predict the labels of an unlabeled set $\mathcal{V}_u$ under the guidance of $\mathcal{V}_l$.

The prevalent paradigm of GNN is the Graph Convolutional Network (GCN) [15]. The graph convolution operation in GCN is formulated as follows,

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right), \quad l = 0, 1, \cdots \tag{1}$$

where $\mathbf{H}^{(l)}$ is the node representation matrix of the l-th layer and $\mathbf{H}^{(0)} = X$. $\tilde{\mathbf{A}} = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$ is the propagation matrix where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, $\hat{\mathbf{D}} = \mathbf{D} + \mathbf{I}$, and $\mathbf{I}$ is the identity matrix. $\mathbf{W}^{(l)}$ is the trainable weight matrix of the l-th layer and $\sigma(\cdot)$ is the activation function (e.g., Relu). Zhang et al. [36] pointed out that the graph convolution operation can be disentangled into the following two

Table 1: A comparison of the state-of-the-art feature propagation methods.

Method	Weighting Coefficient	Transition Matrix	Propagation Equation
Personalized PageRank [21]	$\alpha(1-\alpha)^k$	$\mathbf{AD}^{-1}$	$\sum_{k=0}^{d}\alpha(1-\alpha)^k(\mathbf{AD}^{-1})^k\mathbf{X}$
Heat kernel [18]	$e^{-t}\frac{t^k}{k!}$	$\mathbf{AD}^{-1}$	$\sum_{k=0}^{d}e^{-t}\frac{t^k}{k!}(\mathbf{AD}^{-1})^k\mathbf{X}$
Katz [13]	α^k	$\mathbf{A}$	$\sum_{k=0}^{d}\alpha^k\mathbf{A}^k\mathbf{X}$
SGC [30]	$\theta_d=1, others=0$	$\mathbf{D}^{-1/2}\mathbf{AD}^{-1/2}$	$(\mathbf{D}^{-1/2}\mathbf{AD}^{-1/2})^d\mathbf{X}$
S^2GC[39]	$1/d$	$\mathbf{D}^{-1/2}\mathbf{AD}^{-1/2}$	$\sum_{k=0}^{d}\frac{1}{d}(\mathbf{D}^{-1/2}\mathbf{AD}^{-1/2})^k\mathbf{X}$
APPNP [16] & PPRgo [3] & GBP [6]	$\alpha(1-\alpha)^k$	$\mathbf{D}^{-1/2}\mathbf{AD}^{-1/2}$	$\sum_{k=0}^{d}\alpha(1-\alpha)^k(\mathbf{D}^{-1/2}\mathbf{AD}^{-1/2})^k\mathbf{X}$
GDC [17]	$e^{-t}\frac{t^k}{k!}$	$\mathbf{D}^{-1/2}\mathbf{AD}^{-1/2}$	$\sum_{k=0}^{d}e^{-t}\frac{t^k}{k!}(\mathbf{D}^{-1/2}\mathbf{AD}^{-1/2})^k\mathbf{X}$
GSD-GNN (this paper)	$\frac{\omega^k}{(k!)^\rho\cdot C}$	$\mathbf{D}^{r-1}\mathbf{AD}^{-r}$	$\sum_{k=0}^{d}\frac{\omega^k}{(k!)^\rho\cdot C}(\mathbf{D}^{r-1}\mathbf{AD}^{-r})^k\mathbf{X}$

consecutive operations P and T.

$$\begin{aligned}\text{Feature Propagation }(\mathbf{H}) &= P(\mathbf{H}) = \tilde{\mathbf{A}}\mathbf{H}\\ \text{Feature Transformation }(\mathbf{H}) &= T(\mathbf{H}) = \sigma(\mathbf{HW})\end{aligned} \tag{2}$$

Evidently, in the context of GCN, the operations P and T are inherently intertwined, i.e., every P operation is followed by a corresponding T operation (*PTPT* for short), leading to the poor model scalability. To enhance the scalability, researchers have opted to decouple T and P operations, resulting in the derivation of two distinct architectures, namely *TTPP* [3, 16] and *PPTT* [30]. Notably, although the *TTPP* architecture gains flexibility from disentangling the graph convolution operation, it is less scalable than the *PPTT* architecture as the entanglement of stacked P operations within the training process limits its efficiency.

3.2 The Spectral Sparsifier Theory

Here, we overview the theoretical foundations of Random-Walk Matrix-Polynomials and the corresponding Spectral Sparsifiers [7].

Definition 1 (Random-Walk Matrix-Polynomials (RWMP)). *Let $\mathbf{A}$ and $\mathbf{D}$ be the adjacency matrix and degree matrix of the graph G, respectively. For a non-negative vector $\boldsymbol{\alpha}=(\alpha_1,...,\alpha_d)$ with $\sum_{i=1}^{d}\alpha_i=1$, the following matrix*

$$\mathbf{L}_{\boldsymbol{\alpha}}(G)=\mathbf{D}-\sum_{k=1}^{d}\alpha_k\mathbf{D}\cdot\left(\mathbf{D}^{-1}\mathbf{A}\right)^k \tag{3}$$

is a d-degree random-walk matrix-polynomial of G and $\mathbf{L}_{\boldsymbol{\alpha}}(G)$ is a Laplacian matrix [7].

Lemma 1 (Spectral Sparsifiers of RWMP [7]). *For any $\epsilon>0$, there exists a Laplacian matrix $\tilde{\mathbf{L}}$ with $O(n\log(n/\epsilon^2))$ non-zeros s.t.*

$$(1-\epsilon)\cdot\boldsymbol{x}^\top\tilde{\mathbf{L}}\boldsymbol{x}\le\boldsymbol{x}^\top\mathbf{L}_{\boldsymbol{\alpha}}(G)\boldsymbol{x}\le(1+\epsilon)\cdot\boldsymbol{x}^\top\tilde{\mathbf{L}}\boldsymbol{x} \tag{4}$$

holds for arbitrary $\boldsymbol{x}\in\mathbb{R}^n$.

The Laplacian matrix $\tilde{\mathbf{L}}$ satisfying Eq. (4) is referred to as being spectrally similar to $\mathbf{L}_{\boldsymbol{\alpha}}(G)$ with an approximation parameter ϵ [24, 25], and such a spectral sparsifier $\tilde{\mathbf{L}}$ can be constructed in time $O(d^2m\log^2(n/\epsilon^2))$ [7]. Specifically, an initial sparsifier with $O(dmlog(n/\epsilon^2))$ non-zeros is generated from $L_{\boldsymbol{\alpha}}(G)$. Then, standard spectral sparsification algorithms [24, 25] are applied to the initial sparsifier to further reduce the number of non-zero entries.

4 THE PROPOSED SOLUTIONS

We introduce our novel model of **G**eneralized and **S**calable **D**ecoupled **GNN** (GSD-GNN). Specifically, GSD-GNN first applies the spectral graph theory to obtain the generalized propagation matrix with near-linear time and theoretical error guarantees (Sections 4.1-4.2). Then, GSD-GNN devises node-wise adaptive propagation strategies to further address the over-smoothing dilemma in the vanilla GNN model (Section 4.3). Figure 2 depicts the framework of GSD-GNN.

4.1 Generalized Propagation Matrix

The decoupled GNN model comprises various propagation methods. Table 1 presents the propagation details employed by different decoupled GNNs. We define graph generalized propagation, comprising two parts, the **weighting coefficients**, and the **transition matrix**, to model various propagation techniques and GNN propagation formulas in a unified framework.

Weighting coefficients. Different weight coefficients offer distinct advantages. For example, the coefficients of PPR [21] can sharply decrease in short intervals, emphasizing local nodes. On the other hand, the coefficients of heat kernel [18] can peak at specific hops, prioritizing nodes at specific distances. In pursuit of generality, we adopted the weighting coefficient introduced in [12], which integrates the advantages of different weighting coefficients like PPR and heat kernel. Such a weighting coefficient is defined as

$$\theta_k=\frac{\omega^k}{(k!)^\rho\cdot C}\text{ and }C=\sum_{k=0}^{\infty}\frac{\omega^k}{(k!)^\rho} \tag{5}$$

where ω,ρ are two hyper-parameters. Eq. (5) enables us to approximate different weighting coefficients by varying the hyper-parameters ω and ρ. For instance, Eq. (5) transforms to a heat kernel when $\rho=1$ and transforms to PPR when $\rho=0$. By tuning ω and ρ in Eq. (5), we can find an appropriate weighting coefficient for each specific graph, instead of relying on a fixed weighting coefficient.

Transition matrix. We identify the generalized form of the transition matrix as $\mathbf{D}^{r-1}\mathbf{AD}^{-r}$. By manipulating the parameter r, we can flexibly assign preferences to source and destination nodes, instead of enforcing equality between source and destination nodes.

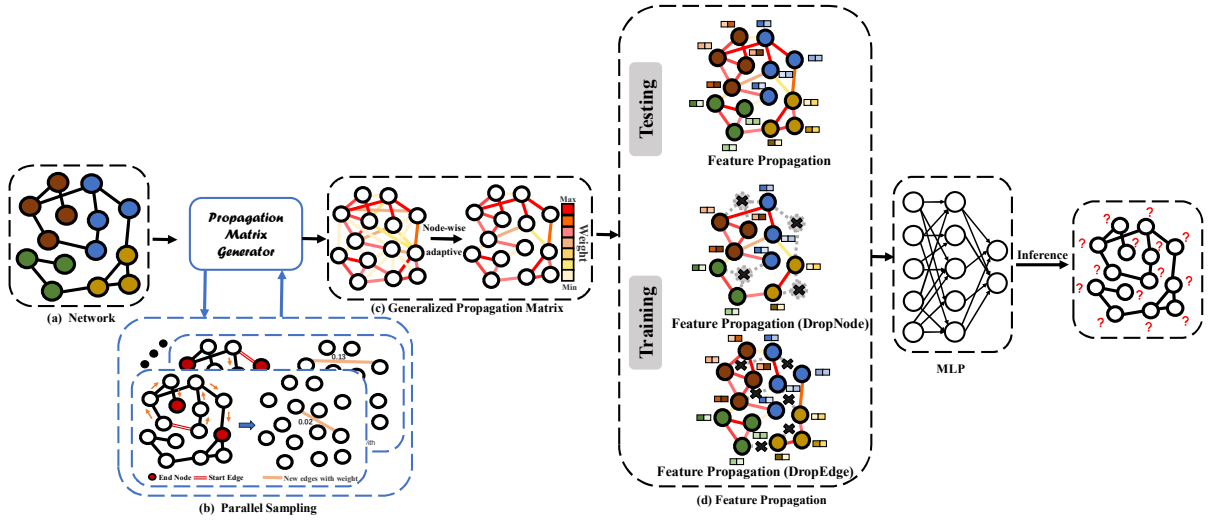

Figure 2: Our model GSD-GNN adopts the following four steps. Step 1: Random walks are initiated by selecting an edge and terminated after completing sampling for a specific path length. A new edge weight is then established between the stopped nodes. Repeating the above steps procedures until reaching the predefined number of sampling edges as shown in Fig (b). Step 2: Calculate the generalized propagation matrix and apply node-wise adaptive strategy Fig (c). For a graph, these processes are performed only once. Step 3: In the feature propagation stage, features propagate on the generalized propagation matrix, integrating the drop technique in the training phase Fig (d). Step 4: Feature transformation is achieved through an MLP.

Combining the weighting coefficient and transition matrix, the generalized propagation matrix can be defined as follows,

$$\mathbf{\Pi} = \sum_{k=0}^{\infty} \theta_k \cdot (\mathbf{D}^{r-1}\mathbf{A}\mathbf{D}^{-r})^k = \sum_{k=0}^{\infty} \frac{\omega^k}{(k!)^\rho \cdot C} \cdot (\mathbf{D}^{r-1}\mathbf{A}\mathbf{D}^{-r})^k \quad (6)$$

By setting different hyper-parameters, we can tailor Eq. (6) to match the propagation equation of **any** GNN algorithm in Table 1. For instance, when $r = 1/2$ and $\rho = 0$, the generalized propagation matrix degenerates to the propagation equation used in APPNP [16], PPRgo [3], and GBP [6].

4.2 From RWMP to Generalized Propagation

Computing the generalized propagation matrix of Eq. (6) using traditional matrix multiplication becomes increasingly time-consuming as the propagation step increases, especially for massive graphs. To overcome this challenge, we explore the following mathematical connections between Random-Walk Matrix-Polynomials (RWMP) and the generalized propagation matrix.

Theorem 1 (RWMP for Generalized Propagation). *For any graph G and its d-degree RWMP $\mathbf{L}_{\boldsymbol{\alpha}}(G)$ in Definition 1, the generalized Propagation matrix can be represented as,*

$$\mathbf{\Pi} = \mathbb{C} \cdot (\mathbf{I} - \mathbf{D}^{r-1}\mathbf{L}_{\boldsymbol{\alpha}}(G)\mathbf{D}^{-r}) + \theta_0 \cdot \mathbf{I} \quad (7)$$

where $\mathbb{C} = \sum_{k=1}^{\infty} \theta_k$ and $\mathbf{I}$ is an identity matrix.

Proof. Firstly, we have

$$\begin{aligned}
\left(\mathbf{D}^{\mathbf{r}-\mathbf{1}}\mathbf{A}\mathbf{D}^{-\mathbf{r}}\right)^k &= \mathbf{D}^{\mathbf{r}-\mathbf{1}}\left(\mathbf{A}\mathbf{D}^{-\mathbf{1}}\right)^k\mathbf{D}^{\mathbf{1}-\mathbf{r}} \\
&= \mathbf{D}^{\mathbf{r}-\mathbf{1}}\left(\mathbf{A}\mathbf{D}^{-\mathbf{1}}\right)\ldots\left(\mathbf{A}\mathbf{D}^{-\mathbf{1}}\right)\mathbf{D}^{\mathbf{1}-\mathbf{r}} \\
&= \mathbf{D}^{\mathbf{r}-\mathbf{1}}\mathbf{A}\left(\mathbf{D}^{-\mathbf{1}}\mathbf{A}\right)\ldots\left(\mathbf{D}^{-\mathbf{1}}\mathbf{A}\right)\mathbf{D}^{-\mathbf{1}}\mathbf{D}^{\mathbf{1}-\mathbf{r}} \\
&= \mathbf{D}^{\mathbf{r}-\mathbf{1}}\mathbf{A}\left(\mathbf{D}^{-\mathbf{1}}\mathbf{A}\right)^{k-1}\mathbf{D}^{-\mathbf{r}}
\end{aligned}$$

Subsequently, we observe the following transformation for Eq. (3).

$$\begin{aligned}
\mathbf{L}_{\boldsymbol{\alpha}}(G) &= \mathbf{D} - \sum_{k=1}^{\infty}\alpha_k\mathbf{D}\cdot\left(\mathbf{D}^{-1}\mathbf{A}\right)^k \\
&= \mathbf{D} - \sum_{k=1}^{\infty}\alpha_k\mathbf{D}\cdot\left(\mathbf{D}^{-1}\mathbf{A}\right)\left(\mathbf{D}^{-1}\mathbf{A}\right)^{k-1} \\
&= \mathbf{D} - \sum_{k=1}^{\infty}\alpha_k\mathbf{A}\left(\mathbf{D}^{-1}\mathbf{A}\right)^{k-1}
\end{aligned}$$

Finally, we can unify the form

$$\begin{aligned}
\mathbf{D}^{\mathbf{r}-\mathbf{1}}\mathbf{L}_{\boldsymbol{\alpha}}(G)\mathbf{D}^{-\mathbf{r}} &= \boldsymbol{I} - \sum_{k=1}^{\infty}\alpha_k\mathbf{D}^{\mathbf{r}-\mathbf{1}}\mathbf{A}\left(\mathbf{D}^{-\mathbf{1}}\mathbf{A}\right)^{k-1}\mathbf{D}^{-\mathbf{r}} \\
&= \boldsymbol{I} - \sum_{k=1}^{\infty}\alpha_k\left(\mathbf{D}^{\mathbf{r}-\mathbf{1}}\mathbf{A}\mathbf{D}^{-\mathbf{r}}\right)^k
\end{aligned}$$

Note that RWMP is a series starting at 1 and $\sum_{i=1}^{\infty}\alpha_i = 1$, while the Generalized Propagation Matrix is a series starting at 0. Thus, we have $\mathbf{\Pi} = \theta_0 I + \sum_{k=1}^{\infty}\theta_k(\mathbf{D}^{\mathbf{r}-\mathbf{1}}\mathbf{A}\mathbf{D}^{-\mathbf{r}})^k$. Let $\mathbb{C} = \sum_{k=1}^{\infty}\theta_k$, we can obtain that $\frac{\mathbf{\Pi}}{\mathbb{C}} = \frac{\theta_0}{\mathbb{C}}I + \sum_{k=1}^{\infty}\frac{\theta_k}{\mathbb{C}}(\mathbf{D}^{\mathbf{r}-\mathbf{1}}\mathbf{A}\mathbf{D}^{-\mathbf{r}})^k$. Furthermore,

Algorithm 1 *Path_Sampling*

Input: $e(u,v)$ and x // x is length of path.
Output: n_0,n_x,Z_p

1: Uniformly pick an integer $j \in [1,x]$
2: Perform $(j-1)$-step random walk from u to n_0
3: Perform $(x-j)$-step random walk from v to n_x
4: Keep track of Z_p between n_0 and n_x during process Z_p will add as edge weight to our sparsifier
5: **return** n_0,n_x,Z_p

Algorithm 2 *Generalized Propagation matrix generator*

Input: An undirected graph $G(V,E)$, The number of non-zeros M, Truncation length d, Hyperparameters ω, ρ, r.
Output: Generalized Propagation Matrix $\widetilde{\Pi}$

1: Initialize a graph $\tilde{\mathbf{G}}(V,\emptyset)$ and calculate weighting coefficient using ω and ρ
2: **for** $i = 1$ to M **do**
3: Uniformly pick an edge e=(u,v)
4: pick an integer $x \in [1,d]$ with corresponding probability
5: $u_{new}, v_{new}, Z_p \leftarrow$ Path_Sampling(e,x)
6: Add an edge (u_{new}, v_{new}) with weight $2xm/MZ_p$ to $\tilde{\mathbf{G}}$
7: $\tilde{\mathbf{L}} \leftarrow$ the unnormalized graph Laplacian of $\tilde{\mathbf{G}}$
8: calculate the generalized propagation matrix $\widetilde{\Pi}$ by Eq. 7 with r
9: **return** $\widetilde{\Pi}$

let $\frac{\theta_k}{\mathbb{C}} = \alpha_k$, we have $\mathbf{\Pi} = \mathbb{C}(\boldsymbol{I} - \mathbf{D^{r-1}L_{\boldsymbol{\alpha}}}(G)\mathbf{D^{-r}}) + \theta_0 \boldsymbol{I}$. Thus, Theorem 1 is true. □

By Theorem 1, a random-walk matrix-polynomial can be used to obtain any propagation matrix. Therefore, the next step is to consider how to obtain a fast and theoretically guaranteed random-walk matrix-polynomial (RWMP). Thanks to Lemma 1, we can find a sparsifier $\widetilde{L}$ to approximate the random-walk matrix-polynomial $\mathbf{L_{\boldsymbol{\alpha}}}(G)$, providing a dual theoretical guarantee of time and precision.

Path sampling. To elucidate the sparsifier construction, we introduce the sampling method detailed in the referenced work [7]. As outlined in Algorithm 1, the sampling process begins with the selection of an initial edge. Subsequently, random walks are conducted from the two nodes of the edge with a specified step size (Algorithm 1, Lines 1-3). Afterward, a new edge is generated with an assigned weight Z_p(Algorithm 1, Line 4). This process continues until the desired number of edges is reached. In Algorithm 1, for each length-x path$[n_1, ..., n_x]$ and $Z_p = \sum_i^x 2/A(n_{i-1}, n_i)$.

Generalized Propagation Matrix Generator. As depicted in Algorithm 2, the construction of the sparsifier $\widetilde{L}$ commences by initializing a vertex set identical to the original graph G (Algorithm 2, Line 1). Subsequently, we utilize the edge sampling method to generate a new graph $\tilde{G}$. After obtaining the required number of edges through sampling (Algorithm 2, Lines 2-7), we compute the Laplacian matrix to serve as the sparsifier $\widetilde{L}$. The path length r is selected based on the probability $\alpha_{\mathbf{k}} = \frac{\theta_k}{\mathbb{C}}$, i.e., r is chosen based on the probabilities of $\frac{\theta_1}{\mathbb{C}}$, $\frac{\theta_2}{\mathbb{C}}$..., starting from 1 and incrementing accordingly. Lastly, compute the generalized propagation matrix Π according to Theorem 1 (Algorithm 2, Lines 8-9).

4.3 Our Algorithm and Theoretical Analysis

Similar to other decoupled GNNs, GSD-GNN decomposes the training process into two distinct phases: feature propagation and feature transformation. Due to GSD-GNN's employment of the PPTT architecture, feature propagation is preprocessed only once. Moreover, the MLP used in the feature transformation phase is efficient and scalable. These advantages ensure that GSD-GNN maintains high efficiency in both the feature propagation and feature transformation phases, making it scalable to large graphs.

To decouple the original GNN, we remove the nonlinear activation σ. We construct a d-step feature propagation as follows:

$$\mathbf{H} = \mathbf{\Pi}'\mathbf{X} = \sum_{k=0}^{d} \theta_k \left(\mathbf{D}^{r-1}\mathbf{A}\mathbf{D}^{-r}\right)^k \cdot \mathbf{X} \tag{8}$$

where $\theta_k = \frac{\omega^k}{(k!)^\rho \cdot C}$. This constitutes a generalized d-order propagation formula. Eq. (8) stands as a comprehensive generalization applicable to most decoupled GNNs. By adjusting ρ and r, we can specialize Eq. (8) to yield various GNN models. Thanks to the swift and effective calculation of Π' using Algorithm 2, we obtain a high-performance generalized method for the feature propagation.

Node-wise adaptive feature propagation strategies. As observed in Figure 1, as d begins to increase, nodes gradually converge towards over-smoothing at varying rates. In practice, a larger d is often set to acquire more global information. Based on the previous analysis, we can obtain the generalized propagation matrix $\tilde{\Pi}$ equivalent to $\sum_{k=0}^{d} \theta_k \tilde{\mathbf{A}}^k$ through Algorithm 2. Therefore, our algorithm may still be affected by over-smoothing issues. To solve this problem, we introduce the following strategy.

Definition 2 (Node-wise adaptive strategy for propagation). *We consider selecting the unique top-k similar nodes for each node for information aggregation based on each node's degree. For a specific node i, our top-k selection strategy is as follows:*

$$\mathbb{K}_i = Max(\mathbb{N}\log(\frac{d_{min}}{d_i} + 1), d_{avg}) \tag{9}$$

where d_{min} and d_i are the minimum node degree and degree of node i, respectively. d_{avg} is average degree, $\mathbb{N}$ represents a hyperparameter.

Our devised strategy allows nodes with higher degrees to aggregate fewer but more significant features compared to nodes with lower degrees. This strategy yields two notable advantages: (i) by adjusting the number of aggregated nodes inversely proportional to the degree, it ensures sufficient information for nodes with lower degrees while avoiding over-smoothing in higher-degree nodes; (ii) enhancing the generalized propagation matrix's sparsity accelerates feature propagation, contributing to the model's scalability. Empirical experiments further suggest that setting $\mathbb{N}$ to 512 generally results in satisfactory performance across diverse datasets.

After feature propagation, embedding matrix H is utilized as the input feature to the MLP. Each layer of MLP is associated with a learned weight matrix W. In a nutshell, we summarize the feature transformation in the following scheme:

$$\boldsymbol{Y} = \mathbf{MLP}\,(\boldsymbol{H}) \tag{10}$$

Improving generalization ability. PMLP[32] highlights that the superior performance of GNN compared to MLP primarily results

Algorithm 3 Generalized and Scalable Decoupled GNN

Input: Adjacency matrix $\tilde{A} \in \mathbb{R}^{n \times n}$, feature matrix $X \in \mathbb{R}^{n \times F}$, number of perturbations J, MLP: $f_{mlp}(X, \Theta)$, epochs N_{epoch}
Output: Prediction $\hat{Y}$.

1: $\tilde{\Pi} \leftarrow$ propagation matrix generator ($\tilde{A}$) and node-wise adaptive.
2: **for** $i = 1$ to N_{epoch} **do**
3: **if** Training **then**
4: **for** $j = 1$ to J **do**
5: Uniformly pick DropNode() or DropEdge();
6: Feature propagation: $\tilde{X}^J = \text{Drop}(\tilde{\Pi}X)$;
7: Feature transformation: $Y^J = f_{mlp}(\tilde{X}^J, \Theta)$;
8: Compute loss $\mathcal{L}_{CE}$ via Eq.12, update the parameters Θ.
9: **else**
10: Feature propagation: $\tilde{X} = \tilde{\Pi}X$;
11: Feature transformation: $\hat{Y} = f_{mlp}(\tilde{X}, \Theta)$;
12: **return** $\hat{Y}$

from the model's stronger generalization abilities. This improvement in generalization is largely attributed to the efficacy of information propagation operations. To enhance the model's generalization, we introduce the well-known dropout techniques during training as follows:

$$\begin{aligned} &\textbf{Training: } \hat{X}_i = \mathcal{D}orpNode(\mathbf{X})_i, \hat{\Pi}_i = \mathcal{D}orpEdge(\tilde{\mathbf{\Pi}})_i \\ &\hat{Y}^{(i)} = \textbf{MLP}(\tilde{\Pi}\hat{X}_i) \text{ or } \textbf{MLP}(\hat{\Pi}_i X) \end{aligned} \tag{11}$$

where $\hat{Y}^{(i)}$ represents the predicted softmax outputs generated by the MLP after the i-th perturbation. Therefore, We adopt the Cross-Entropy (CE) measurement between the predicted softmax outputs and the one-hot ground-truth label distributions as the objective function as follows:

$$\mathcal{L}_{CE} = -\frac{1}{J}\sum_{i=0}^{J}\sum_{j \in \mathcal{V}_{tr}} \mathrm{Y}_j^{\mathrm{T}} \log\left(\hat{\mathrm{Y}}_j^{(i)}\right) \tag{12}$$

$\mathcal{V}_{tr}$ are nodes in the training set. J represents the number of perturbations, where each can encompass drop edge/node.

Workflow of GSD-GNN. As depicted in Algorithm 3, the input graph is preprocessed by GSD-GNN, following Algorithm 1 and Algorithm 2 with the node-wise adaptive propagation strategy applied (Algorithm 3 Line 1). During the training phase, the graph undergoes perturbation J times, with each perturbation involving the uniform selection of a DropEdge or DropNode for each epoch (Algorithm 3 Lines 4-5). The perturbed graph is subsequently input into the MLP for inference, wherein the loss function is computed, and the parameters are updated (Algorithm 3 Lines 7-8). The non-training phase carries out feature propagation, transformation, and inference without any perturbations (Algorithm 3 Lines 10-12).

Next, we analyze the time&space complexities of the proposed *GSD-GNN* and the approximation errors. The formal proofs of these theorems are deferred to our Full Paper [1] due to the space limit.

Theorem 2. *The time complexity and space complexity of Algorithm 3 are $O(nd^2 + M(1+d+\mathbb{E}(x)))$ and $O(M+m+n)$, respectively.*

Table 2: Statistics of Datasets.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	#Features	#Classes
Cora	2,708	5,429	1,433	7
Citeseer	3,327	4,732	3703	6
Pubmed	19,717	44,338	500	3
Flickr	89,250	899,756	500	7
AMiner-CS	593,486	6.2 M	100	18
Reddit	232,965	114 M	602	41
Papers100M	111 M	1.6 B	128	172

We define the approximation error as $\|\Pi - \tilde{\Pi}\|_F$. The error stems from two main factors: truncation and approximation. This situation can be reformulated as $\|\Pi - \Pi' + \Pi' - \tilde{\Pi}\|_F$. According to the triangle inequality, $\|\Pi - \Pi' + \Pi' - \tilde{\Pi}\|_F \leq \|\Pi - \Pi'\|_F + \|\Pi' - \tilde{\Pi}\|_F = \Delta\mathcal{L}_1 + \Delta\mathcal{L}_2$ holds true, where $\Delta\mathcal{L}_1$ and $\Delta\mathcal{L}_2$ represent truncation error and approximation error, respectively.

Theorem 3. *For error $\Delta\mathcal{L}_1$ caused by truncation, the $\Delta\mathcal{L}_1 \leq \frac{1}{C}\sqrt{n}\frac{\omega^{T+1}}{((T+1)!)^{\rho}}$, where n is the number of nodes.*

Theorem 4. *For error $\Delta\mathcal{L}_2$ caused by spectral approximation, the $\Delta\mathcal{L}_2 \leq 4\epsilon \cdot \mathbb{C}\sqrt{n}$, where n is the number of nodes and ϵ is approximation parameter.*

Combining Theorem 3 and Theorem 4, the resulting overall error can be expressed as follows.

$$\|\Pi - \tilde{\Pi}\|_F \leq \Delta\mathcal{L}_1 + \Delta\mathcal{L}_2 \tag{13}$$

5 EMPIRICAL STUDY

In this section, we evaluate our GSD-GNN's performance on the semi-supervised node classification. In particular, we report the effectiveness (i.e., mean accuracy) and efficiency (i.e., running time).

5.1 Experimental Settings

Datasets. We evaluate our solutions on seven publicly available real-world datasets (Table 2), which are widely used benchmarks for evaluating GNN's performance. Specifically, we conduct transductive learning on five citation networks (i.e., Cora, Citeseer, Pubmed, AMiner-CS, and Papers100M) and inductive learning on two social networks (i.e., Flickr and Reddit).

Competitors. We compare our proposed solutions against eleven competitors, which are summarized into three groups as follows.

- The coupled GNNs: GCN [15], GAT [26] and JK-Net [31];
- The decoupled GNNs: APPNP [16], AGP [27], NDLS [37], SGC [30], GBP [6]and PPRGo [3];
- Sampling-based GNNs: GraphSAGE [9], GraphSAINT [35].

Implementations. To alleviate the influence of randomness, we repeat each method five times and report their mean performance. The hyper-parameters of baselines are set as default parameters of the corresponding paper and our hyper-parameters are deferred to our Full Paper [1] due to the space limit. We employ Python, leveraging the DGL [28] library for most parts of the implementation, except for the high-performance implementation of obtaining the propagation matrix, which is implemented in C++. All experiments are conducted on an Ubuntu 22.04.3 machine with an NVIDIA RTX4000 GPU, Intel Xeon(R) CPU (2.20GHz), and 320GB RAM.

Table 3: Mean Accuracy (%) of transductive learning. "OOM" means "out of memory".

Type	Method	Cora	Citeseer	Pubmed	AMiner-CS	Papers100M
Coupled	GCN	81.5± 0.4	70.3± 0.5	79.1± 0.6	49.8± 0.6	OOM
	GAT	82.1 ± 0.7	71.0± 0.4	78.1± 0.7	49.2± 0.8	OOM
	JK-Net	81.2± 0.2	70.7± 0.5	78.4± 0.5	49.6± 0.5	OOM
Decoupled	APPNP	83.0 ± 0.3	71.5± 0.4	80.1± 0.4	51.6± 0.6	OOM
	NDLS	83.1± 0.4	71.2 ± 0.4	80.6 ± 0.6	53.5± 0.8	**65.1± 0.2**
	SGC	81.0± 0.2	71.8± 0.2	79.0± 0.1	51.2± 0.3	63.4± 0.2
	GBP	83.3 ± 0.3	72.0± 0.4	79.4 ± 0.3	52.7± 0.7	64.9± 0.5
	AGP	81.2 ± 0.5	70.1± 0.6	78.8 ± 0.6	51.6± 0.8	61.9± 0.9
	PPRGo	82.4 ± 0.3	71.3 ± 0.3	80.0 ± 0.4	51.7± 0.8	OOM
Sampled	GraphSAGE	80.3± 0.6	71.2± 0.4	77.8± 0.7	51.1± 0.8	OOM
	GraphSAINT	81.3± 0.4	70.8± 0.4	78.5± 0.6	51.8± 0.7	OOM
This paper	**GSD-GNN**	**84.1±0.6**	**73.1±0.4**	**81.2±0.6**	**53.6± 0.5**	64.6±0.5

5.2 Semi-supervised Node Classification

Table 3 shows the results for the semi-supervised transductive node classification task on the four standard graphs Cora, Citeseer, Pubmed, and Papers100M. In the transductive setting, GSD-GNN exceeds the best GNN model among all considered baselines on Cora, Citeseer, Pubmed, and AMiner-CS by a margin of 0.8% to 2.4%. Drop operations were not utilized in the Papers100M billion-scale graph dataset. This decision was made to avoid both the consumption of computation time and memory that would result from using drop operations in such a large-scale graph and saving a sort of propagation matrixes after multiple drop operations. Canceling the Drop operation can indeed impact the generalization performance of MLP. This is demonstrated by GSD-GNN achieving the best performance in all datasets except for dataset Papers100M. However, with only 0.5% less performance than the best NDLS in the baseline, GSD-GNN still achieved competitive performance. During the training phase of inductive learning, it is not permitted to use information from test and validation nodes. Therefore, the training and testing graphs are different and require separate graph precomputation and propagation. The given original node splittings follow previous work[9, 35]. Table 3 shows that several GNN models struggle with the Paper100M dataset, particularly the coupled GNN model. We attribute this difficulty to two primary factors: the substantial size of the dataset and the limited scalability inherent in the architecture of decoupled models. Coupled models use a full-batch training process, which can rapidly deplete GPU memory. During the experiments, it was observed that GCN faced memory exhaustion issues on the AMiner-CS, which is smaller than Paper100M. It only ran successfully after optimizing the data types employed in GCN. Table 4 reports the mean accuracy of inductive learning. Note that certain GNNs are deemed unsuitable for application in inductive testing, hence their exclusion from our analysis — aligning with the approach adopted in prior works[6, 37]. The reasons for their exclusion can be found in the supplementary material. GSD-GNN outperforms the most competitive baseline SGC by a margin of 2.1% on Reddit and ranks first with NDLS on the Flick, surpassing GBP by 0.4%, demonstrating competitiveness.

Table 4: Mean Accuracy (%) of inductive learning.

Method	Reddit	Flickr
SGC	92.4± 0.5	50.1± 0.3
GBP	89.2± 0.2	50.8± 0.2
PPRGo	91.3± 0.2	49.7± 0.1
NDLS	90.4.± 0.5	**51.2± 0.4**
GraphSAGE	91.5± 0.3	50.3± 0.8
GSD-GNN	**94.4±0.4**	**51.2 ± 0.3**

5.3 Efficiency Comparison

Considering the same sizes of the Cora and Citeseer datasets and the limitations of some GNNs in handling the Papers 100M dataset, we aim to assess efficiency across five real-world graph datasets: Cora, Pubmed, Flickr, AMiner-CS, and Reddit. Figure 3 illustrates the efficiency of each method. We computed the propagation matrices for all methods on the CPU to ensure fairness and efficiency in comparing the computation time for the propagation matrix. We record the time during the preprocessing of the Decoupled-based model and training time, excluding the time needed to load and convert data formats. SCG and GraphSAGE are faster than the other four models on all five datasets, but they sacrifice accuracy compared to GSD-GNN. For instance, on the Aminer dataset, GSD-GNN ranks third (181s), while SCG and GraphSAGE rank first (44s) and second (59s), respectively. However, regarding accuracy, SCG and GraphSAGE score 2.4% and 2.5% lower than GSD-GNN, respectively. The speed of GSD-GNN's approximation of the propagation matrix theoretically depends solely on the number of sampled edges. Therefore, while ensuring accuracy, a larger average degree is less favorable to GSD-GNN. As a result, GSD-GNN is less efficient on the Reddit dataset than SCG, GraphSAGE, and GBP. However, this decrease in efficiency is compensated by an improvement in accuracy of 2.1%,2.9% and 5.3%, respectively.

5.4 Parameter Analysis

We discuss how the hyper-parameters influence the performance of GSD-GNN. We report all the parameter analyses on the Cora

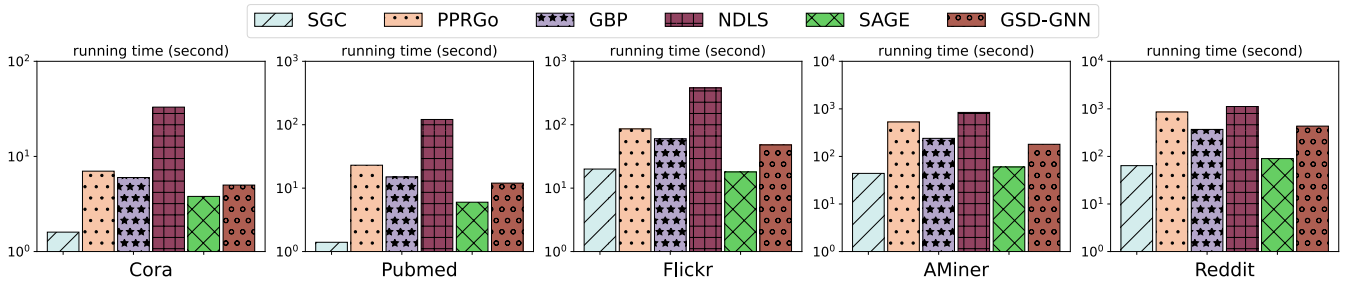

Figure 3: Efficiency comparison of different methods.

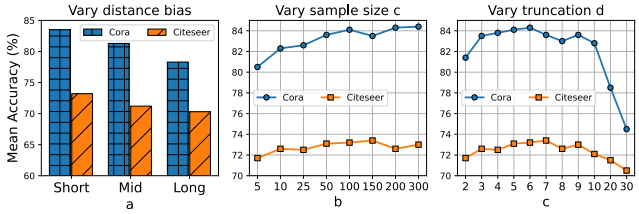

Figure 4: Node classification performance of GSD-GNN with varying parameters.

and Citeseer. We study the effect of coefficient factor ρ and ω, truncation length d, and The number of non-zeros M. With a maximum sampling step size of 50, we adjusted ρ and ω to divide the sampling steps tendency into three bias categories: short-distance ($d < 6$), medium-distance ($10 < d < 20$), and long-distance ($30 < d$). Figure 4.a indicate that the short-distance preference yields better performance. We set the sample size to a multiple of the number of edges $M = cm$. Figure 4.b illustrates that GSD-GNN can have a competitive effect at $c = 5$. However, further improvement is limited due to the diminishing marginal utility of increasing the number of sampling edges. Figure 4.c shows that a single expansion of the sampling step may cause GSD-GNN to focus more on distant nodes, thus ignoring the more important nearby nodes. This can lead to a decrease in the effectiveness of the experiments.

5.5 Node-wise Adaptive Propagation Analysis

Firstly, we introduce our setup for the over-smoothing experiment. $X^l = \tilde{A}^l X$ is the node feature l step feature propagation, and the feature that continuously smooths the node feature through infinite propagation is X^∞, where $X^\infty = \tilde{A}^\infty X$, $\tilde{A}^\infty_{(i,j)} = \frac{(d_i+1)^r (d_j+1)^{1-r}}{2m+n}$, d_i and d_j are the node degrees for v_i and v_j. it shows that the final feature is over-smoothed and unable to capture the full graph structure information since it only relates to the node degrees of target nodes and source nodes. We use Euclidean distance to measure the distance between current node features and over-smoothing node features as $\mathbf{EucDis}^k(i) = \|(\tilde{A}^k X)[i] - (\tilde{A}^\infty X)[i]\|_2$, where $(\tilde{A}^k X)[i]$ is a feature of k number of propagation for node i. For convenience, in the experiment, we set r to 0. To demonstrate the effectiveness of our strategy, we divided the nodes in the Cora, Citeseer, and Pubmed datasets into three groups based on their degree size: degree ≤ 3, $10 \leq$ degree ≤ 50, and $100 \leq$ degree (Variations in the partitioning of the Citeseer dataset exist due to its small size). We then observed the speed of smoothing for each group. We set $\mathbb{N}$ in Eq.9 to 128. Figure 5 demonstrates that Node-wise adaptive propagation effectively addresses the issue of rapid over-smoothing in the large degree group. For each node group, it prevents the convergence of their node features with the growth of aggregation steps. This prevents potential model undifferentiation and consequent impact on experiments.

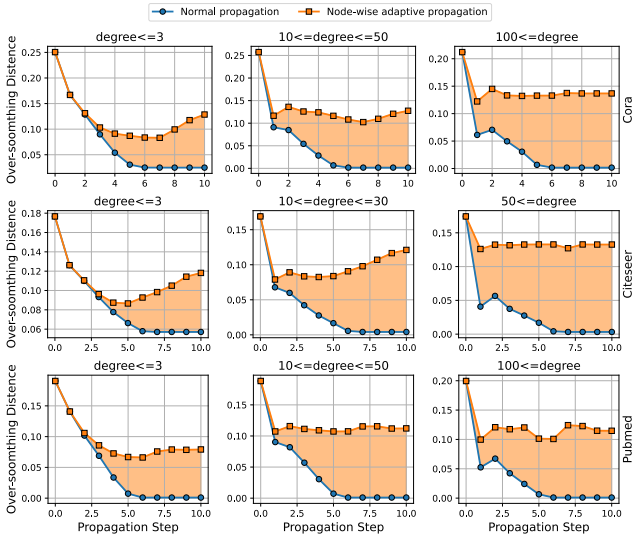

Figure 5: Node-wise adaptive propagation analysis.

6 CONCLUSION

In this paper, we propose GSD-GNN, a scalable and generalized graph neural network based on spectral graph theory, which can deal with massive graphs with millions of nodes and billions of edges. GSD-GNN first non-trivially generalizes random-walk matrix-polynomials and establishes its relationship with the propagation matrix. It then employs a node-wise adaptive strategy to enhance the model performance. In particular, the paper provides a theoretical analysis of the approximation error for the generalized propagation matrix. Finally, extensive experiments on seven real-world graphs and eleven competitors demonstrate that GSD-GNN is indeed highly accurate, and scalable.

REFERENCES

[1] [n. d.]. Technical Report for GCD-GNN. https://anonymous.4open.science/r/GSD-GNN-4200.

[2] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. 2006. Local Graph Partitioning using PageRank Vectors. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS*. IEEE Computer Society.

[3] Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling graph neural networks with approximate pagerank. In *KDD*. 2464–2473.

[4] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* (2018).

[5] Jianfei Chen, Jun Zhu, and Le Song. 2017. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568* (2017).

[6] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable graph neural networks via bidirectional propagation. *Advances in neural information processing systems* 33 (2020), 14556–14566.

[7] Dehua Cheng, Yu Cheng, Yan Liu, Richard Peng, and Shang-Hua Teng. 2015. Spectral Sparsification of Random-Walk Matrix Polynomials. *CoRR* (2015).

[8] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*. 257–266.

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[10] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*. ACM, 639–648.

[11] Roger A. Horn and Charles R. Johnson. 1991. *Topics in Matrix Analysis*.

[12] Keke Huang, Jing Tang, Juncheng Liu, Renchi Yang, and Xiaokui Xiao. 2023. Node-Wise Diffusion for Scalable Graph Learning. WWW.

[13] Leo Katz. 1953. A new status index derived from sociometric analysis. *Psychometrika* (1953).

[14] Nikhil Varma Keetha, Chen Wang, Yuheng Qiu, Kuan Xu, and Sebastian A. Scherer. 2022. AirObject: A Temporally Evolving Graph Embedding for Object Identification. In *CVPR*. 8397–8406.

[15] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*. OpenReview.net.

[16] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*.

[17] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. In *NeurIPS*. 13333–13345.

[18] Risi Kondor. 2002. Diffusion kernels on graphs and other discrete structures. In *International Conference on Machine Learning*.

[19] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, Hongyang Chen, Hongchao Qin, and Guoren Wang. 2023. Efficient Personalized PageRank Computation: The Power of Variance-Reduced Monte Carlo Approaches. *Proc. ACM Manag. Data* (2023).

[20] Meng Liu, Hongyang Gao, and Shuiwang Ji. 2020. Towards deeper graph neural networks. In *KDD*. 338–348.

[21] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank Citation Ranking : Bringing Order to the Web. In *The Web Conference*.

[22] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. 2018. DeepInf: Social Influence Prediction with Deep Learning. In *SIGKDD*. Association for Computing Machinery.

[23] Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. 2021. Interpreting Graph Neural Networks for NLP With Differentiable Edge Masking. In *ICLR*.

[24] Daniel A. Spielman and Nikhil Srivastava. 2011. Graph Sparsification by Effective Resistances. *SIAM J. Comput.* (2011), 1913–1926.

[25] Daniel A. Spielman and Shang-Hua Teng. 2011. Spectral Sparsification of Graphs. *SIAM J. Comput.* 40, 4 (2011), 981–1025.

[26] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2017. Graph Attention Networks. *CoRR* (2017).

[27] Hanzhi Wang, Mingguo He, Zhewei Wei, Sibo Wang, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2021. Approximate Graph Propagation. In *KDD*. 1686–1696.

[28] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315* (2019).

[29] Suhang Wang, Jiliang Tang, Charu C. Aggarwal, Yi Chang, and Huan Liu. 2017. Signed Network Embedding in Social Media. In *SIAM*.

[30] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*, Vol. 97. 6861–6871.

[31] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*.

[32] Chenxiao Yang, Qitian Wu, Jiahua Wang, and Junchi Yan. 2023. Graph Neural Networks are Inherently Good Generalizers: Insights by Bridging GNNs and MLPs. ICLR.

[33] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*. 974–983.

[34] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. 2021. Decoupling the depth and scope of graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 19665–19679.

[35] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019).

[36] Wentao Zhang, Zeang Sheng, Ziqi Yin, Yuezihan Jiang, Yikuan Xia, Jun Gao, Zhi Yang, and Bin Cui. 2022. Model Degradation Hinders Deep Graph Neural Networks. In *KDD*. 2493–2503.

[37] Wentao Zhang, Mingyu Yang, Zeang Sheng, Yang Li, Wen Ouyang, Yangyu Tao, Zhi Yang, and Bin Cui. 2021. Node Dependent Local Smoothing for Scalable Graph Learning. nips:2110.14377

[38] Long Zhao, Xi Peng, Yu Tian, Mubbasir Kapadia, and Dimitris N. Metaxas. 2019. Semantic Graph Convolutional Networks for 3D Human Pose Regression. In *CVPR*. 3425–3435.

[39] Hao Zhu and Piotr Koniusz. 2021. Simple Spectral Graph Convolution. In *ICLR*.

[40] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. *NeurIPS* (2019).

7 SUPPLEMENTARY MATERIALS

7.1 Complexity Analysis

The GSD-GNN is primarily segmented into two phases.

• During the construction of the propagation matrix phase, it is assumed that sampling a neighbor can be accomplished in constant time, denoted as $O(1)$. To achieve a sparsifier with $O(M)$ non-zero elements, we need to sample paths M times. The expected value of x, denoted as $\mathbb{E}(x)$, is given by $\sum_{k=1}^{T} k\theta_k$. Subsequently, To calculate the propagation matrix, it takes $O(M)$ time. As a result, the time complexity for unweighted networks is $O(M\mathbb{E}(x))$. To compute Eq. 7, it only needs to traverse the elements in $\mathbf{L}_{\boldsymbol{\alpha}}(G)$ and perform operations, without any matrix multiplication. The maximum time complexity is $O(M)$, and considering that duplicate edges are sampled, the actual complexity should be less than $O(M)$. Regarding space complexity, storing $\tilde{\mathbf{G}}$ requires $O(M)$ space, while the input network and degree matrix contribute an additional $O(m)$ and $O(n)$ space, respectively. Therefore, the time complexity and the space complexity of Algorithm 2 are $O(M(\mathbb{E}(x)+1))$ and $O(M+m+n)$, respectively.

• During the training phase, The training and inference time complexity of a GSD-GNN can be bounded by $O(nd^2 + Md)$, where $O(nd^2)$ is the total cost of the feature transformation by applying $\mathbf{W}$ the sparse-dense matrix multiplication $\mathbf{\Pi H}$ and $O(Md)$ is the total cost of the sparse-dense matrix multiplication $\mathbf{\Pi H}$.

In a nutshell, the time (resp., space) complexity of *GSD-GNN* is $O(nd^2 + M(1 + d + \mathbb{E}(x)))$ (resp., $O(M + m + n)$).

7.2 Error Analysis

To establish the validity of Theorem 2 and 3, we will rely on several lemmas, the details and proofs of which are presented as follows.

Lemma 2. *For $\mathcal{L} = D^{-1/2}L_\alpha(G)D^{-1/2}$ and normalized sparsifier $\widetilde{\mathcal{L}} = D^{-1/2}\widetilde{L}D^{-1/2}$,all the singular values of $\mathcal{L} - \widetilde{\mathcal{L}} \leq 4\epsilon$.*

Proof. Evidently,$\mathcal{L}$ is a normalized graph Laplacian with eigenvalues in the interval [0, 2), i.e., for $i \in [n]$, $\lambda_i(L) \in$[0,2). From eq.4 in section 3.2, we know that

$$\frac{1}{(1+\epsilon)} \cdot \boldsymbol{x}^\top L_{\boldsymbol{\alpha}}(G)\boldsymbol{x} \leq \boldsymbol{x}^\top \tilde{\mathbf{L}}\boldsymbol{x} \leq \frac{1}{(1-\epsilon)} \cdot \boldsymbol{x}^\top L_{\boldsymbol{\alpha}}(G)\boldsymbol{x}$$

Let x=$D^{-1/2}y$ and all terms in the equation are simultaneously subtracted by $\boldsymbol{x}^\top \mathcal{L}\boldsymbol{x}$,as follows

$$-\frac{\epsilon}{(1+\epsilon)} \cdot \boldsymbol{y}^\top \mathcal{L}\boldsymbol{y} \leq \boldsymbol{y}^\top(\widetilde{\mathcal{L}}-\mathcal{L})\boldsymbol{y} \leq \frac{\epsilon}{(1-\epsilon)} \cdot \boldsymbol{y}^\top \mathcal{L}\boldsymbol{y}$$

Take the absolute value of the equation:

$$|\boldsymbol{y}^\top(\widetilde{\mathcal{L}}-\mathcal{L})\boldsymbol{y}| \leq \frac{\epsilon}{(1-\epsilon)} \cdot \boldsymbol{y}^\top \mathcal{L}\boldsymbol{y} \leq 2\epsilon \boldsymbol{y}^\top \mathcal{L}\boldsymbol{y}$$

here,we assume $\epsilon < 0.5$.Then by Courant-Fisher Theorem, we get that

$$|\lambda_i(\widetilde{\mathcal{L}}-\mathcal{L})| \leq 2\epsilon\lambda_i(\mathcal{L}) \leq 4\epsilon$$

where $i \in [n]$,$\lambda_i(A)$ is i-th largest eigenvalue of matrix A. □

Lemma 3. *[11] if B, C be two $n \times n$ symmetric matrices, for the decreasingly-ordered singular values λ of B, C and BC,*

$$\lambda_{i+j-1}(BC) \leq \lambda_i(B) \times \lambda_j(BC)$$

where $i \leq i, j \leq n$ and $i + j \leq n + 1$.

We examine the approximation error associated with the propagation matrix. We employ the F-norm to gauge the discrepancy between the actual and approximate propagation matrices, defining the error as $\|\Pi - \tilde{\Pi}\|_F$. The error is attributed to two primary factors: truncation and approximation. This situation can be reformulated as $\|\Pi - \Pi' + \Pi' - \tilde{\Pi}\|_F$. According to the triangle inequality, $\|\Pi - \Pi' + \Pi' - \tilde{\Pi}\|_F \leq \|\Pi - \Pi'\|_F + \|\Pi' - \tilde{\Pi}\|_F = \Delta\mathcal{L}_1 + \Delta\mathcal{L}_2$ holds true, where $\Delta\mathcal{L}_1$ and $\Delta\mathcal{L}_2$ represent truncation error and approximation error, respectively. Subsequent analysis will focus on the individual errors stemming from these two terms.

Theorem 3. *For error $\Delta\mathcal{L}_1$ caused by truncation, the $\Delta\mathcal{L}_1 \leq \frac{1}{C}\sqrt{n}\frac{\omega^{T+1}}{((T+1)!)^\rho}$, where n is the number of nodes.*

Proof. Suppose the truncation length is T, then we get

$$\begin{aligned}
\|\Pi - \Pi'\|_F &= \|\frac{1}{C}\sum\nolimits_{i=T+1}^{\infty} \frac{\omega^i}{(i!)^\rho}(D^{r-1}AD^{-r})^i\|_F \\
&= \frac{1}{C}\sum\nolimits_{i=T+1}^{\infty} \frac{\omega^i}{(i!)^\rho}\sqrt{\sum_{j\in[n]} \lambda_j^2((D^{r-1}AD^{-r})^i)} \\
&\leq \frac{1}{C}\sum\nolimits_{i=T+1}^{\infty} \frac{\omega^i}{(i!)^\rho}\sqrt{\sum_{j\in[n]} \lambda_j^2(D^{r-1}AD^{-r})} \\
&\leq \frac{1}{C}\sqrt{n}\frac{\omega^{T+1}}{((T+1)!)^\rho}
\end{aligned}$$

because $\lambda_j((D^{r-1}AD^{-r})^i) \leq \lambda_j(D^{r-1}AD^{-r}) * \lambda_1(D^{r-1}AD^{-r}) * \ldots * \lambda_1(D^{r-1}AD^{-r}) \leq \lambda_j(D^{r-1}AD^{-r})$ by Lemma 3 and $\lambda_s(D^{r-1}AD^{-r}) \in [0, 1]$ for any $s \in [n]$. □

Theorem 4. *For error $\Delta\mathcal{L}_2$ caused by spectral approximation, the $\Delta\mathcal{L}_2 \leq 4\epsilon \cdot \mathbb{C}\sqrt{n}$, where n is the number of nodes and ϵ is approximation parameter.*

Table 5: Parameter settings of our proposed GSD-GNN

Dataset	ω	ρ	d	$M = c * \lvert\mathcal{E}\rvert$	r
Cora	1.15	0.06	5	120	0.4
Citeseer	1.12	0.04	6	400	0.6
Pubmed	4	1.5	15	400	0.5
Flickr	5.5	1.1	6	240	0.4
AMiner-CS	10	1.1	50	80	0.6
Reddit	4	1.15	6	80	0.45
Papers100M	7	1.35	6	10	0.5

Proof. First, we transform the equation $\Pi' - \tilde{\Pi}$, we have

$$\begin{aligned}
\Pi' - \tilde{\Pi} &= \mathbb{C}D^{r-1}(\tilde{\mathbf{L}} - L_{\boldsymbol{\alpha}}(G))D^{-r} \\
&= \mathbb{C}D^{r-1/2}D^{-1/2}(\tilde{\mathbf{L}} - L_{\boldsymbol{\alpha}}(G))D^{-1/2}D^{1/2-r}.
\end{aligned}$$

Then according to Lemma 2 and Lemma 3, we have,

$$\begin{aligned}
\lambda_i(\Pi' - \tilde{\Pi}) &\leq \mathbb{C}\lambda_1(D^{r-1/2})\lambda_i(\widetilde{\mathcal{L}} - \mathcal{L})\lambda_1(D^{1/2-r}) \\
&\leq 4\epsilon \cdot dC.
\end{aligned}$$

So, $\|\Pi' - \tilde{\Pi}\|_F = \sqrt{\sum_{i\in[n]} \lambda_i^2(\Pi' - \tilde{\Pi})} \leq 4\epsilon \cdot \mathbb{C}\sqrt{n}$. □

7.3 Additional Experiments Settings

Parameter Settings. Table 5 reports the parameter settings of our proposed GSD-GNN on each dataset.

Explanation for Inductive Baselines. Certain transductive learning baselines were excluded from the evaluation of inductive learning due to their unsuitability for inductive learning. For instance, Each layer of GCN requires a complete adjacency matrix as input, otherwise its performance will severely degrade. This requirement conflicts with the setting of inductive learning. GraphSAINT [35] randomly samples a subgraph for training, requiring each node in the subgraph to be labeled. In the semi-supervised setting of large graphs like Reddit, the limited labeled nodes significantly degrade GraphSAINT's performance.