Speedup Robust Graph Structure Learning with Low-Rank Information

Hui Xu, Liyao Xiang*, Jiahao Yu, Anqi Cao, Xinbing Wang*
Shanghai Jiao Tong University
{xhui_1,xiangliyao08,yujiahao,caoanqi,xwang8}@sjtu.edu.cn

ABSTRACT

Recent studies have shown that graph neural networks (GNNs) are vulnerable to unnoticeable adversarial perturbations, which largely confines their deployment in many safety-critical domains. Robust graph structure learning has been proposed to improve the GNN performance in the face of adversarial attacks. In particular, the low-rank methods are utilized to purify the perturbed graphs. However, these methods are mostly computationally expensive with $O(n^3)$ time complexity and $O(n^2)$ space complexity. We propose LRGNN, a fast and robust graph structure learning framework, which exploits the low-rank property as prior knowledge to speed up optimization. To eliminate adversarial perturbation, LRGNN decouples the adjacency matrix into a low-rank component and a sparse one, and learns by minimizing the rank of the first part while suppressing the second part. Its sparse variant is formed to reduce the memory footprint further. Experimental results on various attack settings have shown LRGNN acquires comparable robustness with the state-of-the-art much more efficiently, boasting a significant advantage on large-scale graphs.

CCS CONCEPTS

• **Mathematics of computing → Spectra of graphs**.

KEYWORDS

Graph convolutional networks, Matrix Rank Minimization, Robustness

ACM Reference Format:
Hui Xu, Liyao Xiang*, Jiahao Yu, Anqi Cao, Xinbing Wang*. 2021. Speedup Robust Graph Structure Learning with Low-Rank Information. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482299

1 INTRODUCTION

Graphs are ubiquitous structures representing real-world data, such as social networks, academic networks, biological networks, etc. Extensive studies for Graph Neural Networks (GNNs) have arisen in recent years showing the power of graph structure learning [1–10]. Meanwhile, a range of works has demonstrated that GNNs are vulnerable to adversarial attacks [11, 12], where only a few unnoticeable perturbations can dramatically degrade the performance of GNNs. Such vulnerability has raised concerns about utilizing these models in high-security areas and leads to limitations in application. Therefore, obtaining robust GNNs resisting adversarial attacks is critical. In this paper, we focus on the graph structure perturbations injected before training, i.e., graph structure poisoning attacks. In this setting, graph structures are perturbed by adding/deleting/rewiring edges before training GNN models, whereas node features remain unchanged.

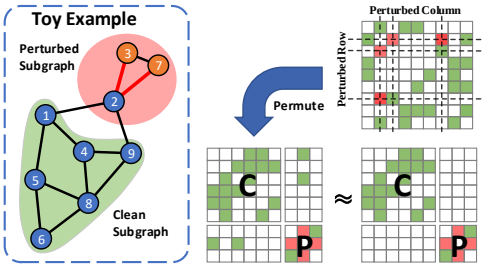


Figure 1: The left is a toy example of adversarial attacks on the graph, where red lines denote adversarial edge insertion. The right is the corresponding adjacency matrix, where the dash lines denote the perturbed rows and columns. In the adjacency matrix, green grids denote normal edges, and red grids denote perturbed edges.

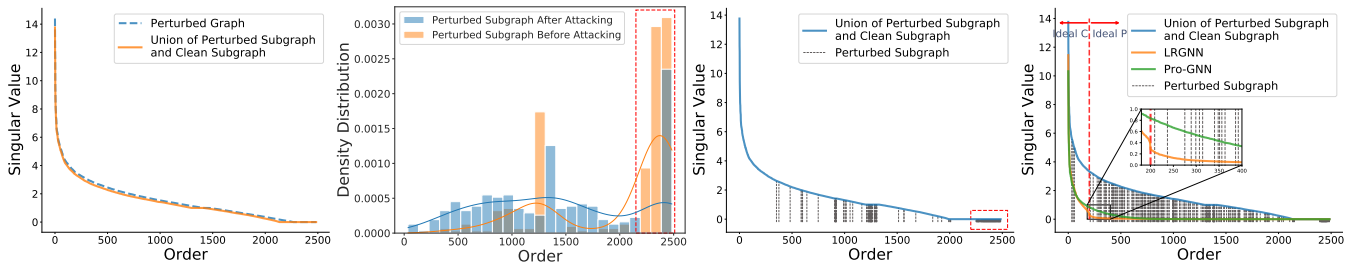
Great efforts have been devoted to robust graph structure learning [13–17]. One promising way is to eliminate the influence of attacks from the aspects of graph spectrum, i.e., the set of eigenvalues of the adjacency matrix of the graph. Empirical results have revealed that either targeted attack or non-targeted attack will significantly change the spectrum of graphs and increase the rank of the adjacency matrix [16, 17]. Moreover, removing adversarial edges reduces rank faster than removing normal edges [17]. Hence prior methods exploit matrix rank minimization to clean the adjacency matrix with respect to the graph spectrum. However, existing methods are computationally expensive as they optimize on perturbed graphs directly, resulting in $O(n^3)$ time complexity and $O(n^2)$ space complexity.

Beyond that, it is difficult for the existing methods to filter out the low-rank components incurred by adversarial attacks. To better understand this problem, we analyze how attacks influence the graph spectrum. Figure 1 depicts a toy example of graph adversarial attacks with edges insertion. We denote the adjacency matrix of the perturbed subgraph as P and the clean subgraph as C. Considering the sparse nature of graphs, the bottom-left and the top-right

*Corresponding authors.

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482299



(a) Singular Values **(b) Order Distribution** **(c) Before Attacking** **(d) After Attacking**

Figure 2: An illustrative example on the spectrum changes of the adjacency matrix by adversarial attacks. In (b)(c), the red dashed rectangles correspond to each other. (d) illustrates the difference between LRGNN and previous methods on spectrum.

submatrices could be approximated by zero matrices. Hence the graph spectrum is the union of the spectra of P and C, since it is well known that the spectrum of a disconnected graph is the union of the spectra of its connected components. Specifically, we show the singular value vs. order on real-world graphs in Figure 2, where the SOTA graph attack, *metattack* [18] is performed.

As we further illustrate in Figure 2b, the distribution of P's singular values are denser after the attack. We visualize the graph spectrum before and after the attack in Figure 2c and 2d, and found adversarial perturbation significantly increases the singular values as well as the rank of P, introducing high-rank components into the graph spectrum. It can be observed that it is intrinsically hard to decouple the spectra of C and P as their singular values are mixed up and thus are dealt with indifferently.

In this paper, we propose LRGNN, standing for **l**ight and **r**obust **GNN** learning with **l**ow-**r**ank information. We resolve two issues: (i) how to decouple the spectra of P and C; and (ii) how to speed up robust graph structure learning. For the first issue, we follow the intuition of robust principal component analysis (RPCA) [19, 20], in assuming the adjacency matrix can be decomposed into a low-rank component (C) and a sparse component (P). As indicated by Figure 2d, the ideal clean subgraph is naturally low-rank, and LRGNN learns to minimize its rank; whereas the perturbed subgraph has a few low-rank components (mixed up with ideal C), while mainly occupies the higher-rank part of the spectrum (ideal P). And LRGNN isolates and suppresses it. Previous methods, not distinguishing the two components, try to suppress them together, which may lead to non-robust performance.

To solve the second issue, we reduce the search space from the entire graph of size $n \times n$ to a low-rank representation of size $r \times r$. With proper initialization, we can prove that optimization over the reduced space is equivalent to that over the original graph. Moreover, to avoid constructing a full adjacency matrix, which requires $n \times n$ memory footprint, we propose a sparse variant of LRGNN to make it more scalable. It samples entries of the adjacency matrix and treats them as incomplete observations to the graph.

Highlights of our contributions are as follows:

- Based on the intuition of RPCA, LRGNN decouples the spectra of C and P in the joint learning, leading to more robust graph purification in the spectrum domain.
- By taking the low-rank information as prior, LRGNN reduces the optimization search space from the entire graph to a low-rank representation. A sparse variant of LRGNN is further derived to reduce memory consumption.
- Experiments on a variety of real-world datasets demonstrate that LRGNN effectively defends against different types of attacks, yet 15x faster and with 10x less memory over SOTA methods on the large Pubmed graph.

The rest of the paper is organized as follows. Section 2 briefly reviews related works. Section 3 introduces the background to facilitate understanding. Section 4 formally describes the problem and our proposed framework. Section 5 reports the evaluation results. Section 6 concludes the paper.

2 RELATED WORK

2.1 Adversarial Attacks On Graphs

Even though GNN successfully takes graph tasks onto deep learning domain, it naturally inherits the vulnerability of general deep learning methods. Even slight or unnoticeable perturbations on graphs can significantly hurt the performance of GNN. For instance, Zügner et al. [21] propose *nettack* which injects unnoticeable perturbation on the graph structure and nodes' features, while preserving important data characteristics. RL-S2V [22] performs reinforcement learning to learn the way of injecting adversarial attacks on graph with prediction feedback from the target classifier. However, these methods mainly focus on the targeted attack, and employ greedy approximation to generate adversarial examples on discrete graph data. As an alternative, Xu et al. [23] model untargeted attack as a min-max problem over the GNN model and graph structures, where graph structure can be optimized in a continuous domain, and the results are projected back to the discrete ones. Likewise, *metattack* [18] proposes to generate untargeted attacks via meta-learning.

2.2 Robust Graph Learning

As extensive methods have demonstrated the latent vulnerability of general GNNs, a variety of mechanisms have been designed to improve the robustness of GNN models. One family is to penalize latent adversarial nodes in information aggregation. RGCN [13] proposes to model hidden-layer features as Gaussian variables and assigns fewer weights to nodes with higher variance in aggregating neighbors' information. SimP-GCN [14] constructs a new feature graph according to node features which achieve robustness by joint learning on structure graph and feature graph.

Another family is to preprocess the graph structure by preserving inherent graph properties. Wu et al. [15] found the attacked nodes

tend to have dissimilar features, by which they propose to remove links between dissimilar nodes as a defense. Entezari et al. [16] found *nettack* affects the the high-rank part of the graph spectrum, and exclude the impact with low-rank approximation. Since the preprocessing step is independent of training, the preprocessed graph may be suboptimal for GNN models. To resolve the issue, Jin et al. [17] propose a joint framework for graph structure learning and the GNN task to preserve sparsity and low-rankness of the graph.

However, we observe a significant lack of efficiency in previous low-rank approximation based approaches. Hence, we propose a method to speed up robust structure learning with the low-rank property as prior knowledge. We also perform graph structure learning jointly with GNN tasks.

3 PRELIMINARIES

We introduce some preliminaries for ease of understanding this work.

3.1 Matrix Rank Minimization

Matrix rank minimization aims to acquire a low-rank approximation of the input matrix [24, 25], which can be formulated as

$$\begin{aligned} \underset{L}{\text{minimize}} \quad & \text{rank}(L) \\ \text{subject to} \quad & ||A - L||_F \leq \delta, \end{aligned} \tag{1}$$

where A is the input matrix, L is the learned low-rank matrix, and δ is the noise level. $||\cdot||_F$ denotes the Frobenius norm. However, to solve such a rank minimization problem is usually impractical as it is NP-hard. As an alternative, we can replace the rank function with the nuclear norm, since the nuclear norm minimization is the tightest convex relaxation of the rank minimization. Hence we can formulate the problem as

$$\begin{aligned} \underset{L}{\text{minimize}} \quad & ||L||_* \\ \text{subject to} \quad & ||A - L||_F \leq \delta, \end{aligned} \tag{2}$$

where $||L||_* = \sum_i \sigma_i$, and σ_i is the i-th singular value of L. Furthermore, as Eq.2 satisfies the Slater's condition [26, 27], we can derive a more efficient solution by

$$\underset{L}{\text{minimize}} \quad \frac{1}{2}||A - L||_F^2 + \mu||L||_*. \tag{3}$$

It is well established that Eq. 3 is equivalent to Eq. 2 for some value $\mu(\delta)$.

3.2 Matrix Completion

Matrix completion (MC) is a promising technique to recover an intact matrix with low-rank property from undersampled/incomplete data [28, 29]. When the sampled entries are corrupted by noise, the MC problem can be formulated as a rank minimization problem:

$$\begin{aligned} \underset{L}{\text{minimize}} \quad & \text{rank}(L) \\ \text{subject to} \quad & ||P_\Omega(A - L)||_F \leq \delta, \end{aligned} \tag{4}$$

where Ω is a subset of $\{(i,j)|0 \leq i < n, 0 \leq j < n\}$, and the operator $P_\Omega : \mathbb{R}^{m\times n} \rightarrow \mathbb{R}^{m\times n}$ is defined as $[P_\Omega(A)] = A_{ij}$, if $(i,j) \in \Omega$, and

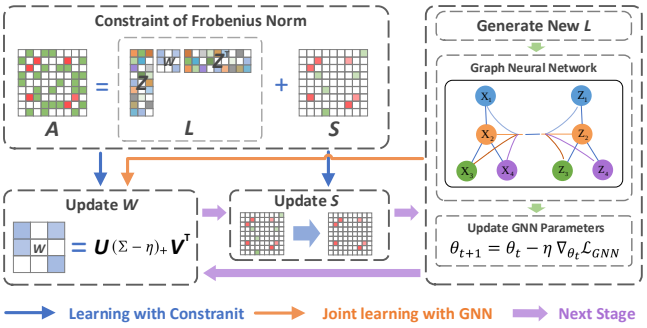


Figure 3: Overall framework of LRGNN. We alternatively update the model over W, S and θ.

$[P_\Omega(A)]_{ij} = 0$ otherwise. Similar to Eq. 3, the optimization problem can also be expressed as

$$\underset{L}{\text{minimize}} \quad \frac{1}{2}||P_\Omega(A - L)||_F^2 + \mu||L||_*. \tag{5}$$

4 THE PROPOSED FRAMEWORK

The illustration of the framework is shown in Figure 3. To defend against adversarial attacks, LRGNN learns to isolate the latent structure attacks into the sparse matrix S while keeping the clean part L low-rank. The low-rank part is parameterized by W, and GNN is parameterized by θ. By optimizing over W, S, and θ, LRGNN recovers the clean adjacency matrix jointly with GNN tasks. In this section, we will define the problem and illustrate the detail of the proposed framework.

4.1 Problem Statement

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$ be a graph, where $\mathcal{V} = \{v_1, v_2, ..., v_n\}$ is the set of nodes, $\mathcal{E}$ is the set of edges and $X = \{x_1^T, x_2^T, ..., x_n^T\} \in \mathbb{R}^{n\times d}$ denotes the node feature matrix with dimension d. The relation between nodes can also be represented by an adjacency matrix $A \in \{0,1\}^{n\times n}$, where $A_{ij} = 1$ if there exists an edge between v_i and v_j and $A_{ij} = 0$ otherwise. Furthermore, following the common setting of node classification, only a partial node set $\mathcal{V}_L = \{v_1, v_2, ...v_l\}$ is labeled, and the labels are correspondingly $C_L = \{c_1, c_2, ...c_l\}$.

We mainly focus on the most general type of GNN, i.e., graph convolutional network (GCN)[3], in this work. Other GNN variants can be applied with straightforward extensions. Given the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$ and label set C_L, GCN aims to learn a mapping function $f_\theta(A, X) : \mathcal{V}_L \rightarrow C_L$ according to the labeled node set to make predictions on the unlabeled nodes. The formulation can be represented as

$$\underset{\theta}{\text{minimize}}\ \mathcal{L}(A, X, \theta, C_L) = \sum_{v_i \in \mathcal{V}_L} l(f_\theta(A, X)_i, y_i), \tag{6}$$

where θ is the parameter of f_θ, and $l(\cdot,\cdot)$ is the loss function. Specifically, $f_\theta(A, X)$ is implemented by a two-layer message passing model, expressed as

$$f_\theta(A, X) = \text{softmax}(\hat{A}\sigma(\hat{A}XW_1)W_2) \tag{7}$$

where $\theta = \{W_1, W_2\}$, σ is ReLU activation function, $\hat{A} = \tilde{D}^{-1/2}(A + I)\tilde{D}^{-1/2}$ and $\tilde{D}$ is the diagonal matrix of $A + I$ with $\tilde{D}_{ii} = \sum_j A_{ij} + 1$. Formally, fast and robust graph structure learning is as follows:

Definition 4.1. Given label set C_L and graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ with adversarially perturbed A and clean X, we aim to simultaneously learn a clean graph structure and the GNN weights efficiently, which has high prediction accuracy on the node classification task of the unlabeled nodes.

4.2 Speed Up Robust Graph Structure Learning

As depicted in Figure 2d, adversarial attacks dramatically increase the singular values of the subgraph perturbed. Hence one effective way to defend the malicious perturbations is to purify the adjacency matrix A by removing high-rank components with respect to Eq. 3. However, there remains a significant computational challenge as the computation of $||L||_*$ requires $O(n^3)$ time complexity due to singular value decomposition (SVD). In this section, we first propose an efficient approach to perform robust structure learning. To effectively filter out the spectrum of the perturbed subgraph, we further adopt a robust PCA based approach to recover a clean adjacency matrix L.

4.2.1 ***Fast Matrix Rank Minimization***. Overall, we provide a new formulation as

$$\underset{W}{\text{minimize}} \quad \frac{1}{2}||A - ZWZ^T||_F^2 + \mu||W||_*, \tag{8}$$

where $Z \in \mathbb{R}^{n\times r}$ is the top-r columns of left singular matrix of A, and $W \in \mathbb{R}^{r\times r}$ is the learned weight matrix. In this formulation, L is decomposed as ZWZ^T, and $||L||_* = ||W||_*$ since $Z^TZ = I_{r\times r}$. Benefiting from optimizing over W in this formulation, we reduce the time complexity from $O(n^3)$ to $O(r^3)$, which is much faster as $r \ll n$. To solve such an optimization problem with the non-differentiable $||W||_*$, we use Forward-Backward Splitting (FBS) algorithm [30] to iteratively update W by

$$W^k = \text{prox}_{\eta\mu||\cdot||_*}(W^{k-1} - \eta\nabla_W||A - ZWZ^T||_F^2), \tag{9}$$

where η is learning rate and $\text{prox}_{\eta\mu||\cdot||_*}(\cdot)$ is the proximal operator of nuclear norm, which can be represented as

$$\text{prox}_{\eta\mu||\cdot||_*}(W) = U(\sigma_i - \eta\mu)_+V^T, \tag{10}$$

where $W = Udiag(\sigma_1, ..., \sigma_r)V^T$ by singular value decomposition.

To understand how it works, we in the following prove that optimizing Eq. 8 is equivalent to optimize Eq. 3 when the adjacency matrix A has rank r.

Theorem 4.2. *If symmetric adjacency matrix $A \in \mathbb{R}^{n\times n}$ has rank r ($r < n$), given the singular value decomposition $A = U\Sigma_rU^T$, $Z = U$, the initial values for W and L are $W^0 = Z^TAZ = \Sigma_r$, and $L^0 = A = Z\Sigma_rZ^T$ respectively, optimizing Eq. 8 is equivalent to optimizing Eq. 3 in the dynamics of the FBS algorithm.*

Proof. By using FBS algorithm, at iteration k, updating L in Eq. 3 and W in Eq. 8 is to perform

$$\hat{L}^k = L^k - \tau^k(L^k - L^0), \tag{11}$$

$$\hat{W}^k = W^k - \tau^k(W^k - W^0). \tag{12}$$

The proximal operator is followed to perform:

$$L^{k+1} = U_{\hat{L}^k}h_k(\Sigma_{\hat{L}^k})V_{\hat{L}^k}^T, \tag{13}$$

$$W^{k+1} = U_{\hat{W}^k}h_k(\Sigma_{\hat{W}^k})V_{\hat{W}^k}^T, \tag{14}$$

where $X = U_X\text{diag}(\sigma_{X,1}, ..., \sigma_{X,n})V_X^T$ is the singular value decompostion of X, and $X = \hat{L}^k, \hat{W}^k$ respectively. And $h_k(\Sigma) = \max(\sigma_i - \tau^k, 0)$ for each diagonal element σ_i in Σ where τ is the learning rate. We will use induction to prove that in each iteration k, $L^k = ZW^kZ^T$.

First we prove that W^k is a diagonal matrix for any k. Obviously $W_0 = \Sigma_r$ and W_1 are diagonal matrices. Suppose that W^k is a diagonal matrix for any k. According to Eq. 12, $\hat{W}^k$ is a diagonal matrix. Therefore, $U_{\hat{W}_k} = V_{\hat{W}_k} = I$ where I is identity matrix and $\Sigma_{\hat{W}_k} = \hat{W}_k$. Hence $W^{k+1} = h_k(\hat{W}^k)$ is diagonal for any k. Meanwhile, we also have $W^{k+1} = h_k(\hat{W}^k)$ for any k.

Then we prove that $L^k = ZW^kZ^T$ for any k. Obviously $L^0 = ZW^0Z^T$ and $L^1 = ZW^1Z^T$. Suppose that $L^k = ZW^kZ^T$ for any k. According to Eq. 11,

$$\hat{L}^k = L^k - \tau^k(L^k - L^0) = Z(W^k - \tau^k(W^k - W^0))Z^T = Z\hat{W}^kZ^T. \tag{15}$$

As $\hat{W}^k$ is diagonal, $\hat{L}^k$ must be symmetric. Then we have $\Sigma_{\hat{L}^k} = \hat{W}^k$. Therefore, $L^{k+1} = U_{\hat{L}^k}h_k(\Sigma_{\hat{L}^k})V_{\hat{L}^k}^T = Zh_k(\hat{W}^k)Z^T = ZW^{k+1}Z^T$. Hence the case holds for $k+1$. The optimization step of Eq. 8 is equivalent to that of Eq. 3. If Eq. 8 converges to a stationary point, Eq. 3 would converge to the same point. □

By Thm. 4.2, we prove that, with proper initialization and FBS algorithm, optimizing Eq. 8 is equivalent to optimizing Eq. 3 when adjacency matrix A has rank r. Note that in the real world, the perturbed adjacency matrix A is not always low-rank with a small value of r. Hence we in practice choose an appropriate r independent of the true rank of the adjacency matrix A, and further empirically demonstrate that Eq. 8 can still acquire robust results even if in this circumstance.

4.2.2 ***Robust Low-Rank Matrix Recovery***. Although Eq. 8 is an efficient solution for recovering a low-rank adjacency matrix L, it is not effective enough to purify the perturbed graph. As depicted in Figure 2(d), the distributions of singular values of clean subgraph and perturbed subgraph are heavily mixed up. With the increase of perturbation rate, the singular values of the perturbed subgraph will gradually fill out the entire spectrum. As the classical nuclear norm minimization treats all singular values equally, relative large singular values from perturbed subgraph cannot be effectively detected and removed.

To solve the problem, we propose an RPCA-based approach separating the latent perturbations from the perturbed adjacency matrix. Specifically, we assume the perturbed adjacency matrix A can be decomposed as a low-rank component L and a sparse component S, i.e., $A = L + S$. In this way, we can isolate the latent perturbations to S and acquire a clean adjacency matrix L. Consequently, the objective function can be formulated as

$$\underset{W,S}{\text{minimize}} \quad \frac{1}{2}||A - S - ZWZ^T||_F^2 + \mu||W||_* + \alpha||S||_1, \tag{16}$$

where $S \in \mathbb{R}^{n\times n}$ is the learned sparse matrix. Here, we use $A - S$ to replace A and add a l_1 norm regularizer for S. Since attackers try to manipulate the input graph with unnoticeable perturbations, matrix S is sparse in nature. In this way, the perturbations or noises with arbitrary magnitude and distribution can be isolated into S.

And we optimize over W for the low-rank matrix $L = ZWZ^T$ by the conclusion of Thm. 4.2.

4.2.3 ***Final Objective Function.*** To avoid the suboptimal solution of GNN model [17], we jointly learn the graph structure with the GNN task. Specifically, we combine the aforementioned proposal with the downstream task, and the final objective function is

$$\begin{aligned}\underset{W,S,\theta}{\text{minimize}} \quad & \frac{1}{2}||A - S - ZWZ^T||_F^2 + \mu||W||_* + \alpha||S||_1 \\ & +\gamma\mathcal{L}_{GNN}(\theta, ZWZ^T, X, C_L), \quad s.t., \quad ZWZ^T \geq 0,\end{aligned} \tag{17}$$

where $\mathcal{L}_{GNN}$ is the loss function of the downstream task, and C_L is the labeled set for training. In particular, to ensure that the summation $\sum_i A_{ij}$ and $\sum_j A_{ij}$ for each row i and column j are nonnegative for the normalization operation in GNN, we add an constraint for learned adjacency matrix as $ZWZ^T \geq 0$.

4.3 Scaling to Large Graphs

Nuclear norm minimization still has $O(n^2)$ space complexity, which is unacceptable for large-scale inputs. Hence we apply a relaxation to LRGNN to make it scalable to large graphs. The algorithm is referred to as LRGNN(S) as S stands for sampling or sparse. LRGNN(S) is inspired by the empirical observation that adversarial attacker prefers to inserting edges rather than deleting edges. According to the experiment in [18], roughly 80% of meta attack's perturbations are edge insertions. This indicates that we can alleviate most attacks even if we only remove existing edges that are potentially adversarial. Since real-world graph is usually sparse, the search space can be reduced from n^2 to $|\mathcal{E}|$, where $|\mathcal{E}|$ is the number of edges and $|\mathcal{E}| \ll n^2$. Hence we propose a new formulation as

$$\begin{aligned}\underset{W,S,\theta}{\text{minimize}} \quad & \frac{1}{2}||P_\Omega(A - S - ZWZ^T)||_F^2 + \mu||W||_* + \alpha||P_\Omega(S)||_1 \\ & +\gamma\mathcal{L}_{GNN}(\theta, P_\Omega(ZWZ^T), X, C_L), \quad s.t., \quad ZWZ^T \geq 0,\end{aligned} \tag{18}$$

where Ω is a subset of $\{(i,j)|0 \leq i < n, 0 \leq j < n\}$, and the operator $P_\Omega : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$ is defined as $[P_\Omega(A)] = A_{ij}$, if $(i,j) \in \Omega$, and $[P_\Omega(A)]_{ij} = 0$ otherwise. This is analogous to the matrix completion problem where the matrix is partially observed. Unlike their problem, we split Ω into Ω_e and Ω_o, where Ω_e corresponds to the existing edges indices and Ω_o denotes the rest. In other words, we take all non-zero entries in A, and a part of zero-entry samples into consideration. Hence we have $[P_{\Omega_e}(A)] = 1$ for $(i,j) \in \Omega_e$, and $[P_{\Omega_o}(A)] = 0$ for $(i,j) \in \Omega_o$.

Considering $|\Omega| \ll n^2$ and the elements for $(i,j) \notin \Omega$ will not be updated, we can vectorize the matrices according to Ω. Thus we express Eq. 18 as

$$\begin{aligned}\underset{W,s,\theta}{\text{minimize}} \quad & \frac{1}{2}||a - s - l||_2^2 + \mu||W||_* + \alpha||s||_1, \\ & +\gamma\mathcal{L}_{GNN}(\theta, l, X, C_L) \quad \text{s.t.,} \quad l \geq 0,\end{aligned} \tag{19}$$

where $l \in \mathbb{R}^{1\times|\Omega|}$, $s \in \mathbb{R}^{1\times|\Omega|}$, and $a \in \mathbb{R}^{1\times|\Omega|}$ are vectorized $P_\Omega(ZWZ^T)$, $P_\Omega(S)$, and $P_\Omega(A)$ respectively. Given an injective mapping function $\mathcal{M} : \Omega \to \{1, 2, ..., |\Omega|\}$ indicating the corresponding index for each pair $(i,j) \in \Omega$, we have $a_{\mathcal{M}((i,j))} = A_{i,j}$ if $(i,j) \in \Omega$. Each element of a equals to the corresponding value in A by $\mathcal{M}$. Similarly, l and s are respectively vectorized representation of ZWZ^T and S by $\mathcal{M}$. In this way, we can formulate the problem in a sparse version. Comparing to the original $O(n^2)$ space complexity, the space complexity of Eq. 19 is much smaller, depending on the sample space. In particular, if $\Omega = \Omega_e$, the formulation can be viewed as a low rank attention mechanism as we only modify the weights of existing edges to minimize the rank of ZWZ^T.

Algorithm 1 LRGNN

Require: Adjacency matrix A, Attribute matrix X, Labels C_L, Hyper-parameters μ, α, γ, r, Learning rate η_W, η_S, η_θ.
Ensure: Weight matrix W and S, GNN parameters θ

1: Given $A = U\Sigma V^T$, initialize $S^0 \leftarrow 0$, $Z \leftarrow U_{:,:r}$, $W^0 \leftarrow Z^T AZ$
2: Randomly initialize θ;
3: **while** Stopping condition is not met **do**
4: $\quad \hat{W}^{k+1} \leftarrow W^k - \eta_W \nabla_W \mathcal{L}(W^k, S^k, \theta^k)$
5: $\quad W^{k+1} \leftarrow \text{prox}_{\eta_W \mu||\cdot||_*}(\hat{W}^{k+1})$
6: $\quad \hat{S}^{k+1} \leftarrow S^k - \eta_S \nabla_S ||A - S^k - ZW^{k+1}Z^T||$
7: $\quad S^{k+1} \leftarrow \text{prox}_{\eta_S \alpha||\cdot||_1}(\hat{S}^{k+1})$
8: $\quad ZW^{k+1}Z^T \leftarrow P_{\mathcal{D}}(ZW^{k+1}Z^T)$
9: $\quad$ **for** i=1 to τ **do**
10: $\quad\quad g \leftarrow \frac{\partial \mathcal{L}_{GNN}(\theta^k, ZW^{k+1}Z^T, X, C_L)}{\partial \theta^k}$
11: $\quad\quad \theta^{k+1} \leftarrow \theta^k - \eta_\theta g$
12: $\quad$ **end for**
13: **end while**
14: **return** W, S, θ

4.4 Optimization

In this section we provide the detail of our algorithm for solving Eq. 17 (LRGNN) and Eq. 19 (LRGNN(S)). Since Eq. 17 and Eq. 19 are very similar, we mainly introduce LRGNN. Overall, joint optimization over W, S, and θ is challenging, and we apply the idea of Alternating Direction Method of Multipliers (ADMM) [31]. The key insight of ADMM is to divide the objective into two (or more) pieces and optimize with respect to one of them at each time while keeping other pieces fixed. The algorithm is shown in Alg. 1.

We initialize W, S, θ and Z in line 1 and line 2. To update W, we fix S and θ, and use FBS algorithm by executing line 4 to line 5 where $\mathcal{L}(W, S, \theta)$ is represented as

$$\mathcal{L}(W, S, \theta) = \frac{1}{2}||A - S - ZWZ^T||_F^2 + \gamma\mathcal{L}_{GNN}(\theta, ZWZ^T, X, C_L). \tag{20}$$

Similarly, we fix W and θ, and update S by line 6 and line 7. It should be noted that the proximal operator of l_1 norm and nuclear norm are

$$\text{prox}_{\eta\mu||\cdot||_1}(S) = sgn(S) \odot (|S| - \eta\alpha)_+, \tag{21}$$

$$\text{prox}_{\eta\mu||\cdot||_*}(W) = Udiag((\sigma_i - \eta\mu)_+)_i V^T, \tag{22}$$

where $W = Udiag(\sigma_1, ..., \sigma_n)V^T$ is the singular value decomposition. $sgn(X)$ and $(X)_+$ are element-wise operation of matrix X. $sgn(X_{ij}) = 1, 0, -1$ if $X_{ij} > 0, = 0, < 0$ respectively, and $(X_{ij})_+ = max\{X_{ij}, 0\}$. We let $\odot$ denote the Hadamard product of matrices. To update θ, we also fix W and S. We project the learned adjacency

matrix ZWZ^T to $P_{\mathcal{D}}(ZWZ^T)$ with respect to $ZWZ^T > 0$ and update θ via stochastic gradient descent by iteratively executing line 9 to line 11 for τ times. By iteratively performing line 3 through line 13 until the stopping condition is met, we acquire the final W, S and θ.

For LRGNN(S), we need to initialize the partially observed matrix A, L and S as vectors a, l and s according to the mapping function $\mathcal{M}$ where $l_{\mathcal{M}((i,j))} = Z_i W Z_j^T, \forall (i,j) \in \Omega$. And other optimization procedures are similar to Alg. 1.

4.5 Complexity Analysis

We mainly discuss the space and time complexity of our algorithms excluding the GNN part, as different GNN methods could be integrated with ours.

LRGNN. The space complexity of LRGNN is $O(n^2)$. The time complexity of LRGNN is $O(nr^2 + n^2 r + r^3)$. Compared to the time complexity $O(n^3)$ for minimizing the rank of L, LRGNN is much more computationally efficient.

LRGNN(S). The space complexity of LRGNN(S) is $O(|\Omega|)$. The time complexity of LRGNN(S) is $O(|\Omega| r^2 + r^3)$.

5 EXPERIMENTS

In this section, we evaluate the effectiveness of LRGNN and LRGNN(S) against different graph adversarial attacks. In particular, we aim to answer the following questions:

- **RQ1** How do LRGNN and LRGNN(S) compare to the state-of-the-art defense methods under different adversarial attacks?
- **RQ2** Can our proposed method achieve a tradeoff between effectiveness and efficiency?
- **RQ3** Can our proposed method decouple the adversarial perturbations from the graph?
- **RQ4** How does different components affect the performance of LRGNN?
- **RQ5** Is the initialization of Z in Thm. 4.2 necessary?

5.1 Experimental settings

5.1.1 Dataset. Following [17, 18, 21], we evaluate LRGNN on four benchmark datasets Cora [32], Citeseer [33, 34], Polblogs[35], and Pumbed [34] for semi-supervised node classification. Table 1 gives the statistics of the largest connected component of each graph. None of the referenced attacks/defenses [13–18, 21] use a larger dataset.

5.1.2 Baselines. We compare our approach with the state-of-the-art defenses [13, 15–17] against structure attacks. *RGCN* [13] models the hidden-layer features as Gaussian variables and assigns fewer weights to nodes with higher variance in aggregating neighbors' information. *GCN-Jaccard* [15] uses the Jaccard similarity on the attributes to eliminate edges connecting dissimilar nodes. Note that this method only works when node features are available. *GCN-SVD* [16] found *nettack* affects the high-rank part of the graph spectrum and performed a low-rank approximation of the adjacency matrix with truncated SVD. *Pro-GNN* [17] utilizes the sparse nature of graph and feature smoothness to learn clean graph structure. Further, we compare robustness to the general-purpose GCN [3], Graph Attention Network (GAT) [4], and SimP-GCN [14]. Particularly, SimP-GCN fights against adversarial attacks by preserving feature smoothness between connected nodes. We implement with the repository DeepRobust [36].

Table 1: Statistics of the largest connected component of the used datasets.

	Nodes	Edges	Features	Classes
Cora	2485	5069	1433	7
Citeseer	2110	3668	3703	6
Polblogs	1222	16714	/	2
Pubmed	19717	44338	500	3

5.1.3 Hyperparameters. We use two-layer GCNs with default parameters, which are consistent among all models. Following prior works, we randomly split the nodes of each dataset into three parts — training set (10%), validation set (10%), and testing set (80%). For a fair comparison, we use the same splitting rule across all methods and set the weight decay for all architectures to 5×10^{-4}. And other hyperparameters are tuned based on loss and accuracy on the validation sets. For RGCN, the number of hidden units is tuned among $\{16, 32, 64, 128\}$. For GCN-Jaccard, the similarity threshold is chosen from $\{0.01, 0.02, 0.03, 0.04, 0.05, 0.1\}$. For GCN-SVD, we select the truncated rank from $\{5, 10, 15, 50, 100, 200\}$. For other baselines, we adopt the default parameters in their open-sourced codes. For LRGNN and LRGNN(S), we select r of W from $\{50, 100, 200, 300, 400, 500, 1000\}$ for Cora, Citeseer and Polblogs, and $\{50, 100, 200\}$ for Pubmed. For LRGNN(S), the number of observed entries are set as $k|\mathcal{E}|$ where k is chosen from $\{1, 10, 20, 40\}$ for Cora, Citeseer and Polblogs, and $\{1, 2, 3, 4, 5, 10\}$ for Pubmed.

5.2 Robustness

We aim to answer the first question. For a fair comparison, we follow the settings in [17] to evaluate our proposed method on node classification task against three different types of attacks, i.e., non-targeted attack, targeted attack and random attack. Non-targeted attack (*metattack*) poisons the graph and degrades the overall performance of GNNs. Targeted attack (*nettack*) injects perturbations on graph to attack targeted nodes. Random attack can be viewed as random noise since it randomly injects edges into the graph.

We attack the original graphs respectively with the above methods, and evaluate the node classification performance of all methods on the attacked graphs.

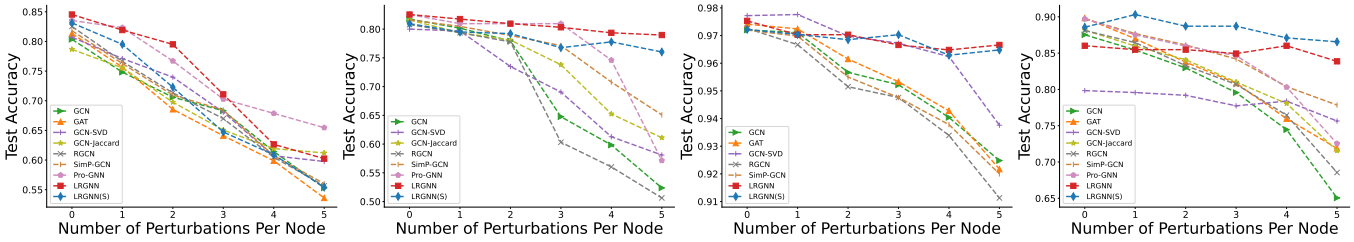
5.2.1 Against Non-targeted Adversarial Attack. We adopt *metattack* and use its different variants on different datasets. For Cora, Citeseer and Polblogs, Meta-Self is applied to achieve the most destructive attacks; while for Pubmed, A-Meta-Self is adopted to save memory and time. And the perturbation rate of edges is set from 0 to 25% with a step size of 5%. All the experiments run 10 times, and we report the average accuracy with standard deviation in Table 2. From the table, we observe that LRGNN and LRGNN(S) outperform most defense methods. Specifically, on Cora, LRGNN improves over GCN by 11% at a 15% perturbation rate and is the closest competitor to the best method. For Citeseer, LRGNN is still comparable to the most

Table 2: Node classification performance (Accuracy±Std) under non-targeted attack (*metattack*).

Dataset	Ptb Rate (%)	GCN†	GAT†	RGCN†	GCN-Jaccard†§	GCN-SVD†	Pro-GNN†§	SimP-GCN	LRGNN	LRGNN(S)
Cora	0	83.50±0.44	83.97±0.65	83.09±0.44	82.05±0.51	80.63±0.45	82.98±0.23	83.69±0.45	83.42±0.30	**84.10±0.48**
	5	76.55±0.79	80.44±0.74	77.42±0.39	79.13±0.59	78.39±0.54	**82.27±0.45**	79.03±1.22	80.90±0.84	79.05±1.00
	10	70.39±1.28	75.61±0.59	72.22±0.38	75.16±0.76	71.47±0.83	**79.03±0.59**	75.74±1.66	77.47±0.87	75.43±1.56
	15	65.10±0.71	69.78±1.28	66.82±0.39	71.03±0.64	66.69±1.18	76.40±1.27	72.65±2.94	**76.73±0.58**	72.60±1.73
	20	59.56±2.72	59.94±0.92	59.27±0.37	65.71±0.89	58.94±1.13	**73.32±1.56**	70.11±6.39	72.86±0.93	70.26±1.02
	25	47.53±1.96	54.78±0.74	50.51±0.78	60.82±1.08	52.06±1.19	69.72±1.69	66.41±7.36	**70.11±1.00**	66.40±2.66
Citeseer	0	71.96±0.55	73.26±0.83	71.20±0.83	72.10±0.63	70.65±0.32	73.28±0.69	**74.25±0.66**	73.13±0.33	71.88±0.50
	5	70.88±0.62	72.89±0.83	70.50±0.43	70.51±0.97	68.84±0.72	72.93±0.57	**73.67±0.63**	72.78±0.58	68.29±0.82
	10	67.55±0.89	70.63±0.48	67.71±0.30	69.54±0.56	68.87±0.62	72.51±0.75	**73.07±1.37**	72.11±1.23	67.24±0.74
	15	64.52±1.11	69.02±1.09	65.69±0.37	65.95±0.94	63.26±0.96	72.03±1.11	**73.09±1.46**	71.18±0.60	65.84±0.86
	20	62.03±3.49	61.04±1.52	62.49±1.22	59.30±1.40	58.55±1.09	70.02±2.28	**70.08±3.55**	66.11±0.76	64.01±0.75
	25	56.94±2.09	61.85±1.12	55.35±0.66	59.89±1.47	57.18±1.87	68.95±2.78	**71.30±2.45**	63.60±0.60	63.25±1.10
Polblogs	0	95.69±0.38	95.35±0.20	95.22±0.14	-	95.31±0.18	-	**95.81±0.40**	94.50±0.23	95.33±0.39
	5	73.07±0.80	83.69±1.45	74.34±0.19	-	89.09±0.22	-	72.97±2.20	**93.33±0.33**	92.41±1.36
	10	70.72±1.13	76.32±0.85	71.04±0.34	-	81.24±0.49	-	72.40±2.51	88.15±0.66	**89.96±0.85**
	15	64.96±1.91	68.80±1.14	67.28±0.38	-	68.10±3.73	-	67.54±2.92	86.22±1.38	**90.72±1.22**
	20	51.27±1.23	51.50±1.63	59.89±0.34	-	57.33±3.15	-	57.33±3.49	83.39±0.61	**87.45±0.95**
	25	49.23±1.36	51.19±1.49	56.02±0.56	-	48.66±9.93	-	56.40±2.87	75.53±1.06	**85.27±1.80**
Pubmed	0	87.19±0.09	83.73±0.40	86.16±0.18	87.06±0.06	83.44±0.21	87.26±0.23	**87.45±0.16**	83.41±0.10	86.45±0.10
	5	83.09±0.13	78.00±0.44	81.08±0.20	86.39±0.06	83.41±0.15	**87.23±0.13**	86.33±0.23	83.35±0.01	86.43±0.12
	10	81.21±0.09	74.93±0.38	77.51±0.27	85.70±0.07	83.27±0.21	**87.21±0.13**	85.73±0.28	83.17±0.09	86.32±0.14
	15	78.66±0.12	71.13±0.51	73.91±0.25	84.76±0.08	83.10±0.18	**87.20±0.15**	85.43±0.30	82.85±0.12	86.24±0.11
	20	77.35±0.19	68.21±0.96	71.18±0.31	83.88±0.05	83.01±0.22	**87.15±0.15**	85.27±0.32	82.50±0.12	86.19±0.12
	25	75.50±0.17	65.41±0.77	67.95±0.15	83.66±0.06	82.72±0.18	**86.76±0.19**	85.20±0.33	81.64±0.21	86.04±0.11

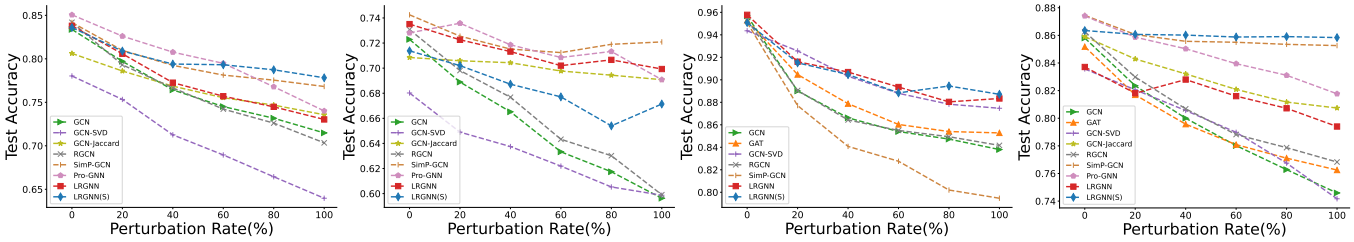
†indicates that the results were taken from [17].

§GCN-jaccard and Pro-GNN cannot be directly applied to datasets without node features.



(a) Cora **(b) Citeseer** **(c) Polblogs** **(d) Pubmed**

Figure 4: The defense performance against targeted attack (*nettack*).



(a) Cora **(b) Citeseer** **(c) Polblogs** **(d) Pubmed**

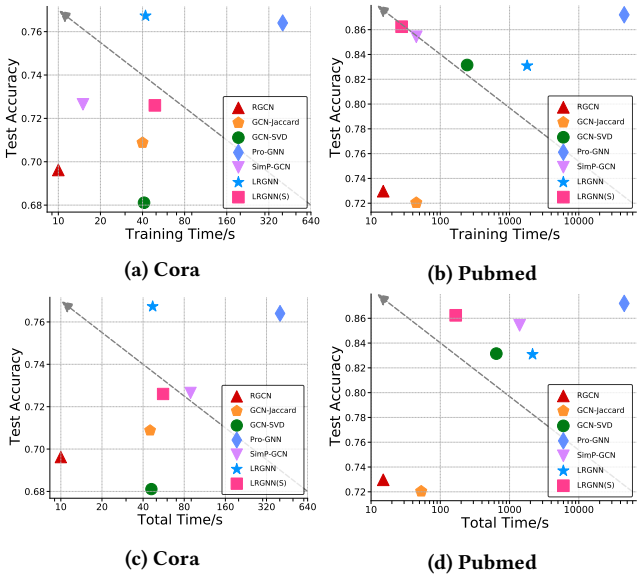
Figure 5: The defense performance against random attack.

effective method when the perturbation ratio is no more than 15%. For Polblogs, LRGNN(S) consistently outperforms other methods under different perturbation rates and improves over vanilla GCN by 36%. In particular, it outperforms other defense methods by 29% at the perturbation rate of 25%. For Pubmed, LRGNN(S) is still competitive with no more than 1% accuracy difference to the most effective method.

As for LRGNN and LRGNN(S), we find that LRGNN(S) consistently outperforms LRGNN on large datasets. This may be because eliminating adversarial attacks on existing edges is more effective on large graphs.

Table 3: GPU memory consumption (/MB) on Cora and Pubmed for all defenses.

Baseline	RGCN	GCN-Jaccard	GCN-SVD	Pro-GNN	SimP-GCN	LRGNN	LRGNN(S)
Cora	2090	1696	1720	2530	2050	2030	1970
Pubmed	11032	1746	3204	29290	8024	22788	2508



(a) Cora (b) Pubmed

(c) Cora (d) Pubmed

Figure 6: Training time (upper) and total time (bottom).

5.2.2 Against Targeted Adversarial Attack. In this experiment, *nettack* is adopted to attack the targeted nodes as we use the default parameter settings in [21]. Following [13, 17], we vary the perturbation number of each targeted node from 1 to 5 with a step size of 1. Nodes with degrees larger than 10 in the testing set are chosen as targeted nodes. Only 10% of those nodes are used in Pubmed while other datasets use all as targeted nodes. The classification performance is displayed in Figure 4. We can see that our proposed methods consistently surpass most baselines. In particular, LRGNN(S) has stable performance against targeted attacks on Pubmed. With the increase of the perturbation number, LRGNN(S) improves over GCN by 24% and over other baselines by 10% under 5 perturbations per targeted node.

5.2.3 Against Random Attack. In this experiment, we study how well LRGNN and LRGNN(S) perform against random noise. We set the perturbation rate from 0 to 100% with a step size of 20%. As shown in Figure 5, the result shows that LRGNN and LRGNN(S) consistently outperform most baselines and successfully defend the random attack. Moreover, LRGNN(S) is quite stable in the face of random attacks on Pubmed.

In conclusion, LRGNN and LRGNN(S) can defend against various types of adversarial attacks, while LRGNN(S) has a more stable performance on large datasets under different perturbation rates.

5.3 Efficiency and Scalability Analysis

To analyze the efficiency and scalability of the proposed method and answer the second question, we show the time cost and memory consumption of all defenses on Cora and Pubmed, respectively. All experiments are done on an NVIDIA A100 GPU and train for epochs recorded in their implementations with a patience of 300. For LRGNN(S), we set $k = 60$ and $k = 1$ for Cora and Pubmed respectively, as chosen by grid search. In Figure 6, we can observe that the training time of LRGNN is 10x faster than Pro-GNN while acquiring similar defense performance and is comparable to other baselines. Meanwhile, the training time of LRGNN(S) is 100x faster than Pro-GNN on Pubmed with similar robustness performance. As for the total time, LRGNN(S) and GCN-SVD are much slower than their training time due to the singular value decomposition in preprocessing, but LRGNN(S) is still much faster than other baselines.

Moreover, we record the memory consumption of all methods in Table 3. We can see that LRGNN has a similar memory cost with Pro-GNN as they both directly optimize on the adjacency matrix. For GCN-Jaccard, its memory consumption is close to GCN as their difference mainly lies in the preprocessing where dissimilar edges are deleted. Since singular value decomposition depends on the full adjacency matrix, GCN-SVD has a higher memory cost than GCN-Jaccard. For RGCN and SimP-GCN, their memory footprints are also large on Pubmed. By contrast, LRGNN(S) only requires one-tenth memory consumption of Pro-GNN on Pubmed, as LRGNN(S) learns the low-rank matrix with partial observations of the adjacency matrix. Hence LRGNN(S) is highly scalable to large inputs.

5.4 Decoupling Analysis

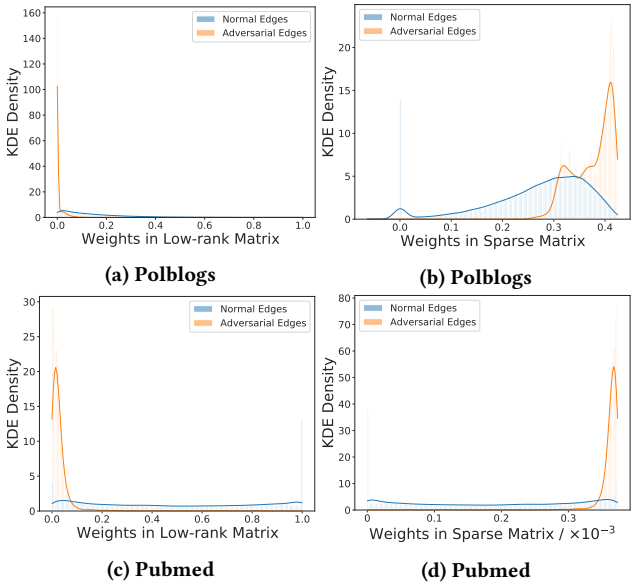
In previous sections, we have demonstrated that the proposed method can effectively fight against various types of attacks efficiently. In this section, we aim to understand the capability of the sparse component S and answer the third question.

5.4.1 Normal Edges Against Adversarial Edges. To decouple the clean subgraph C and the perturbed subgraph P, the sparse matrix S should absorb perturbations in P in the optimization. Meanwhile, most adversarial attacks are edge insertion according to the experiment in [18]. Thus, we investigate the difference in weights between normal and adversarial edges in the learned low-rank adjacency matrix L and the sparse component S.

We depict the weight density distribution of normal and adversarial edges for L and S in Figure 7. The X axis represents the weight of edges in L or S as Y axis represents the KDE density of the number of edges within a specified range of weights. Due to the limit of space, we only show the results on Polblogs for LRGNN and Pubmed for LRGNN(S) under *metattack* with a 15% perturbation rate. According to the results, we can see that the proposed method can learn a clean low-rank adjacency matrix L, where the weights of adversarial edges are significantly reduced and can be clearly distinguished from the normal edges. Meanwhile, the weights of adversarial edges are indeed isolated into the sparse component S as the weights of adversarial edges are mostly larger than those of normal edges. Hence it demonstrates that the proposed method

Table 4: Classification performance with or without component S on Cora dataset.

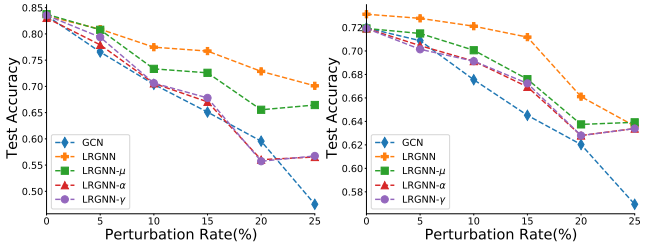
Ptb Rate (%)	0	5	10	15	20	25
w/o S	83.25±0.21	80.03±0.72	75.97±1.02	74.60±0.25	68.77±0.25	67.96±1.32
w/ S	**83.42±0.30**	**80.90±0.95**	**77.47±0.87**	**76.73±0.58**	**72.86±0.93**	**70.11±1.00**



(a) Polblogs (b) Polblogs

(c) Pubmed (d) Pubmed

Figure 7: Weight density distributions of normal and adversarial edges on L and S.



(a) Cora (b) Citeseer

Figure 8: Ablation study on different datasets.

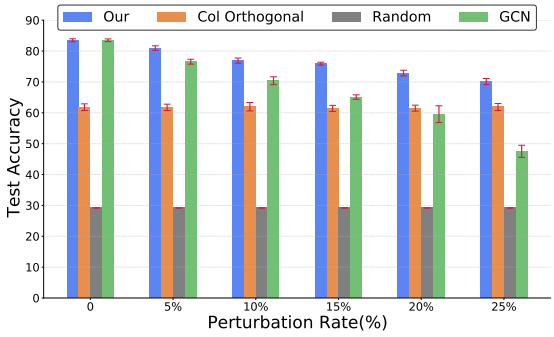


Figure 9: Comparision on different initialization methods.

can decouple P and C by separating the perturbed edges from the normal ones.

5.4.2 Performance With/Without Sparse Component. To further demonstrate the effectiveness of the sparse component S, we record the classification performance against *metattack* on the Cora dataset with and without S. If the model does not have the sparse component, we will remove S from the objective function, and only optimize over W and θ. Under this circumstance, LRGNN optimizes on a classical matrix rank minimization problem. We report the performance in Table 4. As shown in the table, we observe that the performance of LRGNN with S is consistently better than those without S. Meanwhile, the gap gradually enlarges with the increase of the perturbation rate. This may be because the perturbations are no longer unnoticeable, and no longer remain as higher-rank components when the perturbation rate is high. Hence in our method, S can absorb larger perturbations to make the normal edges distinguishable from the adversarial ones, leading to more robust matrix rank minimization.

5.5 Ablation Study

We evaluate different components of LRGNN and answer the fourth question. According to predefined hyperparameters μ, α and γ in the objective function, we can transform the model into three different variants, i.e., LRGNN-μ, LRGNN-α and LRGNN-γ. For each variant, we set other predefined hyperparameters to 0. Meanwhile, we use grid search to choose the best parameters for each variant. The best results are recorded in Figure 8. We can tell that, the low-rank component is the most effective part of LRGNN as LRGNN-μ dramatically outperforms other variants. The performance of LRGNN-α and LRGNN-γ are very close, meaning that the sparse component S alone cannot effectively decouple the perturbations without the low-rank part, suggesting that posing a regularizer over the adjacency matrix does not work.

5.6 Initialization Analysis

Appropriate initialization of Z is important in our proposed method as we hope to get close to the condition of Thm. 4.2 as much as possible. Hence we study how different initialization methods affect the overall performance to answer the fifth question. Specifically, we utilize three different ways to initialize Z for LRGNN, i.e., our initialization, column orthogonalization and random initialization. The best r is selected by grid search and independent of the rank of A. For ours, we set Z as the top-r left singular vectors of the perturbed adjacency matrix A by Thm. 4.2. For column orthogonalization, we choose Z that $Z^T Z = I_{r\times r}$. For random initialization, we randomly generate a $n \times r$ matrix as Z following the Gaussian distribution. All experiments are conducted on Cora with *metattack*. The results are shown in Figure 9. We can observe that LRGNN initialized with our method excels all other methods. This may be because column orthogonalization satisfies $Z^T Z = I_{r\times r}$ but violates $Z^T A Z = \Sigma_r$, it can fight against *metattack* only when perturbation rate is more than 20%. As for random initialization, it is consistently

weaker than the other two methods. Observations from Figure 9 demonstrate that appropriate initialization is crucial in our proposed method, and empirically prove that ours acquires robust defense performance even if A is not rank r.

6 CONCLUSION

Graph neural networks are vulnerable to adversarial attacks as demonstrated in recent studies. Previous methods either fail to obtain robustness, or is of high complexity. To speed up robust structure learning, we propose a novel framework LRGNN and its scalable variant LRGNN(S) with fast and robust matrix rank minimization. A range of experiments sufficiently demonstrates the effectiveness and efficiency of our proposed method. Most importantly, LRGNN(S) achieves similar robustness performance to the best baseline on Pubmed, a large-scale dataset, in defending against *metattack*. But it is 100× faster than the previous method and is 12× memory-saving. Future directions include extending our framework to directed graphs and incorporating node attributes further.

7 ACKNOWLEDGMENTS

This work was partially supported by National Key R&D Program of China (No.2018AAA0101202), NSF China under Grant (No.61902245, 62032020, 61960206002, 42050105, 61829201), and the Science and Technology Innovation Program of Shanghai (No.19YF1424500).

REFERENCES

[1] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[2] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3837–3845, 2016.

[3] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[4] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[5] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[6] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[7] Saurabh Verma and Zhi-Li Zhang. Stability and generalization of graph convolutional neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1539–1548, 2019.

[8] Bin Lu, Xiaoying Gan, Haiming Jin, Luoyi Fu, and Haisong Zhang. Spatiotemporal adaptive gated graph convolution network for urban traffic flow forecasting. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1025–1034, 2020.

[9] Yangtao Wang, Yanzhao Xie, Yu Liu, Ke Zhou, and Xiaocui Li. Fast graph convolution network based multi-label image recognition via cross-modal fusion. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1575–1584, 2020.

[10] Chenghu Zhou, Hua Wang, Chengshan Wang, Zengqian Hou, Zhiming Zheng, Shuzhong Shen, Qiuming Cheng, Zhiqiang Feng, Xinbing Wang, Hairong Lv, et al. Prospects for the research on geoscience knowledge graph in the big data era. *Science China Earth Sciences*, pages 1–11, 2021.

[11] Lichao Sun, Yingtong Dou, Carl Yang, Ji Wang, Philip S Yu, Lifang He, and Bo Li. Adversarial attack and defense on graph data: A survey. *arXiv preprint arXiv:1812.10528*, 2018.

[12] Wei Jin, Yaxin Li, Han Xu, Yiqi Wang, and Jiliang Tang. Adversarial attacks and defenses on graphs: A review and empirical study. *arXiv preprint arXiv:2003.00653*, 2020.

[13] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1399–1407, 2019.

[14] Wei Jin, Tyler Derr, Yiqi Wang, Yao Ma, Zitao Liu, and Jiliang Tang. Node similarity preserving graph convolutional networks. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 148–156, 2021.

[15] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples on graph data: Deep insights into attack and defense. *arXiv preprint arXiv:1903.01610*, 2019.

[16] Negin Entezari, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis. All you need is low (rank) defending against adversarial attacks on graphs. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 169–177, 2020.

[17] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 66–74, 2020.

[18] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[19] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):1–37, 2011.

[20] Namrata Vaswani and Praneeth Narayanamurthy. Static and dynamic robust pca and matrix completion: A review. *Proceedings of the IEEE*, 106(8):1359–1379, 2018.

[21] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2847–2856, 2018.

[22] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *International conference on machine learning*, pages 1115–1124. PMLR, 2018.

[23] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. Topology attack and defense for graph neural networks: An optimization perspective. In *IJCAI*, pages 3961–3967. ijcai.org, 2019.

[24] Shiqian Ma, Donald Goldfarb, and Lifeng Chen. Fixed point and bregman iterative methods for matrix rank minimization. *Mathematical Programming*, 128(1):321–353, 2011.

[25] Karthik Mohan and Maryam Fazel. Iterative reweighted algorithms for matrix rank minimization. *The Journal of Machine Learning Research*, 13(1):3441–3473, 2012.

[26] Zihan Zhou, Xiaodong Li, John Wright, Emmanuel Candes, and Yi Ma. Stable principal component pursuit. In *2010 IEEE international symposium on information theory*, pages 1518–1522. IEEE, 2010.

[27] Shiqian Ma and Necdet Serhat Aybat. Efficient optimization algorithms for robust principal component analysis and its variants. *Proceedings of the IEEE*, 106(8):1411–1426, 2018.

[28] Luong Trung Nguyen, Junhan Kim, and Byonghyo Shim. Low-rank matrix completion: A contemporary survey. *IEEE Access*, 7:94215–94237, 2019.

[29] Emmanuel J Candes and Yaniv Plan. Matrix completion with noise. *Proceedings of the IEEE*, 98(6):925–936, 2010.

[30] Tom Goldstein, Christoph Studer, and Richard G. Baraniuk. A field guide to forward-backward splitting with a FASTA implementation. *CoRR*, abs/1411.3406, 2014.

[31] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[32] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

[33] C Lee Giles, Kurt D Bollacker, and Steve Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pages 89–98, 1998.

[34] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[35] Lada A Adamic and Natalie Glance. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43, 2005.

[36] Yaxin Li, Wei Jin, Han Xu, and Jiliang Tang. Deeprobust: A pytorch library for adversarial attacks and defenses. *arXiv preprint arXiv:2005.06149*, 2020.