GRAND+: Scalable Graph Random Neural Networks

Wenzheng Feng[1], Yuxiao Dong[1], Tinglin Huang[1], Ziqi Yin[2], Xu Cheng[13]
Evgeny Kharlamov[4], Jie Tang[1]
[1]Tsinghua University, [2]Beijing Institute of Technology
[3]Tencent Inc., [4]Bosch Center for Artificial Intelligence
fwz17@mails.tsinghua.edu.cn,{yuxiaod,tl091,jietang}@tsinghua.edu.cn,ziqiyin18@gmail.com
alexcheng@tencent.com,evgeny.kharlamov@de.bosch.com

ABSTRACT

Graph neural networks (GNNs) have been widely adopted for semi-supervised learning on graphs. A recent study shows that the graph random neural network (GRAND) model can generate state-of-the-art performance for this problem. However, it is difficult for GRAND to handle large-scale graphs since its effectiveness relies on computationally expensive data augmentation procedures. In this work, we present a scalable and high-performance GNN framework GRAND+ for semi-supervised graph learning. To address the above issue, we develop a generalized forward push (GFPush) algorithm in GRAND+ to pre-compute a general propagation matrix and employ it to perform graph data augmentation in a mini-batch manner. We show that both the low time and space complexities of GFPush enable GRAND+ to efficiently scale to large graphs. Furthermore, we introduce a confidence-aware consistency loss into the model optimization of GRAND+, facilitating GRAND+'s generalization superiority. We conduct extensive experiments on seven public datasets of different sizes. The results demonstrate that GRAND+ 1) is able to scale to large graphs and costs less running time than existing scalable GNNs, and 2) can offer consistent accuracy improvements over both full-batch and scalable GNNs across all datasets.

CCS CONCEPTS

• **Computing methodologies** → **Semi-supervised learning settings**; • **Information systems** → Social networks.

KEYWORDS

Graph Neural Networks; Scalability; Semi-Supervised Learning

ACM Reference Format:
Wenzheng Feng, Yuxiao Dong, Tinglin Huang, Ziqi Yin, Xu Cheng, Evegeny Kharlamov, Jie Tang. 2022. GRAND+: Scalable Graph Random Neural Networks. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3485447.3512044

The code is available at https://github.com/wzfhaha/GRAND-plus.
Jie Tang is the corresponding author.

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512044

1 INTRODUCTION

Graph structure is a commonplace of both our physical and virtual worlds, such as social relationships, chemical bonds, and information diffusion. The inherit incompleteness of the real-world graph data sparks enormous interests in the problem of semi-supervised learning on graphs [19, 33]. To date, graph neural networks (GNNs) have been considered by many as the *de facto* way to address this problem [1, 11, 19, 28, 29]. Briefly, GNNs leverage the graph structure among data samples to facilitate model predictions, enabling them to produce prominent performance improvements over traditional semi-supervised learning methods [32].

However, there are remaining challenges for GNN-based semi-supervised learning solutions. Notably, the generalization of GNNs usually does not form their strengths, as most of them only use a supervised loss to learn parameters [9, 19, 28, 29]. This setup makes the model prone to overfit the limited labeled samples, thereby degrading the prediction performance over unseen samples. To overcome this issue, the graph random neural network (GRAND) [12] designs graph data augmentation and consistency regularization strategies for GNNs. These designs enable it to bring significant performance gains over existing GNNs for semi-supervised learning on Cora, Citeseer and Pubmed.

Specifically, GRAND develops the random propagation operation to generate effective structural data augmentations. It is then trained with both the supervised loss on labeled nodes and the consistency regularization loss on different augmentations of unlabeled nodes. To achieve a good graph augmentation, random propagation in GRAND proposes to use a mixed-order adjacency matrix to propagate the feature matrix. The propagation essentially requires the power iteration of the adjacency matrix at every training step, making it computationally challenging to scale GRAND to large-scale graphs.

To address this issue, we present the GRAND+ framework for large-scale semi-supervised learning on graphs. GRAND+ is a scalable GNN consistency regularization method. In GRAND+, we introduce efficient approximation techniques to perform random propagation in a mini-batch manner, addressing the scalability limitation of GRAND. Furthermore, we improve GRAND by adopting a confidence-aware loss for regulating the consistency between different graph data augmentations. This design stabilizes the training process and provides GRAND+ with good generalization. Specifically, GRAND+ comprises the following techniques:

- *Generalized feature propagation*: We propose a generalized mixed-order matrix to perform random feature propagation. Such matrix

offers a set of tunable weights to control the importance of different orders of neighborhoods and thus offers a flexible mechanism for dealing with complex real-world graphs.
- *Efficient approximation*: Inspired by recent matrix approximation based GNNs [5, 8], GRAND+ adopts an approximation method—Generalized Forward Push (GFPush)—to efficiently calculate the generalized propagation matrix. This enables GRAND+ to perform random propagation and model learning in a mini-batch manner, offering the model with significant scalability.
- *Confidence-aware loss*: We design a confidence-aware loss for the GRAND+ regularization framework. This helps filter out potential noises during the consistency training by ignoring highly uncertain unlabeled samples, thus improving the generalization performance of GRAND+.

We conduct comprehensive experiments on seven public graph datasets with different genres and scales to demonstrate the performance of GRAND+. Overall, GRAND+ yields the best classification results compared to ten GNN baselines on three benchmark datasets and surpasses five representative scalable GNNs on the other four relatively large datasets with efficiency benefits. For example, GRAND+ achieves a state-of-the-art accuracy of 85.0% on Pubmed. On MAG-Scholar-C with 12.4 million nodes, GRAND+ is about 10 fold faster than FastGCN and GraphSAINT, and offers a 4.9% accuracy gain over PPRGo—previously the fastest method on this dataset—with a comparable running time.

2 SEMI-SUPERVISED GRAPH LEARNING

2.1 Problem

In graph-based semi-supervised learning, the data samples are organized as a graph $G = (V, E)$, where each node $v \in V$ represents a data sample and $E \in V \times V$ is a set of edges that denote the relationships between each pair of nodes. We use $\mathbf{A} \in \{0,1\}^{|V|\times|V|}$ to represent G's adjacency matrix, with each element $\mathbf{A}(s,v) = 1$ indicating that there exists an edge between s and v, otherwise $\mathbf{A}(s,v) = 0$. $\mathbf{D}$ is the diagonal degree matrix where $\mathbf{D}(s,s) = \sum_v \mathbf{A}(s,v)$. $\widetilde{G}$ is used to denote the graph G with added self-loop connections. The corresponding adjacency matrix is $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and the degree matrix is $\widetilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}$.

In this work, we focus on the classification problem, in which each sample s is associated with 1) a feature vector $\mathbf{X}_s \in \mathbf{X} \in \mathbb{R}^{|V|\times d_f}$ and 2) a label vector $\mathbf{Y}_s \in \mathbf{Y} \in \{0,1\}^{|V|\times C}$ with C representing the number of classes. In the semi-supervised setting, only limited nodes $L \in V$ have observed labels ($0 < |L| \ll |V|$), and the labels of remaining nodes $U = V - L$ are unseen. The objective of semi-supervised graph learning is to infer the missing labels $\mathbf{Y}_U$ for unlabeled nodes U based on graph structure G, node features $\mathbf{X}$, and the observed labels $\mathbf{Y}_L$[1].

2.2 Related Work

Graph neural networks (GNNs) have been widely adopted for addressing the semi-supervised graph learning problem. In this part, we review the progress of GNNs with an emphasis on their large-scale solutions to semi-supervised graph learning.

[1]For a matrix $\mathbf{M} \in \mathbb{R}^{a\times b}$, we use $\mathbf{M}_i \in \mathbb{R}^b$ to denote its i-th row vector and let $\mathbf{M}(i,j)$ represent the element of the i-th row and the j-th column.

Graph Convolutional Network. The graph convolutional network (GCN) [19] generalizes the convolution operation into graphs. Specifically, the l-th GCN layer is defined as:

$$\mathbf{H}^{(l+1)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}), \tag{1}$$

where $\mathbf{H}^{(l)}$ denotes the hidden node representations of the l-th layer with $\mathbf{H}^{(0)} = \mathbf{X}$, $\hat{\mathbf{A}} = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}$ is the symmetric normalized adjacency matrix of $\widetilde{G}$, $\mathbf{W}^{(l)}$ denotes the weight matrix of the l-th layer, and $\sigma(\cdot)$ denotes the activation function. In practice, this graph convolution procedure would be repeated multiple times, and the final representations are usually fed into a logistic regression layer for classification.

Simplified Graph Convolution. By taking a closer look at Equation 1, we can observe that graph convolution consists of two operations: *feature propagation* $\hat{\mathbf{A}}\mathbf{H}^{(l)}$ and *non-linear transformation* $\sigma(\cdot)$. Wu et al. [29] simplify this procedure by removing the non-linear transformations in hidden layers. The resulting simplified graph convolution (SGC) is formulated as:

$$\hat{\mathbf{Y}} = \text{softmax}(\hat{\mathbf{A}}^N\mathbf{X}\mathbf{W}), \tag{2}$$

where $\hat{\mathbf{A}}^N\mathbf{X}$ is considered as a simplified N-layer graph convolutions on $\mathbf{X}$, $\mathbf{W}$ refers to the learnable weight matrix for classification, and $\hat{\mathbf{Y}}$ denotes the model's predictions.

GNNs with Mixed-Order Propagation. As pointed by Li et al. [23], $\hat{\mathbf{A}}^N\mathbf{X}$ will converge to a fix point as N increases according to the Markov chain convergence theorem, namely, the over-smoothing issue. To address it, a typical kind of methods [5, 20, 21] suggest to use a more complex mixed-order matrix for feature propagation. For example, APPNP [20] adopts the truncated personalized PageRank (ppr) matrix $\mathbf{\Pi}^{\text{ppr}}_{\text{sym}} = \sum_{n=0}^{N} \alpha(1-\alpha)^n\hat{\mathbf{A}}^n$, where the hyperparameter $\alpha \in (0,1]$ denotes the teleport probability, allowing the model to preserve the local information even when $N \to +\infty$.

Scalable GNNs. Broadly, there are three categories of methods proposed for making GNNs scalable: 1) The node sampling methods employ sampling strategies to speed up the recursive feature aggregation procedure. The representative methods include GraphSAGE [14], FastGCN [7], and LADIES [34]; 2) The graph partition methods attempt to divide the original large graph into several small sub-graphs and run GNNs on sub-graphs. This category consists of Cluster-GCN [10] and GraphSAINT [31]; 3) The matrix approximation methods follow the design of SGC [29] to decouple feature propagation and non-linear transformation, and to utilize some approximation methods to accelerate feature propagation. The proposed GRAND+ framework is highly related to matrix approximation based methods such as PPRGo [5] and GBP [8]. We will analyze their differences in Section 3.6.

3 THE GRAND+ FRAMEWORK

In this section, we briefly review the graph random neural network (GRAND) and present its scalable solution GRAND+ for large-scale semi-supervised graph learning.

3.1 The Graph Random Neural Network

Recently, Feng et al. [12] introduce the graph neural neural network (GRAND) for semi-supervised node classification. GRAND is a GNN

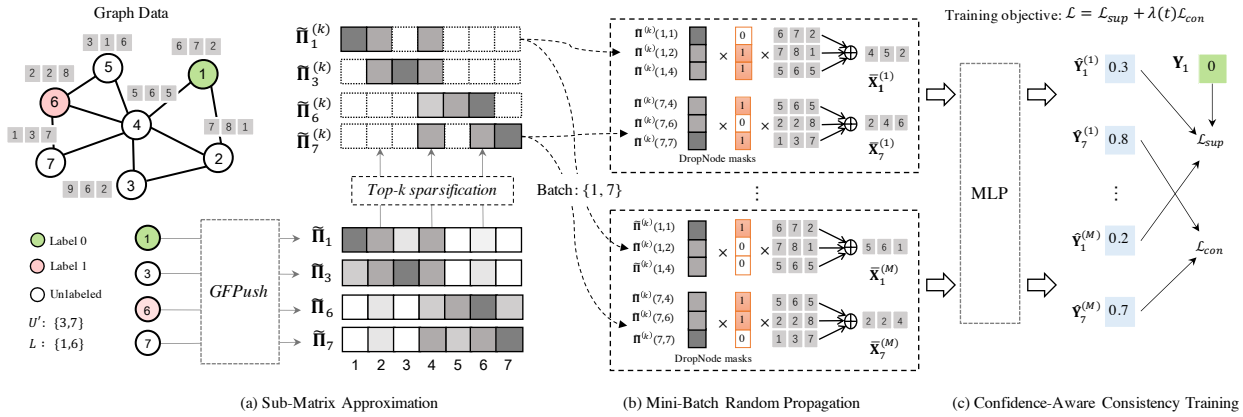


Figure 1: Illustration of GRAND+. (a) GRAND+ adopts *Generalized Forward Push* (*GFPush*) and *Top-k sparsification* to approximate the corresponding rows of propagation matrix $\mathbf{\Pi}$ for nodes in $L \cup U'$. (b) The obtained sparsified row approximations are then used to perform mini-batch random propagation to generate augmentations for nodes in the batch. (c) Finally, the calculated feature augmentations are fed into an MLP to conduct confidence-aware consistency training, which employs both supervised loss $\mathcal{L}_{sup}$ and confidence-aware consistency loss $\mathcal{L}_{con}$ for model optimization.

consistency regularization framework that optimizes the prediction consistency of unlabeled nodes in different augmentations.

Specifically, it designs *random propagation*—a mixed-order propagation strategy—to achieve graph data augmentations. First, the node features $\mathbf{X}$ are randomly dropped with DropNode—a variant of dropout. Then the resultant corrupted feature matrix is propagated over the graph with a mixed-order matrix. Instead of the PPR matrix, GRAND uses an average pooling matrix $\mathbf{\Pi}_{\text{sym}}^{\text{avg}} = \sum_{n=0}^{N} \hat{\mathbf{A}}^n/(N+1)$ for propagation. Formally, the random propagation strategy is formulated as:

$$\overline{\mathbf{X}} = \mathbf{\Pi}_{\text{sym}}^{\text{avg}} \cdot \text{diag}(\mathbf{z}) \cdot \mathbf{X}, \quad \mathbf{z}_i \sim \text{Bernoulli}(1-\delta), \tag{3}$$

where $\mathbf{z} \in \{0,1\}^{|V|}$ denotes the random DropNode masks drawn from Bernoulli$(1-\delta)$, and δ represents DropNode probability. In doing so, the dropped information of each node is compensated by its neighborhoods. Under the homophily assumption of graph data, the resulting matrix $\overline{\mathbf{X}}$ can be seen as an effective data augmentation of the original feature matrix $\mathbf{X}$. Owing to the randomness of DropNode, this method can in theory generate exponentially many augmentations for each node.

In each training step of GRAND, the random propagation procedure is performed for M times, leading to M augmented feature matrices $\{\overline{\mathbf{X}}^{(m)} | 1 \le m \le M\}$. Then all the augmented feature matrices are fed into an MLP to get M predictions. During optimization, GRAND is trained with both the standard classification loss on labeled data and an additional *consistency regularization* loss [4] on the unlabeled node set U, that is,

$$\frac{1}{M \cdot |U|} \sum_{s \in U} \sum_{m=1}^{M} \left\| \hat{\mathbf{Y}}_s^{(m)} - \overline{\mathbf{Y}}_s \right\|_2^2, \quad \overline{\mathbf{Y}}_s = \sum_{m=1}^{M} \frac{1}{M} \hat{\mathbf{Y}}_s^{(m)}, \tag{4}$$

where $\hat{\mathbf{Y}}_s^{(m)}$ is MLP's prediction probability for node s when using $\overline{\mathbf{X}}_s^{(m)}$ as input. The consistency loss provides an additional regularization effect by enforcing the neural network to give similar predictions for different augmentations of unlabeled data. With random propagation and consistency regularization, GRAND achieves better generalization capability over conventional GNNs [12].

Scalability of GRAND. In practice, the n-th power of the adjacency matrix $\hat{\mathbf{A}}^n$ is computationally infeasible when n is large [25]. To avoid this issue, GRAND adopts the power iteration to directly calculate the entire augmented feature matrix $\overline{\mathbf{X}}$ (in Equation 3), i.e., iteratively calculating and summing up the product of $\hat{\mathbf{A}}$ and $\hat{\mathbf{A}}^n \cdot \text{diag}(\mathbf{z}) \cdot \mathbf{X}$ for $0 \le n < N$. This procedure is implemented with the sparse-dense matrix multiplication and has a linear time complexity of $O(|V| + |E|)$. However, it needs to be performed for M times at *every* training step to generate different feature augmentations. Thus the total complexity of T training steps becomes $O(T \cdot M \cdot (|V| + |E|))$, which is prohibitively expensive when dealing with large graphs.

3.2 Overview of GRAND+

We present GRAND+ to achieve both scalability and accuracy for graph based semi-supervised learning. It follows the general consistency regularization principle of GRAND and comprises techniques to make it scalable to large graphs while maintaining GRAND's flexibility and generalization capability.

Briefly, instead of propagating features with power iteration, we develop an efficient approximation algorithm—generalized forward push (GFPush)—in GRAND+ to pre-compute the required row vectors of propagation matrix and perform random propagation in a mini-batch manner. The time complexity of this procedure is controlled by a predefined hyperparameter, avoiding the scalability limitation faced by GRAND. Furthermore, GRAND+ adopts a new confidence-aware loss for consistency regularization, which makes the training process more stable and leads to better generalization performance than GRAND.

Propagation Matrix. In GRAND+, we propose the following generalized mixed-order matrix for feature propagation:

$$\mathbf{\Pi} = \sum_{n=0}^{N} w_n \cdot \mathbf{P}^n, \quad \mathbf{P} = \widetilde{\mathbf{D}}^{-1}\widetilde{\mathbf{A}}, \tag{5}$$

where $\sum_{n=0}^{N} w_n = 1$ and $w_n \ge 0$, $\mathbf{P}$ is the row-normalized adjacency matrix. Different from the propagation matrices used in GRAND

and other GNNs, the form of $\mathbf{\Pi}$ adopts a set of tunable weights $\{w_n | 0 \leq n \leq N\}$ to fuse different orders of adjacency matrices. By adjusting w_n, GRAND+ can flexibly manipulate the importance of different orders of neighborhoods to suit the diverse graphs of distinct structural properties in the real world.

Training Pipeline. To achieve fast training, GRAND+ abandons the power iteration method which directly calculates the entire augmented feature matrix $\overline{\mathbf{X}}$, and instead computes each augmented feature vector separately for each node. Ideally, the augmented feature vector $\overline{\mathbf{X}}_s$ of node s is calculated by:

$$\overline{\mathbf{X}}_s = \sum_{v \in \mathcal{N}_s^{\pi}} \mathbf{z}_v \cdot \mathbf{\Pi}(s, v) \cdot \mathbf{X}_v, \quad \mathbf{z}_v \sim \text{Bernoulli}(1 - \delta). \tag{6}$$

Here we use $\mathbf{\Pi}_s$ to denote the row vector of $\mathbf{\Pi}$ corresponding to node s, $\mathcal{N}_s^{\pi}$ is used to represent the indices of non-zero elements of $\mathbf{\Pi}_s$, $\mathbf{\Pi}(s, v)$ denotes the v-th element of $\mathbf{\Pi}_s$. This paradigm allows us to generate augmented features for only a batch of nodes in each training step, and thus enables us to use efficient mini-batch gradient descent for optimization.

However, it is difficult to calculate the exact form of $\mathbf{\Pi}_s$ in practice. To address this problem, we develop several efficient methods to approximate $\mathbf{\Pi}_s$ in GRAND+. The approximation procedure consists of two stages. In the first stage, we propose an efficient method *Generalized Forward Push* (*GFPush*) to compute an error-bounded approximation $\widetilde{\mathbf{\Pi}}_s$ for the row vector $\mathbf{\Pi}_s$. In the second stage, we adopt a *top-k sparsification* strategy to truncate $\widetilde{\mathbf{\Pi}}_s$ to only contain the top k largest elements. The obtained sparsified row approximation $\widetilde{\mathbf{\Pi}}_s^{(k)}$ is used to calculate $\overline{\mathbf{X}}_s$ as a substitute of $\mathbf{\Pi}_s$ (Eq. 6). For efficiency, it is required to pre-compute the corresponding row approximations for all nodes used in training. In addition to labeled nodes, GRAND+ also requires unlabeled nodes to perform consistency regularization during training. To further improve efficiency, instead of using the full set of U, GRAND+ samples a smaller subset of unlabeled nodes $U' \subseteq U$ for consistency regularization. As illustrated in Figure 1, the training pipeline of GRAND+ consists of three steps:

- *Sub-matrix approximation.* We obtain a sparsified row approximation $\mathbf{\Pi}_s^{(k)}$ for each node $s \in L \cup U'$ through GFPush and top-k sparsification. The resultant sparsified sub-matrix is used to support random propagation.
- *Mini-batch random propagation.* At each training step, we sample a batch of nodes from $L \cup U'$ and generate multiple augmentations for each node in the batch with the approximated row vector.
- *Confidence-aware consistency training.* We feed the augmented features into an MLP to get corresponding predictions and optimize the model with both supervised loss and confidence-aware consistency loss.

3.3 Sub-Matrix Approximation

Generalized Forward Push (GFPush). It can be observed that the row-normalized adjacency matrix $\mathbf{P} = \widetilde{\mathbf{D}}^{-1}\widetilde{\mathbf{A}}$ is also the reverse random walk transition probability matrix [8] on $\widetilde{G}$, where row vector $\mathbf{P}_s$ denotes random walk transition probabilities starting from node s. Based on this fact, we propose an efficient algorithm called *Generalized Forward Push* (*GFPush*) to approximate row vector $\mathbf{\Pi}_s = \sum_{n=0}^{N} w_n \mathbf{P}_s^n$ with a bounded error. GFPush is inspired by the *Forward Push* [2] algorithm for approximating personalized PageRank vector, while has much higher flexibility with the ability to approximate the generalized mixed-order matrix $\mathbf{\Pi}$. The core idea of GFPush is to simulate an N-step random walk probability diffusion process from s with a series of pruning operations for acceleration. To achieve that, we should maintain a pair of vectors at each step n ($0 \leq n \leq N$): 1) *Reserve vector* $\mathbf{q}^{(n)} \in \mathbb{R}^{|V|}$, denoting the probability masses reserved at step n; 2) *Residue vector* $\mathbf{r}^{(n)} \in \mathbb{R}^{|V|}$, representing the probability masses to be diffused beyond step n.

Algorithm 1 shows the pseudo-code of GFPush. At beginning, $\mathbf{r}^{(0)}$ and $\mathbf{q}^{(0)}$ are both initialized as the indicator vector $\mathbf{e}^{(s)}$ where $\mathbf{e}_s^{(s)} = 1$ and $\mathbf{e}_v^{(s)} = 0$ for $v \neq s$, meaning the random walk starts from s with the probability mass of 1. Other reserve and residue vectors (i.e., $\mathbf{r}^{(n)}$ and $\mathbf{q}^{(n)}, 1 \leq n \leq N$) are set to $\vec{0}$. Then the algorithm iteratively updates reserve and residue vectors with N steps. In the n-th iteration, the algorithm conducts a *push* operation (Line 5–9 of algorithm 1) for node v which satisfies $\mathbf{r}_v^{(n-1)} > \mathbf{d}_v \cdot r_{max}$. Here $\mathbf{d}_v = \widetilde{\mathbf{D}}(v, v)$ represents the degree of v, r_{max} is a predefined threshold. In the push operation, the residue $\mathbf{r}_v^{(n-1)}$ of v is evenly distributed to its neighbors, and the results are stored into the n-th residue vector $\mathbf{r}^{(n)}$. Meanwhile, the reserve vector $\mathbf{q}^{(n)}$ is also updated to be identical with $\mathbf{r}^{(n)}$. After finishing the push operation on v, we reset $\mathbf{r}_v^{(n-1)}$ to 0 to avoid duplicate updates.

To gain more intuition of this procedure, we could observe that $\mathbf{r}_v^{(n-1)}/\mathbf{d}_v$ is the conditional probability that a random walk moves from v to a neighboring node u, conditioned on it reaching v with probability $\mathbf{r}_v^{(n-1)}$ at the previous step. Thus each push operation on v can be seen as a one-step random walk probability diffusion process from v to its neighborhoods. To ensure efficiency, GFPush only conducts push operations for node v whose residue value is greater than $\mathbf{d}_v \cdot r_{max}$. Thus when the n-th iteration is finished, $\mathbf{q}^{(n)}$ can be seen as an approximation of the n-step random walk transition vector $\mathbf{P}_s^n$. And $\widetilde{\mathbf{\Pi}}_s = \sum_{n=0}^{N} w_n \mathbf{q}^{(n)}$ is accordingly considered as the approximation of $\mathbf{\Pi}_s$ as returned by the algorithm.

Theoretical Analysis. We have the following theorem about the bounds of time complexity, memory complexity, and approximation error of GFPush.

Theorem 1. *Algorithm 1 has $O(N/r_{max})$ time complexity and $O(N/r_{max})$ memory complexity, and returns $\widetilde{\mathbf{\Pi}}_s$ as an approximation of $\mathbf{\Pi}_s$ with the L_1 error bound: $\| \mathbf{\Pi}_s - \widetilde{\mathbf{\Pi}}_s \|_1 \leq N \cdot (2|E| + |V|) \cdot r_{max}$.*

Proof. See Appendix A.2. □

Theorem 1 suggests that the approximation precision and running cost of GFPush are negatively correlated with r_{max}. In practice, we could use r_{max} to control the trade-off between efficiency and approximation precision.

Top-k Sparsification. To further reduce training cost, we perform top-k sparsification for $\widetilde{\mathbf{\Pi}}_s$. In this procedure, only the top-k largest elements of $\widetilde{\mathbf{\Pi}}_s$ are preserved and other entries are set to 0. Hence the resultant sparsified transition vector $\widetilde{\mathbf{\Pi}}_s^{(k)}$ has at most k non-zero elements. In this way, the model only considers the k most

Algorithm 1: GFPush

Input : Self-loop augmented graph $\widetilde{G}$, propagation step N, node s, threshold r_{max}, weight coefficients w_n, $0 \le n \le N$.
Output : An approximation $\widetilde{\mathbf{\Pi}}_s$ of transition vector $\mathbf{\Pi}_s$ of node s.

$\mathbf{r}^{(n)} \leftarrow \vec{0}$ for $n = 1, ..., N$; $\mathbf{r}^{(0)} \leftarrow \mathbf{e}^{(s)}$ ($\mathbf{e}_s^{(s)} = 1, \mathbf{e}_v^{(s)} = 0$ for $v \neq s$).
$\mathbf{q}^{(n)} \leftarrow \vec{0}$ for $n = 1, ..., N$; $\mathbf{q}^{(0)} \leftarrow \mathbf{e}^{(s)}$.
for $n = 1 : N$ **do**
  while *there exists node* v *with* $\mathbf{r}_v^{(n-1)} > \mathbf{d}_v \cdot r_{max}$ **do**
    for *each* $u \in \mathcal{N}_v$ **do**
      /* $\mathcal{N}_v$ is the neighborhood set of v in graph $\widetilde{G}$. */
      $\mathbf{r}_u^{(n)} \leftarrow \mathbf{r}_u^{(n)} + \mathbf{r}_v^{(n-1)} / \mathbf{d}_v$.
      $\mathbf{q}_u^{(n)} \leftarrow \mathbf{r}_u^{(n)}$.
    end
    $\mathbf{r}_v^{(n-1)} \leftarrow 0$.
    /* Perform a push operation on v. */
  end
end
$\widetilde{\mathbf{\Pi}}_s \leftarrow \sum_{n=0}^{N} w_n \cdot \mathbf{q}^{(n)}$.
return $\widetilde{\mathbf{\Pi}}_s$.

important neighborhoods for each node in random propagation, which is still expected to be effective based on the clustering assumption [6]. Similar technique was also adopted by PPRGo [5]. We will empirically examine the effects of k in Section 4.5.

Parallelization. In GRAND+, we need to approximate row vectors for all nodes in $L \cup U'$. It could be easily checked that different row approximations are calculated independently with each other in GFPush. Thus we can launch multiple workers to approximate multiple vectors simultaneously. This procedure is implemented with multi-thread programming in our implementation.

3.4 Mini-Batch Random Propagation

GRAND+ adopts the sparsified row approximations of $\mathbf{\Pi}$ to perform random propagation in a mini-batch manner. Specifically, at the t-th training step, we randomly sample a batch of labeled nodes L_t from L, and a batch of unlabeled nodes U_t from U'. Then we calculate augmented feature vector $\overline{\mathbf{X}}_s$ for node $s \in L_t \cup U_t$ by:

$$\overline{\mathbf{X}}_s = \sum_{v \in \mathcal{N}_s^{(k)}} \mathbf{z}_v \cdot \widetilde{\mathbf{\Pi}}^{(k)}(s, v) \cdot \mathbf{X}_v, \quad \mathbf{z}_v \sim \text{Bernoulli}(1 - \delta), \tag{7}$$

where $\mathcal{N}_s^{(k)}$ denotes the non-zero indices of $\widetilde{\mathbf{\Pi}}_s^{(k)}$, $\mathbf{X}_v \in \mathbb{R}^{d_f}$ is feature vector of node v. At each training step, we generate M augmented feature vectors $\{\overline{\mathbf{X}}_s^{(m)} | 1 \le m \le M\}$ by repeating this procedure for M times. Let $b = |L_t| + |U_t|$ denote the batch size. Then the time complexity of each batch is bounded by $O(k \cdot b \cdot d_f)$.

Random Propagation for Learnable Representations. In Equation 7, the augmented feature vector $\overline{\mathbf{X}}_s$ is calculated with raw features $\mathbf{X}$. However, in some real applications (e.g., image or text classification), the dimension of $\mathbf{X}$ might be extremely large, which will incur a huge cost for calculation. To mitigate this issue, we can employ a linear layer to transform each $\mathbf{X}_v$ to a low-dimensional hidden representation $\mathbf{H}_v \in \mathbb{R}^{d_h}$ firstly, and then perform random propagation with $\mathbf{H}$:

$$\overline{\mathbf{X}}_s = \sum_{v \in \mathcal{N}_s^{(k)}} \mathbf{z}_v \cdot \widetilde{\mathbf{\Pi}}^{(k)}(s, v) \cdot \mathbf{H}_v, \quad \mathbf{H}_v = \mathbf{X}_v \cdot \mathbf{W}^{(0)}, \tag{8}$$

where $\mathbf{W}^{(0)} \in \mathbb{R}^{d_f \times d_h}$ denotes learnable transformation matrix. In this way, the computational complexity of this procedure is reduced to $O(k \cdot b \cdot d_h)$, where $d_h \ll d_f$ denotes the dimension of $\mathbf{H}_v$.

Prediction. During training, the augmented feature vector $\overline{\mathbf{X}}_s^{(m)}$ is fed into an MLP model to get the corresponding outputs:

$$\hat{\mathbf{Y}}_s^{(m)} = \text{MLP}(\overline{\mathbf{X}}_s^{(m)}, \mathbf{\Theta}), \tag{9}$$

where $\hat{\mathbf{Y}}_s^{(m)} \in [0, 1]^C$ denotes the prediction probabilities of s. $\mathbf{\Theta}$ represents MLP's parameters.

3.5 Confidence-Aware Consistency Training

GRAND+ adopts both supervised classification loss and consistency regularization loss to optimize model parameters during training. The supervised loss is defined as the average cross-entropy over multiple augmentations of labeled nodes:

$$\mathcal{L}_{sup} = -\frac{1}{|L_t| \cdot M} \sum_{s \in L_t} \sum_{m=1}^{M} \mathbf{Y}_s \cdot \log(\hat{\mathbf{Y}}_s^{(m)}). \tag{10}$$

Confidence-Aware Consistency Loss. Inspired by recent advances in semi-supervised learning [4], GRAND adopts an additional consistency loss to optimize the prediction consistency of multiple augmentations of unlabeled data, which is shown to be effective in improving generalization capability. GRAND+ also follows this idea, while adopts a new confidence-aware consistency loss to further improve effectiveness.

Specifically, for node $s \in U_t$, we first calculate the distribution center by taking the average of its M prediction probabilities, i.e., $\overline{\mathbf{Y}}_s = \sum_{m=1}^{M} \hat{\mathbf{Y}}_s^{(m)} / M$. Then we apply *sharpening* [17] trick over $\overline{\mathbf{Y}}_s$ to "guess" a pseudo label $\widetilde{\mathbf{Y}}_s$ for node s. Formally, the guessed probability on the j-th class of node s is obtained via:

$$\widetilde{\mathbf{Y}}(s, j) = \overline{\mathbf{Y}}(s, j)^{\frac{1}{\tau}} / \sum_{c=0}^{C-1} \overline{\mathbf{Y}}(s, c)^{\frac{1}{\tau}}, \tag{11}$$

where $0 < \tau \le 1$ is a hyperparameter to control the sharpness of the guessed pseudo label. As τ decreases, $\widetilde{\mathbf{Y}}_s$ is enforced to become sharper and converges to a one-hot distribution eventually. Then the confidence-aware consistency loss on unlabeled node batch U_t is defined as:

$$\mathcal{L}_{con} = \frac{1}{|U_t| \cdot M} \sum_{s \in U_t} \mathbb{I}(\max(\overline{\mathbf{Y}}_s) \ge \gamma) \sum_{m=1}^{M} \mathcal{D}(\widetilde{\mathbf{Y}}_s, \hat{\mathbf{Y}}_s^{(m)}), \tag{12}$$

where $\mathbb{I}(\max(\overline{\mathbf{Y}}_s) \ge \gamma)$ is an indicator function which outputs 1 if $\max(\overline{\mathbf{Y}}_s) \ge \gamma$ holds, and outputs 0 otherwise. $0 \le \gamma < 1$ is a predefined threshold. $\mathcal{D}(p, q)$ is a distance function which measures the distribution discrepancy between p and q. Here we mainly consider two options for $\mathcal{D}$: *L_2 distance* and *KL divergence*.

Compared with the consistency loss used in GRAND (Cf. Equation 4), the biggest advantage of $\mathcal{L}_{con}$ is that it only considers "highly confident" unlabeled nodes determined by threshold τ in optimization. This mechanism could reduce the potential training noise by filtering out uncertain pseudo-labels, further improving model's performance in practice. Combining $\mathcal{L}_{con}$ and $\mathcal{L}_{sup}$, the final loss for model optimization is defined as:

$$\mathcal{L} = \mathcal{L}_{sup} + \lambda(t) \mathcal{L}_{con}, \tag{13}$$

Algorithm 2: GRAND+

Input : Graph G, feature matrix $\mathbf{X} \in \mathbb{R}^{|V| \times d_f}$, labeled node set L, unlabeled node set U and observed labels $\mathbf{Y}_L \in \mathbb{R}^{|L| \times C}$.
Output : Classification probabilities $\hat{\mathbf{Y}}^{(inf)}$.

1 Sample a subset of unlabeled nodes U' from U.
2 **for** $s \in L \cup U'$ **do**
3 $\widetilde{\mathbf{\Pi}}_s \leftarrow \text{GFPush}(G, s)$.
4 Obtain $\widetilde{\mathbf{\Pi}}_s^{(k)}$ by applying top-k sparsification on $\widetilde{\mathbf{\Pi}}_s$.
  /* Approximating row vectors with pallalization. */
5 **end**
6 **for** $t = 0 : T$ **do**
7 Sample a batch of labeled nodes $L_t \subseteq L$ and a batch of unlabeled nodes $U_t \subseteq U'$.
8 **for** $s \in L_t \cup U_t$ **do**
9 **for** $m = 1 : M$ **do**
10 Generate augmented feature vector $\overline{\mathbf{X}}_s^{(m)}$ with Equation 7.
11 Predict class distribution with $\hat{\mathbf{Y}}_s^{(m)} = \text{MLP}(\overline{\mathbf{X}}_s^{(m)}, \mathbf{\Theta})$.
12 **end**
13 **end**
14 Compute $\mathcal{L}_{sup}$ via Equation 10 and $\mathcal{L}_{con}$ via Equation 12.
15 Update the parameters $\mathbf{\Theta}$ by mini-batch gradients descending: $\mathbf{\Theta} = \mathbf{\Theta} - \eta \nabla_{\mathbf{\Theta}}(\mathcal{L}_{sup} + \lambda \mathcal{L}_{con})$.
  /* Stop training with early-stopping. */
16 **end**
17 Infer classification probabilities $\hat{\mathbf{Y}}^{(inf)} = \text{MLP}(\mathbf{\Pi} \cdot (1-\delta) \cdot \mathbf{X}, \mathbf{\Theta})$.
18 **return** $\hat{\mathbf{Y}}^{(inf)}$.

where $\lambda(t)$ is a linear warmup function [13] which increases linearly from 0 to the maximum value λ_{max} as training step t increases.

Model Inference. After training, we need to infer the predictions for unlabeled nodes. GRAND+ adopts power iteration to calculate the exact prediction results for unlabeled nodes during inference:

$$\hat{\mathbf{Y}}^{(inf)} = \text{MLP}(\sum_{n=0}^{N} w_n (\widetilde{\mathbf{D}}^{-1}\widetilde{\mathbf{A}})^n \cdot (1-\delta) \cdot \mathbf{X}, \mathbf{\Theta}), \tag{14}$$

where we rescale $\mathbf{X}$ with $(1-\delta)$ to make it identical with the expectation of the DropNode perturbed features used in training. Note that unlike GRAND, the above power iteration process only needs to be performed once in GRAND+, and the computational cost is acceptable in practice. Compared with obtaining predictions with GFPush as done in training, this inference strategy could provide more accurate predictions in theory. Algorithm 2 shows the entire training and inference procedure of GRAND+.

3.6 Model Analysis

Complexity Analysis. We provide detailed analyses for the time complexities of GRAND+'s different learning stages. According to Theorem 1, the complexity of approximation stage (Line 2–5 of Algorithm 2) is $O((|U'| + |L|) \cdot N / r_{max})$. As for the training stage (Line 6–15 of Algorithm 2), the total complexity of T training steps is $O(k \cdot b \cdot M \cdot T)$, which is practically efficient for large graphs since b and k are usually much smaller than the graph size. The complexity of inference stage (Line 17 of Algorithm 2) is $O((|V| + |E|) \cdot N)$, which is linear with the sum of node and edge counts.

GRAND+ vs. PPRGo and GBP. Similar with GRAND+, PPRGo [5] and GBP [8] also adopt matrix approximation methods to scale GNNs. However, GRAND+ differs from the two methods in several key aspects. PPRGo scales up APPNP by using Forward Push [2] to approximate the ppr matrix. Compared with PPRGo, GRAND+ is more flexible in real applications thanks to the adopted generalized propagation matrix $\mathbf{\Pi}$ and GFPush algorithm. GBP also owns this merit by using the generalized PageRank matrix [22] for feature propagation. However, it directly approximates the propagation results of raw features through bidirectional propagation [3], whose computational complexity is linear with the raw feature dimension, rendering it difficult to handle datasets with high-dimensional features. Moreover, different from PPRGo and GBP designed for the general supervised classification problem, GRAND+ makes significant improvements for semi-supervised setting by adopting random propagation and consistency regularization to enhance generalization capability.

Table 1: Dataset statistics.

Dataset	Nodes	Edges	Classes	Features
Cora	2,708	5,429	7	1,433
Citeseer	3,327	4,732	6	3,703
Pubmed	19,717	44,338	3	500
AMiner-CS	593,486	6,217,004	18	100
Reddit	232,965	11,606,919	41	602
Amazon2M	2,449,029	61,859,140	47	100
MAG-Scholar-C	10,541,560	265,219,994	8	2,784,240

4 EXPERIMENTS

4.1 Experimental Setup

Baselines. In our experiments, we compare GRAND+ with five state-of-the-art full-batch GNNs—GCN [19], GAT [28], APPNP [20], GCNII [9] and GRAND [12], as well as five representative scalable GNNs—FastGCN [7], GraphSAINT [31], SGC [29], GBP [8] and PPRGo [5]. For GRAND+, we implement three variants with different settings for propagation matrix $\mathbf{\Pi}$ (Cf. Equation 5):

- GRAND+ (P): *Truncated ppr matrix* $\mathbf{\Pi}^{\text{ppr}} = \sum_{n=0}^{N} \alpha(1-\alpha)^n \mathbf{P}^n$.
- GRAND+ (A): *Average pooling matrix* $\mathbf{\Pi}^{\text{avg}} = \sum_{n=0}^{N} \mathbf{P}^n / (N+1)$.
- GRAND+ (S): *Single order matrix* $\mathbf{\Pi}^{\text{single}} = \mathbf{P}^N$.

Datasets. The experimnents are conducted on seven public datasets of different scales, including three widely adopted benchmark graphs—Cora, Citeseer and Pubmed [30], and four relatively large graphs—AMiner-CS [12], Reddit [14], Amazon2M [10] and MAG-scholar-C [5]. For Cora, Citeseer and Pubmed, we use public data splits [19, 28, 30]. For AMiner-CS, Reddit, Amazon2M and MAG-Scholar-C, we use 20×#classes nodes for training, 30×#classes nodes for validation and the remaining nodes for test. The corresponding statistics are summarized in Table 1. More details for the setup and reproducibility can be found in Appendix A.1.

4.2 Results on Benchmark Datasets

To evaluate the effectiveness of GRAND+, we compare it with 10 GNN baselines on Cora, Citeseer and Pubmed. Following the community convention, the results of baseline models on the three benchmarks are taken from the previous works [9, 12, 28]. For GRAND+, we conduct 100 trials with random seeds and report the average accuracy and the corresponding standard deviation over

Table 2: Classification Accuracy (%) on Benchmarks.

Category	Method	Cora	Citeseer	Pubmed
Full-batch GNNs	GCN	81.5 ± 0.6	71.3 ± 0.4	79.1 ± 0.4
	GAT	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3
	APPNP	84.1 ± 0.3	71.6 ± 0.5	79.7 ± 0.3
	GCNII	85.5 ± 0.5	73.4 ± 0.6	80.3 ± 0.4
	GRAND	85.4 ± 0.4	75.4 ± 0.4	82.7 ± 0.6
Scalable GNNs	FastGCN	81.4 ± 0.5	68.8 ± 0.9	77.6 ± 0.5
	GraphSAINT	81.3 ± 0.4	70.5 ± 0.4	78.2 ± 0.8
	SGC	81.0 ± 0.1	71.8 ± 0.1	79.0 ± 0.1
	GBP	83.9 ± 0.7	72.9 ± 0.5	80.6 ± 0.4
	PPRGo	82.4 ± 0.2	71.3 ± 0.3	80.0 ± 0.4
Our Methods	GRAND+ (P)	**85.8 ± 0.4**	**75.6 ± 0.4**	84.5 ± 1.1
	GRAND+ (A)	85.5 ± 0.4	75.5 ± 0.4	**85.0 ± 0.6**
	GRAND+ (S)	85.0 ± 0.5	74.4 ± 0.5	84.2 ± 0.6

Table 3: Accuracy (%) and Running Time (s) on Large Graphs.

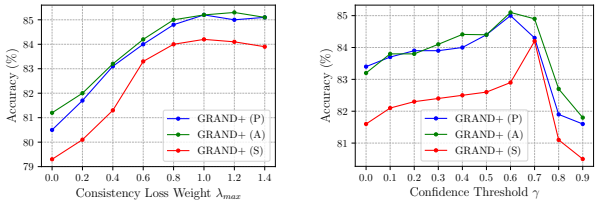
Method	AMiner-CS		Reddit		Amazon2M		MAG.	
	Acc	RT	Acc	RT	Acc	RT	Acc	RT
GRAND	53.1±1.1	750	OOM	–	OOM	–	OOM	–
FastGCN	48.9±1.6	69	89.6±0.6	158	72.9±1.0	239	64.3±5.6	4220
GraphSAINT	51.8±1.3	39	92.1±0.5	39	75.9±1.3	189	75.0±1.7	6009
SGC	50.2±1.2	9	92.5±0.2	31	74.9±0.5	69	–	>24h
GBP	52.7±1.7	21	88.7±1.1	370	70.1±0.9	280	–	>24h
PPRGo	51.2±1.4	11	91.3±0.2	233	67.6±0.5	160	72.9±1.1	434
GRAND+ (P)	53.9±1.8	17	93.3±0.2	183	75.6±0.7	188	77.6±1.2	653
GRAND+ (A)	54.2±1.7	14	**93.5±0.2**	174	75.9±0.7	136	**80.0±1.1**	737
GRAND+ (S)	**54.2±1.6**	10	92.8±0.2	62	**76.2±0.6**	80	77.8±0.9	483

the trials. The results are demonstrated in Table 2. It can be observed that the best GRAND+ variant consistently outperforms all baselines across the three datasets. Notably, GRAND+ (A) improves upon GRAND by a margin of 2.3% (absolute difference) on Pubmed. The improvements of GRAND+ (P) over GRAND on Cora (85.8±0.4 vs. 85.4±0.4) and Citeseer (75.6±0.4 vs. 75.4±0.4) are also statistically significant (p-value $\ll 0.01$ by a t-test). These results suggest the strong *generalization performance* achieved by GRAND+.

4.3 Results on Large Graphs

To justify the scalability of GRAND+, we further compare it with five scalable GNN baselines on four large graphs, i.e., AMiner-CS, Reddit, Amazon2M and MAG-Scholar-C. Note that the feature dimension of MAG-Scholar-C is huge (i.e., 2.8M features per node). To enable GRAND+ to deal with it, a learnable linear layer is added before random propagation to transform the high-dimensional node features to low-dimensional hidden vectors (Cf. Equation 8). For a fair comparison, we conduct careful hyperparameter selection for all methods (Cf. Appendix A.1). We run each model for 10 trails with random splits, and report its average accuracy and average running time (including preprocessing time, training time and inference time) over the trials. The results are summarized in Table 3.

We interpret the results of Table 3 from two perspectives. First, combining with the results in Table 2, we notice that the three variants of GRAND+ exhibit big differences across these datasets:



(a) Accuracy w.r.t. λ_{max}. (b) Accuracy w.r.t. γ

Figure 2: Effects of λ_{max} and γ on Pubmed.

On Cora and Citeseer, GRAND+ (P) achieves better results than GRAND+ (A) and GRAND+ (S); On Pubmed, Reddit and MAG-Scholar-C, GRAND+ (A) surpasses the other two variants; On AMiner-CS and Amazon2M, GRAND+ (S) gets the best classification results. This indicates that the propagation matrix plays a critical role in this task, and further suggests that GRAND+ could *flexibly* deal with different graphs by adjusting the generalized mixed-order matrix $\mathbf{\Pi}$.

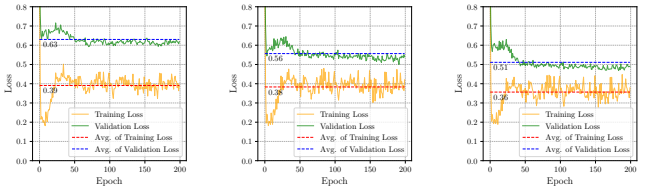
Second, we observe GRAND+ consistently surpasses all baseline methods in accuracy and gets efficient running time on the four datasets. Importantly, on the largest graph MAG-Scholar-C, GRAND+ could succeed in training and making predictions in around 10 minutes, while SGC and GBP require more than 24 hours to finish, because the two methods are designed to directly propagate the high-dimensional raw features in pre-processing step. Compared with FastGCN and GraphSAINT, GRAND+ (S) achieves 8× and 12× acceleration respectively. When compared with PPRGo, the fastest model on this dataset in the past, GRAND+ (S) gets 4.9% improvement in accuracy while with a comparable running time. These results indicate GRAND+ *scales well* on large graphs and further emphasize its *excellent performance*.

We also report the accuracy and running time of GRAND on AMiner-CS. Note that it can not be executed on the other three large datasets due to the out-of-memory error. As we can see, GRAND+ achieves over 40× acceleration in terms of running time over GRAND on AMiner-CS, demonstrating the effectiveness of the proposed approximation techiniques in improving *efficiency*.

4.4 Generalization Improvements

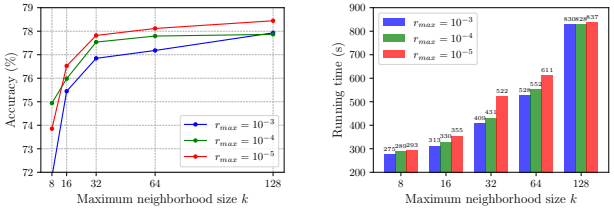
In this section, we quantitatively investigate the benefits of the proposed confidence-aware consistency loss $\mathcal{L}_{con}$ to model's generalization capability. In GRAND+, $\mathcal{L}_{con}$ is mainly dominated by two hyperparameters: *confidence threshold* γ (Cf. Equation 12) and *maximum consistency loss weight* λ_{max} (Cf. Equation 13).

We first analyze the effects of γ and λ_{max} on GRAND+'s classification performance. Specifically, we adjust the values of γ and λ_{max} separately with other hyperparameters fixed, and observe how GRAND+'s accuracy changes on test set. Figure 2 illustrates the results on Pubmed dataset. From Figure 2 (a), it can be seen that the accuracy is significantly improved as λ_{max} increases from 0 to 0.8. When λ_{max} is greater than 0.8, the accuracy tends to be stable. This indicates that the consistency loss could really contribute to GRAND+'s performance. From Figure 2 (b), we observe model's performance benefits from the enlargement of γ when γ is



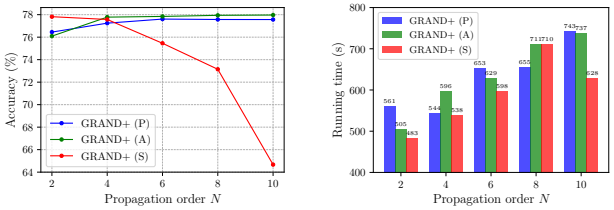
(a) $\lambda_{max} = 0.0$ (b) $\lambda_{max} = 1.0, \gamma = 0.0$ (c) $\lambda_{max} = 1.0, \gamma = 0.6$

Figure 3: Training and Validation Losses on Pubmed.



(a) Classification accuracy. (b) Running time.

Figure 4: GRAND+ w.r.t. k and r_{max} on MAG-Scholar-C.



(a) Classification accuracy. (b) Running time.

Figure 5: Effects of propagation order N on MAG-Scholar-C.

less than 0.7, which highlights the significance of the confidence mechanism. If γ is set too large (i.e., > 0.7), the performance will degrade because too much unlabeled samples are ignored in this case, weakening the effects of consistency regularization.

Figure 2 demonstrates the confidence-aware consistency loss could significantly improve model's performance. We further study its benefits to generalization capability by analysing the cross-entropy losses on training set and validation set. Here we measure model's generalization capability with its *generalization gap* [16]—*the gap between training loss and validation loss*. A smaller generalization gap means the model has a better generalization capability. Figure 3 reports the training and validation losses of GRAND+ (A) on Pubmed. We can observe the generalization gap is rather large when we do not use consistency loss ($\lambda_{max} = 0$) during training, indicating a severe over-fitting issue. And the gap becomes smaller when we change λ_{max} to 1.0. When we set both λ_{max} and γ to proper values (i.e., $\lambda_{max} = 1.0$, $\gamma = 0.6$), the generalization gap further decreases. These observations demonstrate the proposed consistency training and confidence mechanism indeed contribute to GRAND+'s generalization capability.

4.5 Parameter Analysis

Threshold r_{max} and Neighborhood Size k. GRAND+ uses GFPush and top-k sparsification to approximate multiple row vectors of $\mathbf{\Pi}$ to perform mini-batch random propagation (Cf. Section 3.4). The approximation error of this process is mainly influenced by two hyperparameters—threshold r_{max} of GFPush and maximum neighborhood size k for sparsification. We conduct detailed experiments to better understand the effects of k and r_{max} on model's accuracy and running time. Figure 4 illustrates the corresponding results of GRAND+ (S) w.r.t. different values of k and r_{max} on MAG-Scholar-C. As we can see, both the accuracy and running time increase when r_{max} becomes smaller, which is coincident with the conclusion of Theorem 1. While k has an opposite effect—the accuracy and running time are enlarged with the increase of k. Interestingly, as k decreases from 128 to 32, the running time is cut in half with only $\sim 2\%$ performance drop in accuracy. This demonstrates the effectiveness of the top-k sparsification strategy, which could achieve significant acceleration at little cost of accuracy.

Propagation Order N. We study the influence of propagation order N on GRAND+ when using different propagation matrices. Figure 5 presents the classification performance and running time of three GRAND+ variants on MAG-Scholar-C w.r.t. different values of N. As we can see, when $N = 2$, GRAND+ (S) achieves better accuracy and faster running time than GRAND+ (P) and GRAND+ (A). However, as N increases, the accuracy of GRAND+ (S) drops dramatically because of the over-smoothing issue, while GRAND+ (P) and GRAND+ (A) do not suffer from this problem and benefit from a larger propagation order. On the other hand, increasing N will enlarge models' running time. In real applications, we can flexibly adjust the propagation matrix and the value of N to make desired efficiency and effectiveness.

5 CONCLUSION

We propose GRAND+, a scalable and high-performance GNN framework for graph-based semi-supervised learning. The advantages of GRAND+ include both the scalability and generalization capability while the existing state-of-the-art solutions typically feature only one of the two. To this effect, we follow the consistency regularization principle of GRAND in achieving the generalization performance, while significantly extend it to achieve scalability and retain and even exceed the flexibility and generalization capability of GRAND. To achieve these, GRAND+ utilizes a generalized mixed-order matrix for feature propagation, and uses our approximation method generalized forward push (GFPush) to calculate it efficiently. In addition, GRAND+ adopts a new confidence-aware consistency loss to achieve better consistency training. Extensive experiments show that GRAND+ not only gets the best performance on benchmark datasets, but also achieves performance and efficiency superiority over existing scalable GNNs on datasets with millions of nodes. In the future, we would like to explore more accurate approximation methods to accelerate GNNs.

ACKNOWLEDGMENTS

The work is supported by the NSFC for Distinguished Young Scholar (61825602) and Tsinghua-Bosch Joint ML Center.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Hrayr Harutyunyan, Nazanin Alipourfard, Kristina Lerman, Greg Ver Steeg, and Aram Galstyan. 2019. Mixhop: Higher-order graph convolution architectures via sparsified neighborhood mixing. *ICML'19* (2019).

[2] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *FOCS'06*. IEEE, 475–486.

[3] Siddhartha Banerjee and Peter Lofgren. 2015. Fast Bidirectional Probability Estimation in Markov Models. *NeurIPS* (2015).

[4] David Berthelot, Nicholas Carlini, J. Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. 2019. MixMatch: A Holistic Approach to Semi-Supervised Learning. *NeurIPS* (2019), 5050–5060.

[5] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling graph neural networks with approximate pagerank. In *KDD'20*. 2464–2473.

[6] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. 2006. *Semi-supervised learning*. The MIT Press. https://doi.org/10.7551/mitpress/9780262033589.001.0001

[7] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. *ICLR* (2018).

[8] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable Graph Neural Networks via Bidirectional Propagation. *NeurIPS* (2020).

[9] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *ICML*. PMLR, 1725–1735.

[10] 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'19*.

[11] Ming Ding, Jie Tang, and Jie Zhang. 2018. Semi-supervised learning on graphs with generative adversarial nets. *CIKM'18* (2018).

[12] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. 2020. Graph Random Neural Network for Semi-Supervised Learning on Graphs. *NeurIPS* (2020).

[13] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *NeurIPS* (2017), 1025–1035.

[15] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML* (2015).

[16] Shirish Nitish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Tak Peter Ping Tang. 2017. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *ICLR* (2017).

[17] Sohn Kihyuk, Berthelot David, Li Chun-Liang, Zhang Zizhao, Carlini Nicholas, Ekin Cubuk D., Kurakin Alex, Zhang Han, and Raffel Colin. 2020. FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence. *NeurIPS* (2020).

[18] P. Diederik Kingma and Lei Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *ICLR* (2015).

[19] N. Thomas Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. *ICLR* (2017).

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

[21] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion improves graph learning. *arXiv preprint arXiv:1911.05485* (2019).

[22] Pan Li, Eli Chien, and Olgica Milenkovic. 2019. Optimizing generalized pagerank methods for seed-expansion community detection. *arXiv preprint arXiv:1905.10881* (2019).

[23] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI'18*.

[24] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. *ICML* (2013), 1310–1318.

[25] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM'18*. 459–467.

[26] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Paul Bo-June Hsu, and Kuansan Wang. 2015. An Overview of Microsoft Academic Service (MAS) and Applications. *WWW (Companion Volume)* (2015), 243–246.

[27] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *KDD'08*.

[28] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *ICLR* (2018).

[29] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. *ICML* (2019), 6861–6871.

[30] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *ICML* (2016).

[31] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. *ICLR* (2020).

[32] Dengyong Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. *NeurIPS* (2004), 321–328.

[33] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. *ICML* (2003).

[34] 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).

A APPENDIX

A.1 Implementation Note

A.1.1 Running environment. The experiments are conducted on a single Linux server with Intel(R) Xeon(R) CPU Gold 6420 @ 2.60GHz, 376G RAM and 10 NVIDIA GeForce RTX 3090TI-24GB. The Python version is 3.8.5.

A.1.2 Implementation details. We implement GFPush with C++, and use OpenMP to perform parallelization. We use Pytorch to implement the training process of GRAND+, and use pybind[5] to create Python binding for approximation module. In GRAND+ and other baselines, we use BatchNorm [15] and gradient clipping [24] to stabilize the model training, and adopt Adam [18] for optimization.

A.1.3 Dataset details. There are totally 7 datasets used in this paper, that is, Cora, Citeseer, Pubmed, AMiner-CS, Reddit, Amazon2M and MAG-Scholar-C. Our preprocessing scripts for Cora, Citeseer and Pubmed are implemented with reference to the codes of Planetoid[6]. Following the experimental setup used in [19, 28, 30], we run 100 trials with random seeds for the results on Cora, Citeseer and Pubmed reported in Section 4.2. AMiner-CS is constructed by Feng et al. [12] based on the AMiner citation network [27]. In AMiner-CS, each node represents a paper, the edges are citation relations, labels are research topics of papers. Reddit is published by Hamilton et al. [14], in which each node represents a post in the Reddit community, a graph link represents the two posts have been commented by the same user. The task is to predict the category of each post. Amazon2M is published by Chiang et al. [10], where each node is a product, each edge denotes the two products are purchased together, labels represent the categories of products. MAG-Scholar-C is constructed by Bojchevski [5] based on Microsoft Academic Graph (MAG) [26], in which nodes refer to papers, edges represent citation relations among papers and features are bag-of-words of paper abstracts.

For AMiner-CS, Reddit, Amazon2M and MAG-Scholar-C, we use 20×#classes for training, 30×#classes nodes for validation and the remaining nodes for test. For Aminer, Reddit and MAG-Scholar-C, we randomly sample the same number of nodes for each class—20 nodes per class for training and 30 nodes per class for validation. For Amazon2M, we uniformly sample all the training and validation nodes from the whole datasets, as the node counts of some classes are less than 20. For these datasets, we report the average results of 10 trails with random splits.

A.1.4 Hyperparameter Selection. For results in Table 2, we adjust hyperparameters of GRAND+ on validation set, and use the best configuration for prediction, and the results of other baseline methods are taken from previous works [9, 12, 28]. For results in Table 3-5, we conduct detailed hyperparameter search for both GRAND+ and other GNN baselines (i.e., FastGCN, GraphSAINT, SGC, GBP and PPRGo). For each search, we run 3 experiments with random seeds, and select the hyperparameter configuration which gets the best average accuracy on validation set. Then we train model with the selected configuration.

[5]https://github.com/pybind/pybind11
[6]https://github.com/kimiyoung/planetoid

Table 4: Hyperparameter configuration of GRAND+.

		lr	wr	L_m	d_h	r_{max}	k	N	λ_{max}
Cora	GRAND+ (P)	10^{-2}	10^{-3}	2	64	10^{-7}	32	20	1.5
	GRAND+ (A)	10^{-2}	10^{-3}	2	64	10^{-7}	32	4	1.5
	GRAND+ (S)	10^{-2}	10^{-3}	2	64	10^{-7}	32	2	1.5
Citeseer	GRAND+ (P)	10^{-3}	10^{-3}	2	256	10^{-7}	32	10	0.8
	GRAND+ (A)	10^{-3}	10^{-3}	2	256	10^{-7}	32	2	0.8
	GRAND+ (S)	10^{-3}	10^{-3}	2	256	10^{-7}	32	2	0.8
Pubmed	GRAND+ (P)	10^{-2}	10^{-2}	1	-	10^{-5}	16	6	1.0
	GRAND+ (A)	10^{-2}	10^{-2}	1	-	10^{-5}	16	4	1.0
	GRAND+ (S)	10^{-2}	10^{-2}	1	-	10^{-5}	16	2	1.0
AMiner-CS	GRAND+ (P)	10^{-2}	10^{-2}	1	-	10^{-5}	64	6	1.5
	GRAND+ (A)	10^{-2}	10^{-2}	1	-	10^{-5}	64	4	1.5
	GRAND+ (S)	10^{-2}	10^{-2}	1	-	10^{-5}	64	2	1.5
Reddit	GRAND+ (P)	10^{-4}	0	2	512	10^{-5}	64	6	1.5
	GRAND+ (A)	10^{-4}	0	2	512	10^{-5}	64	6	1.5
	GRAND+ (S)	10^{-4}	0	2	512	10^{-7}	64	2	1.5
Amazon2M	GRAND+ (P)	10^{-3}	10^{-5}	2	1024	10^{-6}	64	6	0.8
	GRAND+ (A)	10^{-3}	10^{-5}	2	1024	10^{-6}	64	4	0.8
	GRAND+ (S)	10^{-3}	10^{-5}	2	1024	10^{-6}	32	2	0.8
MAG-Scholar-C	GRAND+ (P)	10^{-2}	0	2	32	10^{-5}	32	10	1.0
	GRAND+ (A)	10^{-2}	0	2	32	10^{-5}	32	10	1.0
	GRAND+ (S)	10^{-2}	0	2	32	10^{-5}	32	2	1.0

The hyperparameter selection for GRAND+ consists of two stages: We first conduct search for basic hyperparameters of neural network. Specifically, we search learning rate lr from $\{10^{-2}, 10^{-3}, 10^{-4}\}$, weight decay rate wr from $\{0, 10^{-5}, 10^{-3}, 10^{-2}\}$, number of hidden layer L_m from {1,2} and dimension of hidden layer d_h from {32, 64, 128, 256, 512, 1024}.

In the second stage, we fix these basic hyperparameters as best configurations and search the following specific hyperparameters: DropNode rate δ, augmentation times per batch M, threshold r_{max} in GFPush, maximum neighborhood size k, propagation order N, confidence threshold γ, maximum consistency loss weight λ_{max}, size of unlabeled subset $|U'|$ and consistency loss function $\mathcal{D}$. To reduce searching cost, we keep some hyperparameters fixed. Specifically, we fix $\delta = 0.5$, $M = 2$ and $\gamma = \frac{2}{\#\text{classes}}$ across all datasets. We set $|U'| = |U|$ for Cora, Pubmed and Citeseer, and set $|U'| = 10000$ for other datasets. We also provide an analysis for the effect of $|U'|$ in Appendix A.3. We adopt KL divergence as the consistency loss function for AMiner-CS, Reddit and Amazon2M, and use L_2 distance for other datasets. This is because L_2 distance is easily to suffer from gradient vanishing problem when dealing with datasets with a large number of classes. We then conduct hyperparameter selection for r_{max}, k, N and λ_{max}. Specifically, we search r_{max} from $\{10^{-5}, 10^{-6}, 10^{-7}\}$, k from {16, 32, 64, 128}, N from {2, 4, 6, 8, 10, 20} and λ_{max} from {0.5, 0.8, 1.0, 1.2, 1.5}. The selected best hyperparameter configurations of GRAND+ are reported in Table 4.

A.2 Theorem Proofs

To prove Theorem 1, we first introduce the following lemma:

Lemma 1. *For any reserve vector $\mathbf{q}^{(n)}$, residue vector $\mathbf{r}^{(n)}$ and random walk transition vector $\mathbf{P}_s^n = (\widetilde{\mathbf{D}}^{-1}\widetilde{\mathbf{A}})_s^n$ ($0 \le n \le N$), we have:*

$$\mathbf{P}_s^n = \mathbf{q}^{(n)} + \sum_{i=1}^{n} (\mathbf{P}^i)^\top \cdot \mathbf{r}^{(n-i)} \tag{15}$$

Proof. We prove the Lemma by induction. For brevity, we use $\mathcal{RHS}^{(n)}$ to denote the right hand side of Equation 15. In Algorithm 1, $\mathbf{q}^{(n)}$ and $\mathbf{r}^{(n)}$ are initialized as $\vec{0}$ for $1 \le n \le N$, $\mathbf{r}^{(0)}$ and $\mathbf{q}^{(0)}$ are initialized as $\mathbf{e}^{(s)}$. Thus, Equation 15 holds at the algorithm beginning based on the following facts:

$$\mathcal{RHS}^{(0)} = \mathbf{e}^{(s)} = \mathbf{P}_s^0,$$
$$\mathcal{RHS}^{(n)} = (\mathbf{P}^n)^\top \cdot \mathbf{e}^{(s)} = \mathbf{P}_s^n, \quad 1 \le n \le N.$$

Then we assume Equation 15 holds at beginning of the $n'-$th iteration, we will show that the equation is still correct after a push operation on node v. We have three cases with different values of n:

1) When $n < n'$, the push operation does not change $\mathbf{q}^{(n)}$ and $\mathbf{r}^{(n-i)}, 1 \le i \le n$. Thus Equation 15 holds for $n < n'$.

2) When $n = n'$, the push operation decrements $\mathbf{r}^{(n-1)}$ by $\mathbf{r}_v^{(n-1)} \cdot \mathbf{e}^{(v)}$ and increments $\mathbf{q}^{(n)}$ by $\sum_{u \in \mathcal{N}_v} \mathbf{r}_v^{(n-1)} / \mathbf{d}_v \cdot \mathbf{e}^{(u)}$. Consequently, we have

$$\mathcal{RHS}^{(n)} = \mathbf{P}_s^n + \sum_{u \in \mathcal{N}_v} \mathbf{r}_v^{(n-1)} / \mathbf{d}_v \cdot \mathbf{e}^{(u)} - \mathbf{r}_v^{(n-1)} \cdot \mathbf{P}_v = \mathbf{P}_s^n + \vec{0} = \mathbf{P}_s^n.$$

Thus Equation 15 holds for $n = n' + 1$.

3) When $n > n'$, the push operation will decrease $\mathbf{r}^{(n')}$ by $\mathbf{r}_v^{(n')} \cdot \mathbf{e}^{(v)}$ and increase $\mathbf{r}^{(n'+1)}$ by $\sum_{u \in \mathcal{N}_v} \mathbf{r}_v^{(n')} / \mathbf{d}_v \cdot \mathbf{e}^{(u)}$. Thus we have

$$\begin{aligned}
\mathcal{RHS}^{(n)} &= \mathbf{P}_s^n + (\mathbf{P}^{n-n'-1})^\top \sum_{u \in \mathcal{N}_v} \mathbf{r}_v^{(n')} / \mathbf{d}_v \cdot \mathbf{e}^{(u)} - (\mathbf{P}^{n-n'})^\top (\mathbf{r}_v^{(n')} \cdot \mathbf{e}^{(v)}) \\
&= \mathbf{P}_s^n + (\mathbf{P}^{n-n'-1})^\top \cdot \left(\sum_{u \in \mathcal{N}_v} \mathbf{r}_v^{(n')} / \mathbf{d}_v \cdot \mathbf{e}^{(u)} - (\mathbf{r}_v^{(n')} \cdot \mathbf{P}_v)^\top \right) \\
&= \mathbf{P}_s^n + (\mathbf{P}^{n-n'-1})^\top \cdot \vec{0} = \mathbf{P}_s^n.
\end{aligned}$$

Thus Equation 15 holds for $n > n' + 1$.

Hence the induction holds, and the lemma is proved. □

Then, we could prove Theorem 1 as following.

Theorem 1. *Algorithm 1 has $O(N/r_{max})$ time complexity and $O(N/r_{max})$ memory complexity, and returns $\widetilde{\mathbf{\Pi}}_s$ as an approximation of $\mathbf{\Pi}_s$ with the L_1 error bound: $\| \mathbf{\Pi}_s - \widetilde{\mathbf{\Pi}}_s \|_1 \le N \cdot (2|E| + |V|) \cdot r_{max}$.*

Proof. Let $\mathcal{V}_n$ be the set of nodes to be pushed in step n. When the push operation is performed on $v \in \mathcal{V}_n$, the value of $\| \mathbf{r}^{(n-1)} \|_1$ will be decreased by at least $r_{max} \cdot \mathbf{d}_v$. Since $\| \mathbf{r}^{(n-1)} \|_1 \le 1$, we must have $\sum_{v \in \mathcal{V}_n} \mathbf{d}_v \cdot r_{max} \le 1$, thus:

$$\sum_{v \in \mathcal{V}_n} \mathbf{d}_v \le 1 / r_{max}. \tag{16}$$

Time Complexity. For the push operation on v in step n, we need to perform $\mathbf{d}_v$ times of updates for $\mathbf{r}^{(n)}$. So the total time of push operations in step n is bounded by $\sum_{v \in \mathcal{V}_n} \mathbf{d}_v$. Therefore, based on Equation 16, the time complexity of each step is bounded by $O(1/r_{max})$ and the total time complexity of Algorithm 1 has a $O(N/r_{max})$ bound.

Memory Complexity. When the n-th step iteration finishes, the count of non-zero elements of $\mathbf{r}^{(n)}$ is no more than $\sum_{v \in \mathcal{V}_n} \mathbf{d}_v$, as the push operation on v only performs $\mathbf{d}_v$ times of updates for $\mathbf{r}^{(n)}$. Thus the count of non-zero elements of $\mathbf{q}^{(n)}$ is also less than $\sum_{v \in \mathcal{V}_n} \mathbf{d}_v$. According to Equation 16, we can conclude that $\widetilde{\mathbf{\Pi}}_s$ has at most N/r_{max} non-zero elements. In implementation, all the vectors are stored as sparse structures. Thus Algorithm 1 has a memory complexity of $O(N/r_{max})$.

Error Bound. According to Lemma 1, we can conclude the following equations:

$$\begin{aligned}
\left\| \mathbf{P}_s^{(n)} - \mathbf{q}^{(n)} \right\|_1 &= \left\| \sum_{i=1}^{n} (\mathbf{P}^i)^\top \cdot \mathbf{r}^{(n-i)} \right\|_1 \\
&= \left\| \sum_{i=1}^{n} \mathbf{r}^{(n-i)} \right\|_1 \\
&\le \sum_{i=1}^{n} \left\| \mathbf{r}^{(n-i)} \right\|_1 .
\end{aligned} \tag{17}$$

After algorithm termination, we have $0 \le \mathbf{r}_v^{(n)} \le \mathbf{d}_v \cdot r_{max}$ for all $v \in V$. Hence,

$$\left\| \mathbf{r}^{(n)} \right\|_1 = \sum_{v \in V} \mathbf{r}_v^{(n)} \le \sum_{v \in V} \mathbf{d}_v \cdot r_{max} = (2|E| + |V|) \cdot r_{max}. \tag{18}$$

According to Equation 17, we can conclude that $\left\| \mathbf{P}_s^{(n)} - \mathbf{q}^{(n)} \right\|_1 \le n \cdot (2|E| + |V|) \cdot r_{max}$. Further more, we have:

$$\left\| \mathbf{\Pi}_s - \widetilde{\mathbf{\Pi}}_s \right\|_1 \le \sum_{n=0}^{N} w_n \left\| \mathbf{P}_s^{(n)} - \mathbf{q}^{(n)} \right\|_1 \le \sum_{n=0}^{N} w_n \cdot n \cdot (2|E| + |V|) \cdot r_{max} \tag{19}$$

As for $0 \le w_n$ and $\sum_{n=0}^{N} w_n = 1$, hence:

$$\sum_{n=0}^{N} w_n \cdot n \cdot (2|E| + |V|) \cdot r_{max} \le N \cdot (2|E| + |V|) \cdot r_{max}, \tag{20}$$

which indicates $\| \mathbf{\Pi}_s - \widetilde{\mathbf{\Pi}}_s \|_1 \le N \cdot (2|E| + |V|) \cdot r_{max}$. □

A.3 Additional Experiments

Table 5: Effects of unlabeled subset size ($|U'|$).

|U'|	Aminer			Reddit			Amazon2M		
	Acc (%)	RT (s)	AT (ms)	Acc (%)	RT (s)	AT (ms)	Acc (%)	RT (s)	AT (ms)
0	51.1 ± 1.4	10	149	92.3 ± 0.2	53	717	75.0 ± 0.7	63	2356
10^3	53.6 ± 1.6	9	153	92.6 ± 1.2	58	882	75.2 ± 0.5	62	2630
10^4	54.2 ± 1.6	10	250	92.8 ± 0.2	62	2407	76.1 ± 0.6	69	3649
10^5	54.4 ± 1.2	13	1121	92.9 ± 0.2	78	17670	76.3 ± 0.7	86	14250

Analysis for the size of U'. In GRAND+, a subset of unlabeled nodes U' are sampled from U for consistency regularization. To this end, we need to pre-compute the sparsified approximation $\widetilde{\mathbf{\Pi}}_v$ of row vector $\mathbf{\Pi}_v$ for each node $v \in U'$. Here we empirically analyze how the size of U' affects the classification accuracy (Acc), running time (RT) and approximation time (AT) of GRAND+. Table 5 presents the results of GRAND+ (S) when we vary $|U'|$ from 0 to 10^5 on AMiner-CS, Reddit and Amazon2M. We have the two observations: First, as $|U'|$ changes from 0 (meaning the consistency loss degenerates to 0) to 10^3, the classification performances are improved significantly with little changes on running time, which indicates the consistency regularization serves as an economic way for improving GRAND+'s generalization performance under semi-supervised setting. Second, when $|U'|$ exceeds 10^4, the increase rate of the accuracy will slow down, while the running time and approximation time increase more faster. This observation indicates the sampling procedure on unlabeled nodes is important for ensuring model's efficiency, which also enables us to explicitly control the trade-off between effectiveness and efficiency of GRAND+ through the sampling size $|U'|$.