Scaling Up Graph Neural Networks Via Graph Coarsening

Zengfeng Huang*
School of Data Science
Fudan University
huangzf@fudan.edu.cn

Shengzhong Zhang
School of Data Science
Fudan University
szzhang17@fudan.edu.cn

Chong Xi
School of Data Science
Fudan University
cxi19@fudan.edu.cn

Tang Liu
Fudan University
cnliutang@gmail.com

Min Zhou
Huawei Technologies Co. Ltd
zhoumin27@huawei.com

ABSTRACT

Scalability of graph neural networks remains one of the major challenges in graph machine learning. Since the representation of a node is computed by recursively aggregating and transforming representation vectors of its neighboring nodes from previous layers, the receptive fields grow exponentially, which makes standard stochastic optimization techniques ineffective. Various approaches have been proposed to alleviate this issue, e.g., sampling-based methods and techniques based on pre-computation of graph filters.

In this paper, we take a different approach and propose to use graph coarsening for scalable training of GNNs, which is generic, extremely simple and has sublinear memory and time costs during training. We present extensive theoretical analysis on the effect of using coarsening operations and provides useful guidance on the choice of coarsening methods. Interestingly, our theoretical analysis shows that coarsening can also be considered as a type of regularization and may improve the generalization. Finally, empirical results on real world datasets show that, simply applying off-the-shelf coarsening methods, we can reduce the number of nodes by up to a factor of ten without causing a noticeable downgrade in classification accuracy.

CCS CONCEPTS

• **Computing methodologies → Learning latent representations**; *Semi-supervised learning settings*; *Neural networks.*

KEYWORDS

Graph Coarsening; Graph Neural Networks; Scalable Training

ACM Reference Format:
Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. 2021. Scaling Up Graph Neural Networks Via Graph Coarsening. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3447548.3467256

1 INTRODUCTION

In the recent few years, graph neural network (GNN) has emerged as a major tool for graph machine learning [3, 7, 11, 17, 20, 21, 25, 30, 37], which has found numerous applications in scenarios with explicit or implicit graph structures, e.g., [13, 22, 31, 32, 38, 42, 44]. Despite the tremendous success, the difficulty of scaling up GNNs to large graphs remains one of the main challenges, which limits their usage in large-scale industrial applications. In traditional machine learning settings, the loss function of the model can be decomposed into the individual sample contributions, and hence stochastic optimization techniques working with mini-batches can be employed to tackle training set that is much larger than the GPU memory. However, GNN computes the representation of a node recursively from its neighbors, making the above strategy non-viable, as the loss corresponding to each sample in a ℓ-layer GNN depends on the subgraph induced by its ℓ-hop neighborhood, which grows exponentially with ℓ. Therefore, full-batch gradient descent is often used for training GNNs [20, 37], but this does not scale to large graphs due to limited GPU memory.

Recently, a large body of research work studies this issue and various techniques have been proposed to improve the scalability of GNNs. One prominent direction is to decouple the interdependence between nodes hence reducing the receptive fields. Pioneered by [17], layer-wise sampling combined with mini-batch training has proved to be a highly effective strategy, and since then, several follow-up works try to improve this baseline with optimized sampling process, better stochastic estimations, and other extensions [4, 5, 10, 33, 47]. Another related technique is based on subgraph sampling, which carefully samples a small subgraph in each training iteration and then simply performs full-batch gradient descent on this subgraph [8, 43]. In practice, performing random sampling from a large graph in each epoch requires many random accesses to the memory, which is not friendly to GPUs [33].

A second approach is largely motivated by [39], in which the authors show that removing the nonlinear activations in GCN [20] does not affect the accuracy by much on common benchmarks. The resulting model is simply a linear diffusion process followed by a classifier. Then the diffusion process can be pre-computed and stored, after which the classifier can be trained with naive stochastic optimization. Recently, this idea is extended to more general propagation rules akin to personalized Pagerank, and highly scalable algorithms for pre-computing the propagation process are investigated [2, 6]. Although such methods often perform better than sampling-based techniques on popular benchmarks [6], they only work for a restricted class of architectures: graph diffusion

[*] Corresponding author

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467256

and nonlinear feature transformation are decoupled, which does not retain the full expressive power of GNNs [40].

Our Contributions. In this paper, we investigate a simple and generic approach based on graph coarsening. In a nutshell, our method first applies an appropriate graph coarsening method, e.g., [27], which outputs a coarse graph with much smaller number of nodes and edges; then trains a GNN on this coarse graph; finally transfers the trained model parameters of this smaller model to the GNN defined on the original graph for making inference. Since, the training is only done on a much smaller graph, the training time and memory cost are *sublinear*, while all previous methods have time complexity at least linear in the number of nodes [6]. Moreover, full-batch gradient descent can be applied, which not only avoids doing random sampling on a large graph repeatedly, but is also much simpler than previous techniques, since any GNN model can be applied directly without changing the code. Our contributions are summarized as follows.

(1) A new method based on graph coarsening for scaling up GNN is proposed, which is generic, extremely simple and has sublinear training time and memory without using sampling.
(2) Extensive theoretical analysis is presented. We analyze the effect of coarsening operations on GNNs quantitatively and provides useful guidance on the choice of coarsening methods. Interestingly, our theoretical analysis shows that coarsening can also be considered as a type of regularization and may improve the generalization, which has been further verified by the empirical results.
(3) Empirical studies on real world datasets show that, simply applying off-the-shelf coarsening methods, we can reduce the number of nodes by up to a factor of ten without causing a noticeable downgrade in classification accuracy.

We remark that our methods and existing ones mentioned above are *complementary techniques*, and can be easily combined to tackle truly industrial-scale graphs.

2 PRELIMINARIES

2.1 Graph and Matrix Notations

In this paper, all graphs considered are undirected. A graph with node feature is denoted as $G = (V, E, X)$, where V is the vertex set, E is the edge set, and $X \in \mathbb{R}^{n \times f}$ is the feature matrix (i.e., the i-th row of X is the feature vector of node v_i). Let $n = |V|$ and $m = |E|$ be the number of vertices and edges respectively. We use $A \in \{0,1\}^{n \times n}$ to denote the adjacency matrix of G, i.e., the (i, j)-th entry in A is 1 if and only if their is an edge between v_i and v_j. The degree of a node v_i, denoted as d_i, is the number of edges incident on v_i. The degree matrix D is a diagonal matrix and the its i-th diagonal entry is d_i.

For a d-dimensional vector x, $\|x\|_2$ is the Euclidean norm of x. We use x_i to denote the ith entry of x, and $\text{diag}(x) \in \mathbb{R}^{d \times d}$ is a diagonal matrix such that the i-th diagonal entry is x_i. We use $A_{i:}$ and $A_{:i}$ to denote the i-th row and column of A respectively, and A_{ij} for the (i, j)-th entry of A. We use $\|A\|_2$ to denote the spectral norm of A, which is the largest singular value of A, and $\|A\|_F$ for the *Frobenius Norm*, which is $\sqrt{\sum_{i,j} a_{i,j}^2}$. The trace of a square matrix A is denoted by $\text{Tr}(\text{A})$, which is the sum of the diagonals in A. It is well-known that $\text{Tr}(A)$ is equal to the sum of its eigenvalues. For notational convenience, we always write $A_P \triangleq P^T AP$ for any matrix P with the same number of rows as A.

2.2 Graph Laplacian and Graph Fourier Transformation

The Laplacian matrix of a graph G is defined as $L_G = D - A$; when the underling graph G is clear from the context, we omit the subscription and simply write L. A key property of L is that its quadratic form measures the "smoothness" of a signal w.r.t. the graph structure, and thus is often used for regularization purposes. More formally, for any vector $x \in \mathbb{R}^n$, it is easy to verify that

$$x^T Lx = \sum_{i,j} A_{ij}(x_i - x_j)^2 = \sum_{(v_i,v_j)\in E} (x_i - x_j)^2. \tag{1}$$

Here, x can be viewed as a one-dimensional feature vector and $x^T Lx$ measures the smoothness of features across edges. This can be extended to multi-dimensional features. For any matrix $X \in \mathbb{R}^{n \times d}$, where X_i is the feature of the i-th node, then we have

$$\sum_{(v_i,v_j)\in E} \|X_{i:} - X_{j:}\|^2 = \sum_{i,j} A_{ij} \|X_{i:} - X_{j:}\|^2 = \text{Tr}(X^T LX). \tag{2}$$

In many applications, the symmetric normalized version of L, i.e., $D^{-1/2}LD^{-1/2}$, is the right matrix to consider, which is denoted as $\mathcal{N}$. Since $\mathcal{N}$ is real symmetric, it can be diagonalized and it is known that all its eigenvalues are in the range $[0, 2]$. Let $0 = \lambda_1 \leq \cdots \leq \lambda_n \leq 2$ be the eigenvalues of $\mathcal{N}$ with corresponding eigenvectors $u_1, \cdots, u_n$ and let $\mathcal{N} = U\Lambda U^T = \sum_{i=1}^n \lambda_i u_i u_i^T$ be the eigendecomposition. In graph signal processing, given an n-dimensional discrete signal $x \in \mathbb{R}^n$, its Graph Fourier Transformation (GFT) is $\hat{x} = U^T x$ [20]. The corresponding eigenvalue of a Fourier mode is the frequency. From this perspective, the orthogonal projector to the low-frequency eigenspace acts as a low-pass filter, which only retains contents in the lower frequencies; on the other hand, a projector to the high-frequency space is a high-pass filter.

2.3 Graph Neural Networks

In each layer of a GNN, the representation of a node is computed by recursively aggregating and transforming representation vectors of its neighboring nodes from the last layer. One special case is the Graph Convolutional Network (GCN) [20], which aims to generalize CNN to graph-structured data. Kipf and Welling [20] define graph convolution (GC) as $Z = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}XW$, where $\tilde{A} = A + I$, $\tilde{D} = D + I$, and W is a learnable parameter matrix. GCNs consist of multiple convolution layers of the above form, with each layer followed by a non-linear activation. In [21], the authors propose APPNP, which uses a propagation rules inspired from personalized Pagerank. More precisely, the APPNP model is defined as follows:

- $Z^{(1)} = H \triangleq f(X, W)$, where $f(X, W)$ is a neural network with parameter set W.
- $Z^{(k+1)} = (1 - \beta)\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}Z^{(k)} + \beta H$, where $\beta \in (0, 1]$ is a hyperparameter.

3 OUR METHOD

3.1 Graph Coarsening

Given a graph $G = (V, E, X)$, the coarse graph is a smaller weighted graph $G' = (V', E', X', W)$ with edge weights W. Denote $n' \triangleq |V'|$

and $m' \triangleq |E'|$. G' is obtained from the original graph by first computing a partition $P = \{C_1, C_2, \cdots, C_{n'}\}$ of V, i.e., the clusters $C_1 \cdots C_{n'}$ are disjoint and cover all the nodes in V. Each cluster C_i corresponds to a "super-node" in G' and the "super-edge" connecting the super-nodes C_i, C_j has weight equal to the total number of edges connecting nodes in C_i to C_j: $W_{ij} = \sum_{u \in C_i, v \in C_j} A_{ij}$.

The partition can be represented by a matrix $\hat{P} \in \{0, 1\}^{n \times k}$, with $\hat{P}_{ij} = 1$ if and only if vertex i belongs to cluster C_j. So, each row of P contains exactly one nonzero entry and columns of P are pairwise orthogonal. Then $W = A_{\hat{P}} \triangleq \hat{P}^T A \hat{P}$ and $A_{\hat{P}}$ is identified as the adjacency matrix of G'. Similarly, $D_{\hat{P}} \triangleq \hat{P}^T D \hat{P}$ is the degree matrix of G'. Note that the number of edges in the coarse graph is also significantly smaller than m, as each super-edge combines many edges in the original graph. It means that the number of non-zero entries in the adjacency matrix $A_{\hat{P}}$ is much smaller than A.

Let $c_j, j = 1, \cdots n'$ be the number of vertices in C_j, and $C \triangleq \mathrm{diag}(c_1, \cdots, c_k)$. The normalized version of $\hat{P}$ is $P \triangleq \hat{P} C^{-1/2}$, i.e., $P_{ij} = 1/\sqrt{c_j}$ if $v_i \in C_j$ and 0 otherwise. It is easy to verify that P has orthonormal columns, and thus $P^T P = I$. We use $\mathcal{P}$ to denote the set of all normalized partition matrices.

3.2 Our Method

The generic algorithm. We mainly focus on the semi-supervised node classification setting, where we are given an attributed graph $G = (V, E, X)$ and labels for a small subset of nodes. Assume the number of classes is l. We use $Y \in \{0, 1\}^{n \times l}$ to represent the label information: if v_i is labeled, then $Y_{i:}$ is the corresponding one-hot indicator vector, otherwise $Y_{i:} = 0$. We use $\mathrm{GNN}_G(W)$ to denote the GNN model based on G. Given a loss function ℓ, e.g., cross entropy, the loss of the model is denoted as $\ell(\mathrm{GNN}_G(W), Y)$. The training algorithm is to minimize the loss w.r.t. W. The time and memory costs of training are proportional to the size of G. To improve the computational costs, we first compute a coarse approximation of G, denoted as G', via graph coarsening described above, then minimize the loss $\ell(\mathrm{GNN}_{G'}(W), Y')$ w.r.t. W. The optimal parameter matrix W^* is then used in $\mathrm{GNN}_G()$ for prediction.

In the coarse graph, each node is a super-node corresponding to a cluster of nodes in the original graph. The feature vector of each super-node is the mean of the feature vectors of all nodes in the cluster, i.e., $X' = P^T X$. We set the label of each super-node similarly, i.e., $P^T Y$. However, it is possible that the super-node contains nodes from more than one class. For this case, we pick the dominating label, i.e., apply a row-wise argmax operation on $P^T Y$. In our experiments, we find that discarding such super-nodes with mixed labels often benefits the accuracy. However, in general, more sophisticated aggregation schemes can be applied to suit the application at hand. See Algorithm 1 for the description of our framework. We remark that graph coarsening can be efficiently pre-computed on CPUs, where the main memory size could be much larger than GPUs. The coarse graph G' is weighted and the number of nodes in each super-node may vary significantly. Thus, when constructing the smaller model $\mathrm{GNN}_{G'}(W)$, we sometimes need to revise the propagation scheme. Next, we give a slightly more general GC, which is motivated from our theoretical analysis in Section 4.

Algorithm 1 Training GNN with Graph Coarsening

```
Input: $G = (V, E, X)$; Labels $Y$; Model $\mathrm{GNN}_G(W)$; Loss $\ell$;
Output: Output trained weight matrix $W^*$
Apply a graph coarsening algorithm on $G$, and output a normalized partition matrix $P$.
Construct the coarse graph $G'$ using P;
Compute the feature matrix of $G'$ by $X' = P^T X$
Compute the labels of $G'$ by $Y' = \arg\max(P^T Y)$
Train parameter $W$ to minimize the loss $\ell(\mathrm{GNN}_{G'}(W), Y')$ to obtain a optimal weight matrix $W^*$
return $W^*$;
```

Graph convolution on the coarse graph. We define the convolution operation on G' as

$$Z = (D_{\hat{P}} + C)^{-1/2}(A_{\hat{P}} + C)(D_{\hat{P}} + C)^{-1/2} X' W.$$

Here we add C instead of I as in [20] to reflect the relative size of each super-node, for which we will give a theoretical justification in Section 4. Also, this definition includes the standard GC as a special case, i.e., when there is no coarsening, then $C = I$. By definitions of P and C, we have

$$\tilde{A}_P \triangleq P^T \tilde{A} P = P^T (A + I) P = A_P + I = C^{-1/2} A_{\hat{P}} C^{-1/2} + I,$$

$$\tilde{D}_P \triangleq P^T \tilde{D} P = P^T (D + I) P = D_P + I = C^{-1/2} D_{\hat{P}} C^{-1/2} + I.$$

Since $D_{\hat{P}}$ is diagonal, $\tilde{D}_P$ is further simplified to $C^{-1} D_{\hat{P}} + I = C^{-1}(D_{\hat{P}} + C)$. Then the coarse graph convolution is equivalent to

$$Z = \tilde{D}_P^{-1/2} \tilde{A}_P \tilde{D}_P^{-1/2} X' W, \tag{3}$$

which looks more similar to the standard GC.

4 THEORETICAL FOUNDATIONS

Note that, when $\beta \longrightarrow 0$, the propagation step of APPNP is the same as GCN. So one can think of GCN as a model which stacks multiple single-step APPNP models, interlaced by non-linear activations. In this section, we provide rigorous analysis on how APPNP behaves on the coarse graph, present theoretical guarantees on the approximation errors of different coarsening methods, and make interesting connections to existing graph coarsening schemes. We first provide a variational characterization of APPNP, from which we derive APPNP on the coarse graph.

4.1 A Characterization of APPNP

Let $Z^{(t)}$ be the output of the t-th layer in APPNP. It can be shown that Z^t converges to the solution to a linear system, see e.g., [21, 45].

Proposition 1. *$Z^{(\infty)}$ is the solution to the linear system*

$$\left(I - (1 - \beta)\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}\right) Z = \beta H \triangleq f(X, W). \tag{4}$$

It is known that the above linear system is non-singular (in fact positive definite) when β is strictly positive [9], and thus the solution exists and is unique. It is a standard fact in numerical optimization that the solution to such a linear system is the optima of some convex quadratic optimization problem.

Proposition 2. *Let Y^* be the optima of the following quadratic optimization problem:*

$$\min_{Y\in\mathbb{R}^{n\times h}} (1-\beta)\mathrm{Tr}\left(Y^TLY\right) + \beta\|\tilde{D}^{1/2}Y - H\|_F^2. \tag{5}$$

Then $Z^* = \tilde{D}^{1/2}Y^*$ is the unique solution to (4).

Let L' be the Laplacian of the coarse graph. APPNP on the coarse graph corresponds to an optimization problem of the same form except L is replaced by L'. With this perspective, we can quantitatively analyze the effect of replacing L with L' in APPNP. Of course the quadratic variational representation is not unique, and similar formulations have been used to derive label and feature propagation schemes [14, 45, 46].

In (5), the optimization problem is unconstrained. To motivate graph coarsening, we generalize it to the constrained case, where we require $Y \in C \subseteq \mathbb{R}^{n\times h}$ for some constraint set C, i.e.,

$$\min_{Y\in C} (1-\beta)\mathrm{Tr}\left(Y^TLY\right) + \beta\|\tilde{D}^{1/2}Y - H\|_F^2. \tag{6}$$

We will show that applying graph coarsening is roughly equivalent to putting a special constraint C on APPNP. Therefore, coarsening can also be considered as a type of regularization and may improve the generalization, which is verified by our empirical results.

Possible Choices of C. The canonical example of C is a set of matrices whose columns are within some k-dimensional subspace with $k < n$. More precisely, let $V \in \mathbb{R}^{n\times k}$ be an orthonormal basis of the k-dimensional subspace, then $C = \{Y : Y = VR, \text{for some } R \in \mathbb{R}^{k\times h}\}$. Different choices of C's give rise to different variants of APPNP, e.g., one could encode sparsity, rank, and general norm constraints in C, which may be highly useful depending on the tasks and datasets at hand. For the graph coarsening purpose, we will only focus on the case where C is a subspace. Nevertheless, being subspaces has already included many interesting special cases. For instance, when C is the eigenspace of the normalized Laplacian $\mathcal{N}$ corresponding to small eigenvalues, then it acts as a low-pass filter; on the other hand, when C consists of eigenvectors with high eigenvalues, then it is a high-pass filter.

4.2 Subspace Constraints and Dimensionality Reduction

In this subsection, we show that subspace constraints will benefit computation, as we essentially only need to solve a lower-dimensional problem. From now on, C is always a linear subspace of dimension k, and let $V \in \mathbb{R}^{n\times k}$ be an orthonormal basis of C. As a result, (6) can be rewritten as

$$\min_{Y:Y=VR \text{ for some } R\in\mathbb{R}^{k\times h}} (1-\beta)\mathrm{Tr}\left(Y^TLY\right) + \beta\|\tilde{D}^{1/2}Y - H\|_F^2. \tag{7}$$

Thus, we only need to solve a lower-dimensional problem

$$R^* = \arg\min_{R\in\mathbb{R}^{k\times h}} (1-\beta)\mathrm{Tr}\left(R^TV^TLVR\right) + \beta\|\tilde{D}^{1/2}VR - H\|_F^2. \tag{8}$$

The optima of (7) can be recover via $Y^* = VR^*$. Let $L_V = V^TLV$, which is an $k \times k$ matrix and thus much smaller than L, similarly let $A_V = V^TAV$, $\tilde{A}_V = V^T\tilde{A}V = A_V + I$, $D_V = V^TDV$ and $\tilde{D}_V = V^T\tilde{D}V = D_V + I$.

Theorem 4.1. *Let R^* be the optima of the quadratic optimization problem* (8). *Then $Z^* = \tilde{D}_V^{1/2}R^*$ is the unique solution to the linear system*

$$\left(I - (1-\beta)\tilde{D}_V^{-1/2}\tilde{A}_V\tilde{D}_V^{-1/2}\right) Z = \beta\tilde{D}_V^{-1/2}V^T\tilde{D}^{1/2}F.$$

Proof. By taking the gradient of (8) and set it to 0, we have

$$(1-\beta)(D_V - A_V)R^* + \beta(D_V + I)R^* - \beta V^T\tilde{D}^{1/2}F = 0.$$

By rearranging the terms, it implies

$$(D_V + I - (1-\beta)(A_V + I))\, R^* = \beta V^T\tilde{D}^{1/2}F$$
$$\Longrightarrow\ \tilde{D}_V^{1/2}(\tilde{D}_V^{1/2} - (1-\beta)\tilde{D}_V^{-1/2}\tilde{A}_V)R^* = \beta V^T\tilde{D}^{1/2}F.$$

Multiply both sides by $\tilde{D}_V^{-1/2}$ and reparameterize $Z^* = \tilde{D}_V^{1/2}R^*$,

$$\left(I - (1-\beta)\tilde{D}_V^{-1/2}\tilde{A}_V\tilde{D}_V^{-1/2}\right) Z^* = \beta\tilde{D}_V^{-1/2}V^T\tilde{D}^{1/2}F,$$

which proves the lemma. □

One should see the resemblance between the above equation and (4), and thus we may approximately solve Z^* using the same propagation rule.

Corollary 4.2. *Consider the propagation rule:*

- $Z^{(1)} = H' \triangleq \tilde{D}_V^{-1/2}V^T\tilde{D}^{1/2}H$,
- $Z^{(k+1)} = (1-\beta)\tilde{D}_V^{-1/2}\tilde{A}_V\tilde{D}_V^{-1/2}Z^{(k)} + \beta H'$.

Then, Z^t converges to Z^*.

This is almost the same as APPNP, but now the dimension, i.e., the size of the symmetric propagation matrix $\tilde{D}_V^{-1/2}\tilde{A}_V\tilde{D}_V^{-1/2}$ is k by k, which is smaller than that in the original APPNP.

Unfortunately, now the time to compute the propagation matrix, $\tilde{D}_V^{-1/2}\tilde{A}_V\tilde{D}_V^{-1/2}$, is $O(nk^2)$ which is expensive for moderately large k. Note the original propagation matrix $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ can be computed in time $O(m)$, and for sparse graph this is only $O(n)$. Moreover, $\tilde{D}_V^{-1/2}\tilde{A}_V\tilde{D}_V^{-1/2}$ is a dense matrix, which requires $O(k^2)$ space to store and in each propagation, the time complexity is $O(k^2h)$, where h is the size of feature vectors in $Z^{(t)}$. In comparison, for sparse graphs, $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ only requires $O(m)$ space and each propagation takes $O(mh)$ time. Therefore, unless the reduction ratio is extremely high, say reduce from 10^6 to 10^3, the computational costs and space usage could increase significantly, which defeats the purpose of graph coarsening in the first place.

4.3 Sparse Projections and Graph Coarsening

To overcome the above issues, we restrict the orthonormal matrix V to be sparse. In this paper, we will only consider the family of normalized partition matrices of size $n \times k$ (see Section 3.1 for the definitions), denoted as $\mathcal{P}$. Given a target constraint subspace C and its orthonormal basis V, we will first find a matrix $P \in \mathcal{P}$ that is close to V and then replace V with P in (8). Since P is also an orthonormal matrix, we can apply Corollary 4.2 directly. Therefore, for this surrogate quadratic objective, the propagation rule become

1) $Z^{(1)} = H := \tilde{D}_P^{-1/2}P^T\tilde{D}^{1/2}F$,

2) $Z^{(k+1)} = (1-\beta)\tilde{D}_P^{-1/2}\tilde{A}_P\tilde{D}_P^{-1/2}Z^{(k)} + \beta H$.

The above propagation converges to some R close to R^* (8), as long as $P \approx V$. How to find such a P will be discussed in the following subsections. Recall that $\tilde{A}_P = A_P + I = P^TAP + I$; and $\tilde{D}_P = P^TDP+I$ is still a diagonal matrix. Note when $P = I$, i.e., there is no coarsening, this recovers APPNP. Moreover, the propagation matrix $\tilde{D}_P^{-1/2}\tilde{A}_P\tilde{D}_P^{-1/2}$ is exactly the graph convolution we defined for coarse graph in Section 3.2. Now since P contains one non-zero entry per row, the time complexity to compute $\tilde{D}_P^{-1/2}\tilde{A}_P\tilde{D}_P^{-1/2}$ is $O(m)$, which is $O(n)$ for sparse graph. The number of non-zero entries in $\tilde{D}_P^{-1/2}\tilde{A}_P\tilde{D}_P^{-1/2}$ is m', i.e., the number of super-edges in G'. Then the space to store it is $O(m')$ and the time complexity to compute each propagation is $O(m'h)$. Thus, the time and space complexity in the forward pass are improved by a factor of $\frac{m}{m'}$ over the original graph, and note m' could be much smaller than m as each super-edge corresponds to many edges in the original graph. More importantly, the number of nodes is reduced from n to n'. So, the space and time complexity in backpropagation are improved by a factor of $\frac{n}{n'}$.

4.4 Nuclear Norm Error, k-Means, and Spectral Clustering

From the above discussion, the main question left is how to efficiently compute a partition matrix P which is a good approximation to the target orthonormal matrix V. In this subsection, we provide suitable metrics to quantify the approximation error and give efficient and effective approximation algorithms.

Our goal is to find a matrix $P \in \mathcal{P}$ whose column space is close to the space spanned by V. Since both P and V are orthonormal, so if P is close to V, then P^TV should be close to identity. Hence, one natural error metric is the distance between P^TV and I. Since P^TV is not symmetric in general, it is more convenient to measure the distance between V^TPP^TV and I, or $\|V^TPP^TV - I\|$ for some matrix norm $\|\cdot\|$. We next show that, when the matrix norm is chosen to be the nuclear norm (denoted as $\|\cdot\|_1$), i.e., the sum of singular values, the problem is equivalent to k-means clustering.

Theorem 4.3. *Let $S = \{v_1, \cdots, v_n\}$ be a set of n points, where v_i is the i-th row of V. Let* $\mathrm{Cost}(P)$ *be the k-means cost of the partition induced by P with respect to S. Then we have $\|I - V^TPP^TV\|_1 =$* $\mathrm{Cost}(P)$ *for all $P \in \mathcal{P}$.*

Proof. First observe that the matrix $I - V^TPP^TV$ is positive semidefinite, and therefore

$$\|I - V^TPP^TV\|_1 = \mathsf{Tr}\left(I - V^TPP^TV\right). \tag{9}$$

Moreover,

$$\begin{aligned}\mathsf{Tr}\left(I - V^TPP^TV\right) &= \mathsf{Tr}\left(V^TV - V^TPP^TV\right)\\ &= \mathsf{Tr}\left(V^TV - 2V^TPP^TV + V^TPP^TV\right)\\ &= \mathsf{Tr}\left(V^TV - 2V^TPP^TV + V^TPP^TPP^TV\right)\\ &= \mathsf{Tr}\left((PP^TV - V)^T(PP^TV - V)\right)\\ &= \|PP^TV - V\|_F^2,\end{aligned}$$

where in the last equality, we use the fact that $\|A\|_F^2 = \mathsf{Tr}(A^TA)$ for any A. Together with (9), we have

$$\|I - V^TPP^TV\|_1 = \|PP^TV - V\|_F^2. \tag{10}$$

The r.h.s. of (10) is exactly the k-means cost of the partition induced by P. To see this, let $C_1, \cdots, C_k$ be the clusters of points in this partition, i.e., $v_i \in C_j$ iff $P_{ij} \neq 0$, then the corresponding k-means cost of this partition is

$$\mathrm{Cost}(P) = \sum_{j=1}^{k}\sum_{v\in C_j}\|v - g_j\|_2^2, \tag{11}$$

where g_j is the centroid of the j-th cluster. Recall the definition of $\hat{P}$ (with $P = \hat{P}C^{-1/2}$), which is the unnormalized partition matrix. Then $g_j = \frac{1}{|C_j|}\sum_{v\in C_j} v = \frac{1}{c_j}\hat{P}_{:j}^TV$. Therefore,

$$\begin{aligned}\mathrm{Cost}(P) &= \sum_{j=1}^{k}\sum_{v\in C_j}\|v - \frac{1}{c_j}\hat{P}_{:j}^TV\|_2^2 = \sum_{i=1}^{n}\|v_i - \frac{1}{c_j}\hat{P}_{i:}\hat{P}^TV\|_2^2\\ &= \|PP^TV - V\|_F^2.\end{aligned}$$

By (10), we have $\mathrm{Cost}(P) = \|I - V^TPP^TV\|_1$ for all normalized partition matrix $P \in \mathcal{P}$, which proves the Lemma. □

We have the following simple corollary.

Corollary 4.4. $P^* = \arg\min_{P\in\mathcal{P}}\|I - V^TPP^TV\|_1$ *if and only if the partition induced by P^* has optimal k-means cost w.r.t. S.*

Connection to Spectral Clustering. From the above corollary, to obtain a good approximation P in terms of nuclear norm, it is equivalent to solve the k-means problem w.r.t. V. Note that when V consists of the k eigenvectors of the normalized Laplacian $\mathcal{N}$ with lowest eigenvalues, then applying k-means to V is the *spectral clustering* algorithm. Thus, in this paper, we provide an alternative explanation of the role of k-means in spectral clustering algorithms.

For sparse graphs, the time to compute the k lowest eigenvectors will be dominated by the complexity of k-means computation. In the worst case, the k-means problem is known to be NP-hard, and approximation algorithms are used in practice, e.g., Lloyd's algorithm [26], which takes $O(nkd)$ time per iteration, where d is the dimension of each points. For spectral clustering $d = k$. Therefore, spectral clustering does not scale well to large graphs for our application, since k, the number of clusters, will be quite large compared to typical graph clustering scenarios. We next investigate a relaxed error norm, and make a connection to a recent work of Loukas [27] on graph coarsening.

4.5 Spectral Norm Error

In the above subsection, we measure the error of P w.r.t. V by $\|I - V^TPP^TV\|_1$, which is the sum of singular values; next we relax this to the spectral norm $\|I - V^TPP^TV\|_2$, i.e., the maximum singular value. We have

$$\begin{aligned}\|I - V^TPP^TV\|_2 &= \max_{x:\|x\|_2=1}\left|x^T(V^TV - V^TPP^TV)x\right|\\ &= \max_{x:\|x\|_2=1}\left|x^TV^TVx - x^TV^TPP^TPP^TVx\right|\\ &= \max_{x:\|x\|_2=1}\left|\|Vx\|_2^2 - \|PP^TVx\|_2^2\right|\end{aligned}$$

Note that $y = Vx$ has norm 1 for any unit-norm x, and thus $\{y : y = Vx, \forall \text{x s.t. } \|x\|_2 = 1\}$ is the set of all unit vector in the subspace spanned by V, i.e., C. Thus we have

$$\begin{aligned}\|I - V^T PP^T V\|_2 &= \max_{y \in C, \|y\|=1} \left| \|y\|_2^2 - \|PP^T y\|_2^2 \right| \\ &= \max_{y \in C} \frac{\left| \|y\|_2^2 - \|PP^T y\|_2^2 \right|}{\|y\|_2^2} \\ &= \max_{y \in C} \frac{\|y - PP^T y\|_2^2}{\|y\|_2^2}, \qquad (12)\end{aligned}$$

where the last equality is from Pythagorean theorem (since PP^T is an orthogonal projection). This is essentially equivalent to the Grassmannian distance between two subspaces, which is defined as $\|PP^T - VV^T\|_2$. The equivalence proof is nontrivial and can be found in the book [19] (Theorem 6.34).

The above error measure is independent on the underlying graph. In many graph applications, it is often more suitable to use a generalized Euclidean norm $\| \cdot \|_L$, i.e., $\|x\|_L = \sqrt{x^T Lx}$, where L is the Laplacian of the graph. Using this generalized norm in (12), we will consider the following graph dependent error metric:

$$\max_{y \in C} \frac{\|y - PP^T y\|_L^2}{\|y\|_L^2}. \qquad (13)$$

It is still difficult to efficiently compute an partition matrix P that minimize the above objective. Fortunately, this objective has been studied in [27] recently (see Definition 11 in [27]), and the author proposed efficient approximation algorithms for the case when V is the first k eigenvectors. Moreover, several effective heuristics are discussed and tested empirically on real world datasets.

In our experiments, the coarsening algorithms from [27], which aim to minimize (13), perform better than spectral clustering. We believe this is mainly due to the generalized Euclidean norm used. Next, we provide a theoretical explanation on this.

Theorem 4.5. *Suppose* $\max_{y \in C} \frac{\|y - PP^T y\|_L}{\|y\|_L} \leq \varepsilon < 1$, *then we have for any* $y \in C$, *there exists* $x \in \mathbb{R}^k$ *such that*

$$\left| y^T Ly - x^T P^T LPx \right| \leq 3\varepsilon \|y\|_L^2.$$

Proof. Given y, we simply set $x = P^T y$. Then,

$$\begin{aligned}\left| \sqrt{y^T Ly} - \sqrt{x^T P^T LPx} \right| &= \left| \sqrt{y^T Ly} - \sqrt{y^T PP^T LPP^T y} \right| \\ &= \left| \|L^{1/2} y\|_2 - \|L^{1/2} PP^T y\|_2 \right| \\ &\leq \|L^{1/2}(y - PP^T y)\|_2 \quad \text{Triangle inequality} \\ &= \|y - PP^T y\|_L \\ &\leq \varepsilon \sqrt{y^T Ly} \quad \text{By assumption}\end{aligned}$$

Equivalently, $(1 - \varepsilon)\|y\|_L \leq \sqrt{x^T P^T LPx} \leq (1 + \varepsilon)\|y\|_L$, which implies $(1 - \varepsilon)^2 \|y\|_L^2 \leq x^T P^T LPx \leq (1 + \varepsilon)^2 \|y\|_L^2$. Since $(1 - \varepsilon)^2 = 1 - 2\varepsilon + \varepsilon^2 \geq 1 - 2\varepsilon$ and $(1 + \varepsilon)^2 = 1 + 2\varepsilon + \varepsilon^2 \leq 1 + 3\varepsilon$, the theorem follows from the above inequalities. □

Similarly, if we have $\max_{y \in \text{span}(P)} \frac{\|y - VV^T y\|_L}{\|y\|_L} \leq \varepsilon < 1$, we can also prove that, for any $x \in \mathbb{R}^k$, there exists $y \in C$ such that

$$\left| y^T Ly - x^T P^T LPx \right| \leq 3\varepsilon \|Px\|_L^2.$$

For graph coarsening, we essentially replace the graph regularization term $E(y) = y^T Ly, y \in C$ in (6) by $E'(x) = x^T P^T LP^L x, x \in \mathbb{R}^k$. So if the two conditions holds simultaneously, this replacement does not change the optimization problem by much, and the resulting embedding should be similar, which is qualitatively verified in the experiments.

5 RELATED WORK

To overcome the scalability issue of training GNNs. Layer-wise sampling combined with mini-batch training has been extensively studied [4, 5, 10, 17, 33, 47]. Subgraph sampling for scaling up GNNs, which sample a small subgraph in each training iteration and perform full-batch training on this subgraph, is also explored recently [8, 43]. The authors in [33] study the problem of how to reduce the sampling frequency in aforementioned sub-sampling approaches. Edge sampling is also used as effective tool for tackling oversmoothing [34]. Another approach focuses on how to simplify the models without sacrificing, in particular, to decouple the graph diffusion process from the feature transformation. In this way, the diffusion process can be pre-computed and stored, after which the classifier can be trained with naive stochastic optimization[2, 6, 39]. [35] propose a method to pre-compute and store graph convolutional filters of different size. Graph reduction techniques have been used to speed up combinatorial problems [15, 29]. Graph reduction with spectral approximation guarantees are studied in [18, 23, 27]. Recently, graph coarsening has been applied to speedup graph embedding algorithms [12, 16, 24]. As far as we are aware, this is the first work applying graph coarsening to speedup the training of GNNs in the semi-supervised setting.

6 EXPERIMENTS

In this section, we evaluate the performance of our method on two representative GNN architectures, namely GCN and APPNP: GCN has a structure with interlacing layers of graph diffusion and feature transformation, and APPNP decouples feature transformation from the diffusion. We compare the effect of different coarsening ratios on GCN and APPNP, including the full-graph training. We also test the effect of several representative graph coarsening methods.

6.1 Experimental Setup

Data splits. The results are evaluated on five real world networks Cora, Citeseer, Pubmed, Coauthor Physics and DBLP [1, 20, 36] for semi-supervised node classification. Refer to the appendix for more details of the five datasets. For Cora, Citeseer, and Pubmed, we use the public split from [41], which is widely used in the literature. In particular, the training set contains 20 labeled nodes per class, with an additional validation set of 500 and accuracy is evaluated on a test set of 1,000 nodes. For the other two datasets, the performance is tested on random splits [36], where 20 labeled nodes per class are selected for training, 30 per class for validation, and all the other nodes are used for testing. Moreover, we also test the performances

Table 1: Summary of results in terms of mean classification accuracy and standard deviation (in percent) over 20 runs on different datasets. The coarsening ratios of GCN and APPNP are $c = [0.7, 0.5, 0.3, 0.1]$ for each dataset respectively. The highest accuracy for each model in each column is highlighted in bold.

Method	Cora		Citeseer		Pubmed		Coauthor Physics		DBLP	
	5	Fixed	5	Fixed	5	Fixed	5	20	5	20
GCN	67.5±4.8	81.5±0.6	57.3±3.7	71.1±0.7	67.4±5.6	**79.0**±0.6	91.2±2.1	93.7±0.6	61.5±4.8	72.6±2.3
GCN (c=0.7)	67.9±4.3	82.3±0.6	57.5±5.9	71.8±0.4	68.3±5.2	78.9±0.4	91.0±1.9	**93.8**±0.6	61.4±5.0	72.1±2.1
GCN (c=0.5)	68.8±4.6	**82.7**±0.5	57.7±5.3	**72.0**±0.5	**68.9**±4.4	78.5±0.3	**91.5**±2.0	93.7±0.7	61.8±4.8	72.7±2.0
GCN (c=0.3)	**69.4**±4.5	81.7±0.5	58.1±5.2	71.4±0.3	68.7±4.2	78.4±0.4	90.8±2.3	93.4±0.6	64.8±5.2	74.5±1.9
GCN (c=0.1)	67.6±5.1	77.8±0.7	**58.3**±6.3	71.1±0.4	68.5±5.2	78.3±0.5	87.8±3.6	91.5±1.4	**67.9**±5.6	**76.0**±2.1
APPNP	72.8±3.8	83.3±0.5	59.4±4.5	71.8±0.5	70.4±4.9	80.1±0.2	92.0±1.6	**94.0**±0.6	**72.9**±4.2	79.0±1.1
APPNP (c=0.7)	**73.9**±4.6	**83.9**±0.8	59.7±4.3	71.8±0.6	70.7±5.5	**80.4**±0.3	**92.3**±1.6	93.7±0.8	72.0±4.5	78.7±1.3
APPNP (c=0.5)	73.4±4.3	83.7±0.7	60.4±4.8	**72.0**±0.5	**71.2**±5.0	79.6±0.3	91.8±1.9	93.9±0.5	72.3±4.0	79.1±1.2
APPNP (c=0.3)	73.1±3.5	82.5±0.6	**60.9**±5.7	71.6±0.4	70.6±5.3	78.4±0.7	91.7±1.5	93.6±0.6	72.7±4.2	**79.7**±1.0
APPNP (c=0.1)	70.8±4.9	80.2±0.8	60.7±5.8	71.8±0.5	70.4±4.9	77.3±0.5	88.6±3.3	91.0±1.2	72.1±5.8	79.0±1.7

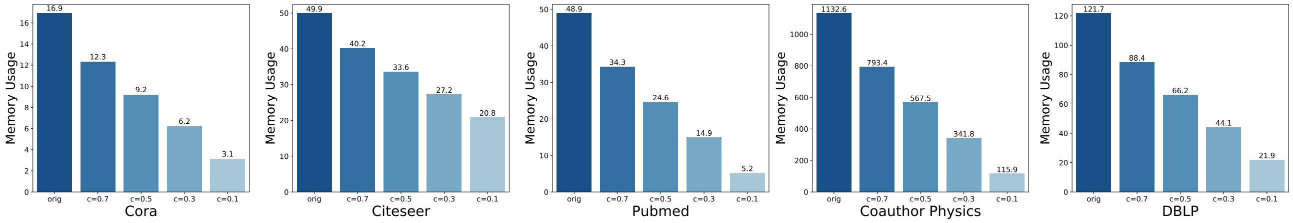


Figure 1: The Memory Usage of APPNP and coarse APPNP.

on each dataset under few label rates. We also evaluate in the few-shot regime, where, for each dataset, the training and validation set both have 5 labeled nodes per class, and the test set consists of all the rest. All the results are averaged over 20 runs and standard deviations are reported.

Implementation details. For the original GCN and APPNP, we follow the settings suggested in the previous papers [28, 36] for hyperparameters. In addition, we tuned the hyperparameter of models for better performance on Coauthor Physics and DBLP. For the fairness of comparison, our models use the same network architectures as baselines. For evaluating the effect of different coarsening ratios, we report the results of variation neighborhoods coarsening; see [27] for the detail. During the coarsening process, we remove super-nodes with mixed labels from the training set and the validation set, and also remove unlabeled isolated nodes. The detailed hyperparameter settings are listed in appendix.

6.2 Results and Analysis

Table 1 presents the node classification accuracy and standard deviation of different coarsening ratios. The memory usages are summarized in Figure 1.

Performance of GCN. Our results demonstrate that coarse GCN achieves good performance across five datasets under diffenernt experimental settings. In most cases, the coarsening operation will not reduce the accuracy by much. Interestingly, the best result for all settings (except for the public split on Pubmed) is not achieved on full-graph training. This verifies our hypothesis on the regularization effect of graph coarsening. It is also observed that, when the coarsening ratio is 0.3, the performance of GCN is competitive against full-graph training; actually, the performance is improved on 7 out of 10 settings. Even when the graph is reduced by 10 times, the performance is still comparable and in 6 out of 10 cases, the accuracy is higher than or the same as using full-graph training.

Performance of APPNP. For APPNP, we observe similar phenomenons as for GCN, even though the performance gain is not as noticeable as that on GCN. The resuts clearly are clearly consistent with our theoretical analysis.

Memory Usage. Figure 1 shows the memory usage of APPNP with different coarsening ratios; The memory usages of GCN are very similar to APPNP, and thus we omit the results on GCN. Compared with the size of the input tensor, the space occupied by the parameters is very small, so the proportion of the space occupied by the coarse APPNP is close to the coarsening rate.

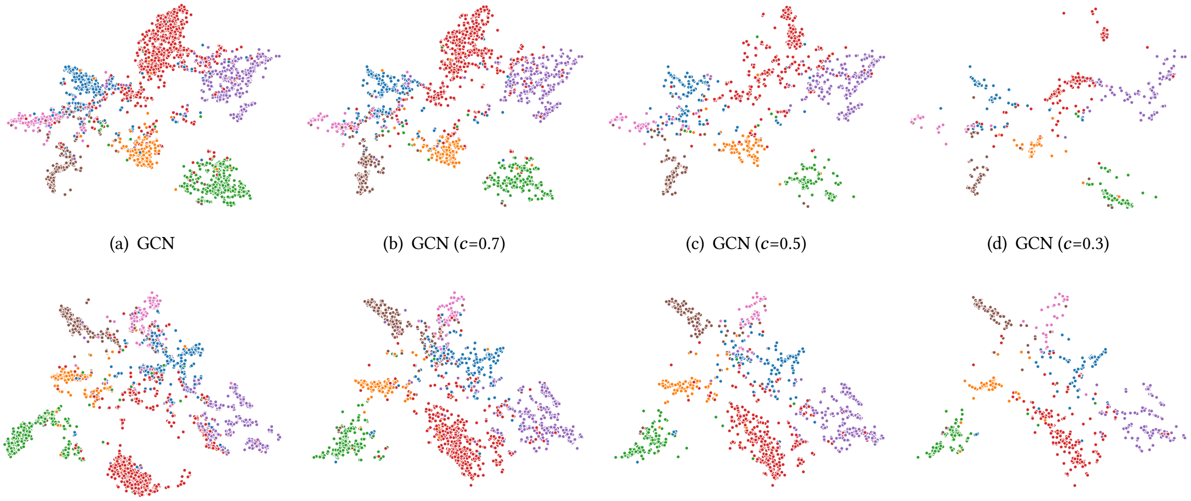
Visualization. We provide visualizations of the output layer with t-SNE for qualitative analyses. Here, we present the visualization results with different coarsening ratios on Cora in Figure 2, where nodes with the same color are from the same class. We clearly observe that, even though the number of nodes are different for each coarsening ratio, the overall distribution of node embeddings are quite similar across all ratios. This qualitatively verifies the theoretical analysis on the approximation quality of graph coarsening.

6.3 Studies on Different Coarsening Methods

Here we also study the efficacy of different coarsening methods for the proposed framework. We test the classification performance of four coarsening methods discussed in [27] together with spectral

Table 2: Summary of results in terms of accuracy, standard deviation and coarsening time(secs) with different coarsening methods.

Dataset	Coarsening Method	*c*=0.7			*c*=0.5			*c*=0.3		
		GCN	APPNP	Time	GCN	APPNP	Time	GCN	APPNP	Time
Cora	Spectral Clustering	82.2±0.5	83.2±0.4	23.4	81.5±0.7	82.5±0.5	16.3	79.4±0.5	78.0±1.3	10.0
	Variation Neighborhoods	**82.3**±0.6	**83.9**±0.8	2.0	**82.7**±0.5	83.7±0.7	1.3	81.7±0.5	**82.5**±0.6	2.1
	Variation Edges	**82.3**±0.5	83.6±0.6	0.3	82.2±0.5	**83.9**±0.5	0.5	80.0±0.4	81.1±0.7	0.6
	Algebraic JC	81.9±0.7	82.9±0.7	0.3	81.6±0.6	83.5±0.6	0.5	**82.2**±0.5	**82.5**±0.7	0.7
	Affinity GS	81.4±0.4	83.3±0.4	2.3	82.0±0.7	83.7±0.6	3.2	81.2±0.6	81.9±1.1	3.7
DBLP	Spectral Clustering	71.5±2.2	78.9±1.0	720.6	72.8±1.9	78.7±0.9	492.2	73.7±1.8	77.4±1.3	273.5
	Variation Neighborhoods	72.1±2.1	78.7±1.3	8.3	72.7±2.0	79.1±1.2	9.4	74.5±1.9	79.7±1.0	12.6
	Variation Edges	72.3±2.4	78.9±1.0	2.8	73.4±1.9	79.1±1.2	4.3	74.2±1.7	79.5±1.2	6.2
	Algebraic JC	72.5±2.3	78.6±1.6	3.0	73.1±2.0	78.3±1.1	5.5	74.0±1.7	79.1±1.2	7.3
	Affinity GS	**73.2**±2.1	**79.2**±1.6	135.7	**73.9**±1.7	**79.6**±0.7	199.6	**75.3**±1.6	**79.9**±1.1	225.9



(a) GCN

(b) GCN (*c*=0.7)

(c) GCN (*c*=0.5)

(d) GCN (*c*=0.3)

(e) APPNP

(f) APPNP (*c*=0.7)

(g) APPNP (*c*=0.5)

(h) APPNP (*c*=0.3)

Figure 2: Visualization of embeddings with t-SNE.

clustering on Cora and DBLP. The four coarsening methods from [27] are Variation Neighborhoods, Variation Edges, Algebraic JC and Affinity GS. In order to compare fairly, we use the same network structure and hyperparameters.

Table 2 shows the result of different coarsening methods. Except for spectral clustering, there is no obvious difference between other coarsening methods. Compared with other methods, Variation Neighborhoods has best overall testing accuracies, and the coarsening time of variation neighborhoods is also acceptable. Variation Edge and Algebraic JC are competitive in classification accuracies, and their computational time is faster than Variation Neighborhoods.The time of spectral clustering is high mainly because the number of clusters in the *k*-means steps is large,and we can observe that the time goes down as the coarsening ratio gets lower.

7 CONCLUSION

In this paper, we propose a different approach, which use graph coarsening, for scalable training of GNNs. Our method is generic, extremely simple and has sublinear training time and space. We present rigorous theoretical analysis on the effect of using coarsening operations and provides useful guidance on the choice of coarsening methods. Interestingly, our theoretical analysis shows that coarsening can also be considered as a type of regularization and may improve the generalization. Finally, empirical results on real world datasets show that, simply applying off-the-shelf coarsening methods, we can reduce the number of nodes by up to a factor of ten without causing a noticeable downgrade in classification accuracy. To sum up, this paper adds a new and simple technique in the toolbox for scaling up GNNs; from our theoretical analysis and empirical studies, it proves to be highly effective.

8 ACKNOWLEDGMENTS

This work is supported by Shanghai Sailing Program Grant No. 18YF1401200, National Natural Science Foundation of China Grant No. 61802069, Shanghai Science and Technology Commission Grant

No. 17JC1420200, and Science and Technology Commission of Shanghai Municipality Project Grant No. 19511120700.

REFERENCES

[1] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *International Conference on Learning Representations*.

[2] 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 *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2464–2473.

[3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*.

[4] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*.

[5] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *International Conference on Machine Learning*. 942–950.

[6] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable Graph Neural Networks via Bidirectional Propagation. In *Advances in Neural Information Processing Systems*.

[7] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*.

[8] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 257–266.

[9] Fan RK Chung and Fan Chung Graham. 1997. *Spectral graph theory*. Number 92. American Mathematical Soc.

[10] Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. 2020. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1393–1403.

[11] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3844–3852.

[12] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. 2019. GraphZoom: A Multi-level Spectral Approach for Accurate and Scalable Graph Embedding. In *International Conference on Learning Representations*.

[13] Elizabeth Dinella, Hanjun Dai, Ziyang Li, Mayur Naik, Le Song, and Ke Wang. 2020. Hoppity: Learning graph transformations to detect and fix bugs in programs. In *International Conference on Learning Representations*.

[14] Buchnik Eliav and Edith Cohen. 2018. Bootstrapped graph diffusions: Exposing the power of nonlinearity. In *Proceedings of the ACM on Measurement and Analysis of Computing Systems*.

[15] Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Racke, Inbal Talgam-Cohen, and Kunal Talwar. 2014. Vertex sparsifiers: New results from old techniques. *SIAM J. Comput.* (2014).

[16] Matthew Fahrbach, Gramoz Goranci, Richard Peng, Sushant Sachdeva, and Chi Wang. 2020. Faster graph embeddings via coarsening. In *International Conference on Machine Learning*.

[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[18] Yu Jin, Andreas Loukas, and Joseph JaJa. 2020. Graph coarsening with preserved spectral properties. In *International Conference on Artificial Intelligence and Statistics*.

[19] Tosio Kato. 1995. *Perturbation theory for linear operators*. Springer Science & Business Media.

[20] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

[21] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *International Conference on Learning Representations*.

[22] Chang Li and Dan Goldwasser. 2019. Encoding social information with graph convolutional networks forpolitical perspective detection in news media. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2594–2604.

[23] Huan Li and Aaron Schild. 2018. Spectral subspace sparsification. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science*.

[24] Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. 2018. Mile: A multi-level framework for scalable graph embedding. *arXiv preprint arXiv:1802.09612* (2018).

[25] Meng Liu, Hongyang Gao, and Shuiwang Ji. 2020. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[26] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[27] Andreas Loukas. 2019. Graph Reduction with Spectral and Cut Guarantees. *Journal of Machine Learning Research* 20, 116 (2019), 1–42.

[28] Jan E. Lenssen Matthias Fey. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *International Conference on Learning Representations Workshop*.

[29] Ankur Moitra. 2009. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*.

[30] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition*.

[31] Aditya Paliwal, Felix Gimeno, Vinod Nair, Yujia Li, Miles Lubin, Pushmeet Kohli, and Oriol Vinyals. 2020. Reinforced genetic algorithm learning for optimizing computation graphs. In *International Conference on Learning Representations*.

[32] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. 2021. Learning Mesh-Based Simulation with Graph Networks. In *International Conference on Learning Representations*.

[33] Morteza Ramezani, Weilin Cong, Mehrdad Mahdavi, Anand Sivasubramaniam, and Mahmut Kandemir. 2020. GCN meets GPU: Decoupling "When to Sample" from "How to Sample". In *Advances in Neural Information Processing Systems*.

[34] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*.

[35] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. 2020. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198* (2020).

[36] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *arXiv preprint arXiv:1811.05868* (2018).

[37] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations*.

[38] Jiayi Wei, Maruth Goyal, Greg Durrett, and Isil Dillig. 2020. Lambdanet: Probabilistic type inference using graph neural networks. In *International Conference on Learning Representations*.

[39] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning*. 6861–6871.

[40] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.

[41] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *International Conference on Machine Learning*. 40–48.

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

[43] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *International Conference on Learning Representations*.

[44] Muhan Zhang and Yixin Chen. 2019. Inductive Matrix Completion Based on Graph Neural Networks. In *International Conference on Learning Representations*.

[45] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. In *Advances in neural information processing systems*.

[46] Meiqi Zhu, Xiao Wang, Chuan Shi, Houye Ji, and Peng Cui. 2021. Interpreting and Unifying Graph Neural Networks with An Optimization Framework. In *Proceedings of The Web Conference 2021*.

[47] 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. In *Advances in neural information processing systems*.

A APPENDIX

Here we describe more details about the experiments to help in reproducibility.

Datasets. See Table 3 for a concise summary of the five datasets. The nodes in the networks are documents, each having a sparse bag-of-words feature vector; the edges represents citation links between documents.

Table 3: Summary of the datasets used in our experiments

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	5,429	1,433	7
Citeseer	3,327	4,732	3,703	6
Pubmed	19,717	44,338	500	3
Coauthor Physics	34,493	247,962	8,415	5
DBLP	17,716	52,867	1,639	4

Hyperparameters. For the coarse GCN, we use Adam optimizer with learning rates of $[0.01, 0.01, 0.01, 0.001, 0.01]$ and a L_2 regularization with weights $[0.0005, 0.0005, 0.0005, 0, 0.0005]$. The number of training epochs are $[60, 200, 200, 200, 50]$ and the early stopping is set to 10. For the coarse APPNP, α is set to $[0.1, 0.1, 0.1, 0.1, 0.05]$ and the number of layers is set to $[10, 10, 10, 20, 20]$ respectively. We use Adam optimizer with learning rates of $[0.01, 0.01, 0.01, 0.0005, 0.01]$ and a L_2 regularization with weights $[0.0005, 0.0005, 0.0005, 0, 0.0005]$. The number of training epochs are $[200, 200, 200, 500, 200]$ and the early stopping is set to $[10, 10, 10, 10, 0]$. The source code can be found in https://github.com/szzhang17/Scaling-Up-Graph-Neural-Networks-Via-Graph-Coarsening.

Configuration. All the models are implemented in Python and PyTorch Geometric. Experiments are conducted on an NVIDIA 2080 Ti GPU, Intel(R) Core(TM) i7-10750H CPU@2.60GHz and Intel(R) Xeon(R) Silver 4116 CPU@2.10GHz.