RawlsGCN: Towards Rawlsian Difference Principle on Graph Convolutional Network

Jian Kang[1], Yan Zhu[2], Yinglong Xia[2], Jiebo Luo[2,3], and Hanghang Tong[1]
[1]University of Illinois at Urbana-Champaign, {jiank2, htong}@illinois.edu;
[2]Facebook AI, {yzhu, yxia}@fb.com;
[3]University of Rochester, {jluo}@cs.rochester.edu;

ABSTRACT

Graph Convolutional Network (GCN) plays pivotal roles in many real-world applications. Despite the successes of GCN deployment, GCN often exhibits performance disparity with respect to node degrees, resulting in worse predictive accuracy for low-degree nodes. We formulate the problem of mitigating the degree-related performance disparity in GCN from the perspective of the Rawlsian difference principle, which is originated from the theory of distributive justice. Mathematically, we aim to balance the utility between low-degree nodes and high-degree nodes while minimizing the task-specific loss. Specifically, we reveal the root cause of this degree-related unfairness by analyzing the gradients of weight matrices in GCN. Guided by the gradients of weight matrices, we further propose a pre-processing method RawlsGCN-Graph and an in-processing method RawlsGCN-Grad that achieves fair predictive accuracy in low-degree nodes without modification on the GCN architecture or introduction of additional parameters. Extensive experiments on real-world graphs demonstrate the effectiveness of our proposed RawlsGCN methods in significantly reducing degree-related bias while retaining comparable overall performance.

CCS CONCEPTS

• **Information systems** → **Data mining**.

KEYWORDS

Graph neural networks, algorithmic fairness, distributive justice

ACM Reference Format:
Jian Kang[1], Yan Zhu[2], Yinglong Xia[2], Jiebo Luo[2,3], and Hanghang Tong[1]. 2022. RawlsGCN: Towards Rawlsian Difference Principle on Graph Convolutional Network. 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/10.1145/3485447.3512169

1 INTRODUCTION

Graph structured data naturally appears in many real-world scenarios, ranging from social network analysis [22], drug discovery [6], financial fraud detection [32] to traffic prediction [8], recommendation [30] and many more. The success of deep learning on grid-like data has inspired many graph neural networks in recent years. Among them, Graph Convolutional Network (GCN) [18] is one of the most fundamental and widely used ones, often achieving superior performance in a variety of tasks and applications.

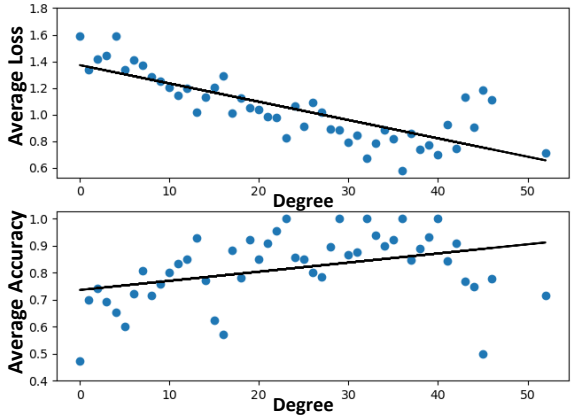


Figure 1: An illustrative example of the underrepresentation of low-degree nodes in semi-supervised node classification. A 2-layer GCN is trained on the Amazon-Photo dataset. Blue dots refers to the average loss and average accuracy of a specific degree group (i.e., the set of nodes with the same degree) in the top and bottom figures, respectively. Black lines are the regression lines of the blue dots in each figure. For visualization clarity, we only consider the degree groups that contain more than five nodes.

Despite their strong expressive power in node/graph representation learning, recent studies show that GCN tends to underrepresent nodes with low degrees [27], which could result in high loss values and low predictive accuracy in many tasks and applications. As shown in Figure 1, it is clear that low-degree nodes suffer from higher average loss and lower average accuracy in semi-supervised node classification. Such a performance disparity w.r.t. degrees is even more alarming, given that node degrees of real-world graphs often follow a long-tailed power-law distribution which means that a large fraction of nodes have low node degrees. In other words, the overall performance of GCN might be primarily beneficial to a few high-degree nodes (e.g., celebrities on a social media platform) but biased against a large number of low-degree nodes (e.g., grassroot users on the same social media platform).

To date, only a few efforts have been made to improve the performance for low-degree nodes. For example, DEMO-Net [31] randomly initializes degree-specific weight parameters in order to preserve the neighborhood structure for low-degree nodes. SL-DSGCN [27] introduces a degree-specific weight generator based on recurrent neural network (RNN) and a semi-supervised learning

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

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512169

module to provide pseudo labels for low-degree nodes. Recently, Tail-GNN [20] proposes a novel neighborhood translation mechanism to infer the missing local context information of low-degree nodes. However, the fundamental cause of degree-related unfairness in GCN largely remains unknown, which in turn prevents us from mitigating such unfairness from its root. Furthermore, existing works introduce either additional degree-specific weight parameters [27, 31] or additional operation on node representations (e.g., forged node generator, neighborhood translation) [20], which significantly increase the computational cost in both time and space and thus hinder the scalability of these models.

In order to tackle these limitations, we introduce the Rawlsian difference principle [25] to mitigate degree-related unfairness in GCN. As one of the earliest definitions of fairness from the theory of distributive justice [25], the Rawlsian difference principle aims to maximize the welfare of the least fortunate group and achieves stability when the worst-off group seeks to preserve its status quo. In the context of GCN, it requires a GCN model to have balanced performance among the groups of nodes with the same degree when the Rawlsian difference principle is in its stability. Given its essential role in training the GCN through backpropagation, we focus our analysis on the gradients of weight matrices of GCN. In particular, we establish the mathematical equivalence between the gradient of the weight matrix in a graph convolution layer and a weighted summation over the influence matrix of each node, weighted by its corresponding node degree in the input of GCN. This analysis not only reveals the root cause of degree-related unfairness in GCN, but also naturally leads to two new methods to mitigate the performance disparity with respect to node degrees, including (1) a pre-processing method named RawlsGCN-Graph that precomputes a doubly stochastic normalization of the input adjacency matrix to train the GCN; and (2) an in-processing method named RawlsGCN-Grad that normalizes the gradient so that each node will have an equal importance in computing the gradients of weight matrices.

The main contributions of this paper are summarized as follows.

- **Problem.** To our best knowledge, we are the first to introduce the Rawlsian difference principle to GCN so as to mitigate the degree-related unfairness.
- **Analysis.** We reveal the mathematical root cause of the degree-related unfairness in GCN.
- **Algorithms.** We propose two easy-to-implement methods to mitigate the degree bias, with no need to change the existing GCN architecture or introduce additional parameters.
- **Evaluations.** We perform extensive empirical evaluations on real-world graphs, which demonstrate that our proposed methods (1) achieve comparable accuracy with the vanilla GCN, (2) significantly decrease the degree bias, and (3) take almost the same training time as the vanilla GCN. Surprisingly, by mitigating the bias of low-degree nodes, our methods can sometimes improve the overall classification accuracy by a significant margin.

The rest of this paper is organized as follows. Section 2 formally defines the problem of enforcing the Rawlsian difference principle on GCN. Our mathematical analysis and the proposed methods are introduced in Section 3. Section 4 presents the experimental settings and evaluations. We review the related work in Section 5. Finally, Section 6 concludes the paper.

2 PROBLEM DEFINITION

In this section, we first introduce the main symbols used throughout the paper in Table 1. Then we present a brief review on the Graph Convonlutional Network (GCN) and the Rawlsian difference principle. Finally, we formally define the problem of enforcing the Rawlsian difference principle on GCN.

Table 1: Table of symbols.

Symbols	Definitions and Descriptions
$\mathbf{A}$	a matrix
$\mathbf{A}^T$	transpose of matrix $\mathbf{A}$
$\mathbf{u}$	a vector
$\mathcal{G}$	a graph
$\mathcal{V}$	a set of nodes
$J(\cdot)$	objective function
$\sigma(\cdot)$	activation function
$\mathbf{X}$	node feature matrix
$\mathbf{H}^{(l)}$	node representations at l-th layer
$\mathbf{W}^{(l)}$	weight matrix at l-th layer
L	number of graph convolution layers
d_l	hidden dimension of l-th layer
$deg(u)$	degree of node u

Unless otherwise specified, we denote matrices with bold upper-case letters (i.e., $\mathbf{A}$), vectors with bold lower-case letters (i.e., $\mathbf{x}$) and scalars with italic lower-case letters (i.e., c). We use rules similar to NumPy in Python for matrix and vector indexing. $\mathbf{A}[i, j]$ represents the entry of matrix $\mathbf{A}$ at the i-th row and the j-th column. $\mathbf{A}[i, :]$ and $\mathbf{A}[:, j]$ represent the i-th row and the j-th column of matrix $\mathbf{A}$, respectively. We use superscript ^{T} to represent the transpose of a matrix, i.e., $\mathbf{A}^T$ is the transpose of matrix $\mathbf{A}$.

2.1 Preliminaries

A – Graph Convolutional Network. We denote a graph as $\mathcal{G} = \{\mathcal{V}_\mathcal{G}, \mathbf{A}, \mathbf{X}\}$ where $\mathcal{V}_\mathcal{G}$ is the set of n nodes in the graph (i.e., $n = |\mathcal{V}_\mathcal{G}|$), $\mathbf{A}$ is the $n \times n$ adjacency matrix and $\mathbf{X} \in \mathbb{R}^{n \times d_0}$ is the node feature matrix.

GCN is a typical graph neural network model that contains a stack of graph convolution layers. Based on the first-order Chebyshev polynomial, the graph convolution layer learns the latent node representations through the message-passing mechanism in two major steps. First, each node in the graph aggregates its own representation with the representations of its one-hop neighbors. Then, the aggregated representations are transformed through a fully-connected layer. Mathematically, for the l-th graph convolution layer, denoting its output node representations as $\mathbf{H}^{(l)}$ (we assume $\mathbf{H}^{(0)} = \mathbf{X}$ for notation consistency), it computes the latent representation with $\mathbf{H}^{(l)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}), \forall l \in \{1, \ldots, L\}$ where $\sigma(\cdot)$ is the nonlinear activation function, $\mathbf{W}^{(l)} \in \mathbb{R}^{d_{l-1} \times d_l}$ is the weight matrix, and $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is the renormalized graph Laplacian with $\tilde{\mathbf{D}}$ being the diagonal degree matrix of $\mathbf{A} + \mathbf{I}$.

B – The Rawlsian Difference Principle. The Rawlsian difference principle is one of the major aspects of the equality principle

in the theory of distributive justice by John Rawls [25]. The difference principle achieves equality by maximizing the welfare of the worst-off groups. When the Rawlsian difference principle is in its stability, the performance of all groups are balanced since there is no worst-off group whose welfare should be maximized and all groups preserve their status quo. For a machine learning model that predicts task-specific labels for each data sample, the welfare is often defined as the predictive accuracy of the model [13, 24]. We denote (1) $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_h\}$ as a dataset that can be divided into h different groups, (2) $J(\mathcal{D}, \mathbf{Y}, \theta)$ as the task-specific loss function that the model with parameters θ aims to minimize where $\mathbf{Y}$ is the model output and (3) $U(\cdot, \theta)$ as the utility function that measures the predictive accuracy over a set of samples using the model with parameters θ. The Rawlsian difference principle can be mathematically formulated as

$$\begin{aligned} \min_{\theta} \quad & \text{Var}(\{U(\mathcal{D}_i, \theta) | i = 1, \ldots, h\}) \\ \text{s.t.} \quad & \theta = \text{argmin } J(\mathcal{D}, \mathbf{Y}, \theta) \end{aligned} \tag{1}$$

where $\text{Var}(\{U(\mathcal{D}_i, \theta) | i = 1, \ldots, h\})$ calculates the variance of the utilities of the groups $\{\mathcal{D}_1, \ldots, \mathcal{D}_h\}$.

2.2 Problem Definition

Despite superior performance of GCN in many tasks, GCN is often biased towards benefiting high-degree nodes. Following the overarching difference principle by John Rawls, we view the inconsistent predictive accuracy of GCN for high-degree and low-degree nodes as a distributive justice problem. However, directly enforcing the Rawlsian difference principle (Equation (1)) is nontrivial for two major reasons. First (C1), in many real-world applications, there could be multiple utility measures of interest. For example, in a classification task, an algorithm administrator might be interested in different measures like the classification accuracy, precision, recall and F1 score. Even if only one utility measure is considered, it is likely that the measure itself is non-differentiable, which conflicts with the end-to-end training paradigm of GCN. Second (C2), it is hard to decide whether a node is low-degree or high-degree by a clear threshold of degree value. A bad choice of the threshold value could even introduce more bias in calculating the average utilities. For example, if we set the threshold to be too large, the group of low-degree nodes might contain relatively high-degree nodes on which GCN achieves high utility. Then its average utility will increase by including these relatively high-degree nodes. In this case, even when the GCN balances the utilities between the groups of low-degree nodes and high-degree nodes, many nodes with relatively low degrees still suffer from the issue of low predictive accuracy.

To address the first challenge (C1), we replace the utility function U with the loss function J as a proxy measure of predictive accuracy. The intuition lies in the design of the end-to-end training paradigm of GCN, in which we minimize the loss function in order to maximize the predictive accuracy of GCN. Thus, we aim to achieve a balanced loss in the stability of the Rawlsian difference principle. As for the second challenge (C2), instead of setting a hard threshold to split the groups of low-degree nodes and high-degree nodes, we split the node set $\mathcal{V} = \cup_{i=1}^{deg_{\max}} \mathcal{V}_i$ to a maximum of $deg_{\max}$ degree groups where $\mathcal{V}_i$ refers to the set of nodes whose degrees are equal to i. With that, we formally define the problem of enforcing the Rawlsian difference principle on GCN as follows.

Problem 1. *Enforcing the Rawlsian Difference Princple on GCN*

Input: (1) an undirected graph $\mathcal{G} = \{\mathcal{V}_\mathcal{G}, \mathbf{A}, \mathbf{X}\}$; (2) an L-layer GCN with the set of weights θ; (3) a task-specific loss function $J(\mathcal{G}, \mathbf{Y}, \theta)$ where $\mathbf{Y}$ is the model output.

Output: a well-trained GCN that (1) minimizes the task specific loss $J(\mathcal{G}, \mathbf{Y}, \theta)$ given the input graph $\mathcal{G}$ and (2) achieves a balanced loss for all degree groups $\mathcal{V}_i$ ($i = 1, \ldots, deg_{\max}$).

3 METHODOLOGY

In this section, we propose a family of algorithms, namely RawlsGCN, to enforce the Rawlsian difference principle on GCN. We first present analysis on the source of degree-related unfairness, which turns out to be rooted in the gradient of weight parameters in the GCN. Then we discuss how to compute doubly stochastic matrix which is the key to mitigate the degree-related unfairness. Based on that, we present a pre-processing method (RawlsGCN-Graph) and an in-processing method (RawlsGCN-Grad) to solve Problem 1.

3.1 Source of Unfairness

The key to solve Problem 1 is to understand why the loss of a GCN varies among nodes with different degrees after training. Since the key component in training a GCN is the gradient matrix of the weight parameters with respect to the loss function, we seek to understand the root cause of such degree-related unfairness by analyzing it mathematically. In this section, our detailed analysis (Theorem 1) reveals the following fact: in a graph convolution layer, the gradient matrix $\frac{\partial J}{\partial \mathbf{W}}$ of the loss function J with respect to the weight parameter $\mathbf{W}$ is equivalent to a weighted summation of the influence matrix of each node,[1] weighted by its degree in input adjacency matrix $\hat{\mathbf{A}}$.

Theorem 1. *Suppose we have an input graph $\mathcal{G} = \{\mathcal{V}_\mathcal{G}, \mathbf{A}, \mathbf{X}\}$, the renormalized graph Laplacian $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$, a nonlinear activation function $\sigma()$ and an L-layer GCN that minimizes a task-specific loss function J. For any l-th hidden graph convolution ($\forall l \in \{1, \ldots, L\}$) layer, the gradient of the loss function J with respect to the weight parameter $\mathbf{W}^{(l)}$ is a linear combination of the influence of each node weighted by its degree in the renormalized graph Laplacian.*

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}} = \sum_{j=1}^{n} deg_{\hat{\mathbf{A}}}(j) \mathbf{I}_j^{(row)} = \sum_{i=1}^{n} deg_{\hat{\mathbf{A}}}(i) \mathbf{I}_i^{(col)} \tag{2}$$

where $deg_{\hat{\mathbf{A}}}(i)$ is the degree of node i in the renormalized graph Laplacian $\hat{\mathbf{A}}$, $\mathbf{I}_j^{(row)} = \left(\mathbf{H}^{(l-1)}[j, :]\right)^T \mathbb{E}_{i \sim p_{\hat{N}(j)}} \left[\frac{\partial J}{\partial \mathbf{E}^{(l)}[i,:]}\right]$ is the row-wise influence matrix of node j, $\mathbf{I}_i^{(col)} = \left(\mathbb{E}_{j \sim p_{\hat{N}(i)}} \left[\mathbf{H}^{(l-1)}[j, :]\right]\right)^T \frac{\partial J}{\partial \mathbf{E}^{(l)}[i,:]}$ is the column-wise influence matrix of node i, $\mathbf{H}^{(l-1)}$ is the input node embeddings of the hidden layer and $\mathbf{E}^{(l)} = \hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}$ is the node embeddings before the nonlinear activation.

Proof. To compute the derivative of the objective function J with respect to the weight $\mathbf{W}^{(l)}$ in the l-th graph convolution layer, by the graph convolution $\mathbf{H}^{(l)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})$ and the chain rule of matrix derivative, we have

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}[i, j]} = \sum_{a=1}^{n} \sum_{b=1}^{d_l} \frac{\partial J}{\partial \mathbf{H}^{(l)}[a, b]} \frac{\partial \mathbf{H}^{(l)}[a, b]}{\partial \mathbf{W}^{(l)}[i, j]} \tag{3}$$

[1] We use J to represent $J(\mathcal{G}, \mathbf{Y}, \theta)$ for notation simplicity.

where d_l is the number of columns in $\mathbf{H}^{(l)}$. To compute Equation (3), a key term to compute is $\frac{\partial \mathbf{H}^{(l)}[a,b]}{\partial \mathbf{W}^{(l)}[i,j]}$. Denoting σ' as the derivative of the activation function σ, by the graph convolution, we get

$$\begin{aligned}\frac{\partial \mathbf{H}^{(l)}[a,b]}{\partial \mathbf{W}^{(l)}[i,j]} &= \frac{\partial \sigma\big((\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})[a,b]\big)}{\partial (\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})[a,b]} \frac{\partial (\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})[a,b]}{\partial \mathbf{W}^{(l)}[i,j]} \\ &= \sigma'\big((\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})[a,b]\big)(\hat{\mathbf{A}}\mathbf{H}^{(l-1)})[a,i]\mathbb{1}[b==j]\end{aligned} \tag{4}$$

Combining Equations (3) and (4), we have

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{W}^{(l)}[i,j]} &= \sum_a \big(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\big)^T[i,a]\Big(\frac{\partial J}{\partial \mathbf{H}^{(l)}} \circ \sigma'(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})\Big)[a,j] \\ &= \big(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\big)^T[i,:]\Big(\frac{\partial J}{\partial \mathbf{H}^{(l)}} \circ \sigma'(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})\Big)[:,j]\end{aligned} \tag{5}$$

where $\circ$ represents the element-wise product. Writing Equation (5) into matrix form, we have

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}} = (\mathbf{H}^{(l-1)})^T\hat{\mathbf{A}}^T\Big(\frac{\partial J}{\partial \mathbf{H}^{(l)}} \circ \sigma'(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})\Big) \tag{6}$$

Let $\mathbf{E}^{(l)} = \hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}$ denote the node embeddings before the nonlinear activation, i.e., $\mathbf{H}^{(l)} = \sigma(\mathbf{E}^{(l)})$. By the chain rule of matrix derivative, we have $\frac{\partial J}{\partial \mathbf{E}^{(l)}} = \frac{\partial J}{\partial \mathbf{H}^{(l)}} \circ \sigma'(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})$. Then Equation (6) can be written as[2]

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}} = (\mathbf{H}^{(l-1)})^T\hat{\mathbf{A}}^T\frac{\partial J}{\partial \mathbf{E}^{(l)}} \tag{7}$$

To analyze the influence of each node on the gradient $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$, we factorize Equation (7) as follows.

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\hat{\mathbf{A}}^T[i,j](\mathbf{H}^{(l-1)}[i,:])^T\frac{\partial J}{\partial \mathbf{E}^{(l)}[j,:]} \tag{8}$$

Denoting the distribution $p_{\hat{\mathcal{N}}(i)}$ over the neighborhood of node i in the renormalized graph Laplacian $\hat{\mathbf{A}}$ such that $p_{\hat{\mathcal{N}}(i)}(j) \propto \hat{\mathbf{A}}[i,j] = \hat{\mathbf{A}}[j,i], \forall j \sim p_{\hat{\mathcal{N}}(i)}$, we can rewrite Equation (8) as

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{W}^{(l)}} &= \sum_{j=1}^{n} deg_{\hat{\mathbf{A}}}(j)\big(\mathbf{H}^{(l-1)}[j,:]\big)^T\mathbb{E}_{i\sim p_{\hat{\mathcal{N}}(j)}}\left[\frac{\partial J}{\partial \mathbf{E}^{(l)}[i,:]}\right] \\ &= \sum_{i=1}^{n} deg_{\hat{\mathbf{A}}}(i)\left(\mathbb{E}_{j\sim p_{\hat{\mathcal{N}}(i)}}\left[\mathbf{H}^{(l-1)}[j,:]\right]\right)^T\frac{\partial J}{\partial \mathbf{E}^{(l)}[i,:]}\end{aligned} \tag{9}$$

where $deg_{\hat{\mathbf{A}}}(i) = \sum_{i=1}^{n}\hat{\mathbf{A}}[i,j] = \sum_{i=1}^{n}\hat{\mathbf{A}}[j,i]$ is the degree of node i in the renormalized graph Laplacian $\hat{\mathbf{A}}$. We define the row-wise influence matrix $\mathbf{I}_j^{(\text{row})}$ of a node j and the column-wise influence matrix $\mathbf{I}_i^{(\text{col})}$ of a node i as follows.

$$\begin{aligned}\mathbf{I}_j^{(\text{row})} &= \big(\mathbf{H}[j,:]\big)^T\mathbb{E}_{i\sim p_{\hat{\mathcal{N}}(j)}}\left[\frac{\partial J}{\partial \mathbf{E}^{(l)}[i,:]}\right] \\ \mathbf{I}_i^{(\text{col})} &= \left(\mathbb{E}_{j\sim p_{\hat{\mathcal{N}}(i)}}\left[\mathbf{H}[j,:]\right]\right)^T\frac{\partial J}{\partial \mathbf{E}^{(l)}[i,:]}\end{aligned} \tag{10}$$

Then Equation (9) can be written as the weighted summation over the (row-/column-wise) influence matrix of each node, weighted by its corresponding degree in the renormalized graph Laplacian $\hat{\mathbf{A}}$.

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}} = \sum_{j=1}^{n} deg_{\hat{\mathbf{A}}}(j)\mathbf{I}_j^{(\text{row})} = \sum_{i=1}^{n} deg_{\hat{\mathbf{A}}}(i)\mathbf{I}_i^{(\text{col})} \tag{11}$$

□

[2] A simplified result on linear GCN without nonlinear activation is shown in [11], while our result (Equation (7)) generalizes to GCN with *arbitrary* differentiable nonlinear activation function.

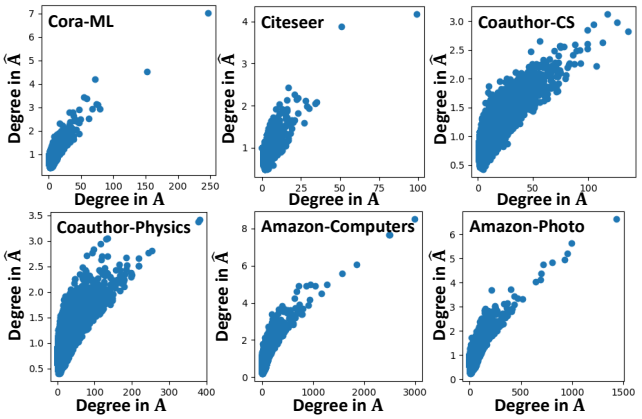


Figure 2: Node degrees in the original adjacency matrix A (x-axis) vs. the corresponding node degrees in the renormalized graph Laplacian Â (y-axis).

Remark: By Theorem 1, the gradient $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$ is essentially equivalent to a linear combination of the influence matrix of each node in the graph, with the corresponding node degree $deg_{\hat{\mathbf{A}}}$ in the renormalized graph Laplacian $\hat{\mathbf{A}}$ serving as the importance of the influence matrix. As a consequence, as long as the node degrees are not equal to a constant (e.g., 1), the gradient $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$ will favor the nodes with higher degrees in $\hat{\mathbf{A}}$. It is noteworthy that, even if $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is symmetrically normalized, such normalization only guarantees the largest eigenvalue of $\hat{\mathbf{A}}$ to be 1, whereas the degrees in $\hat{\mathbf{A}}$ are *not* constant as shown in Figure 2. From the figure, we have two key observations: (1) the node degrees in $\hat{\mathbf{A}}$ are not equal among all nodes (y-axis); and (2) there is a positive correlation between a node degree in $\mathbf{A}$ (x-axis) and a node degree in $\hat{\mathbf{A}}$ (y-axis). Putting everything together, it means that the higher the node degree in $\mathbf{A}$, the more importance it has on the gradient of the weight matrix. This shows exactly why the vanilla GCN favors high-degree nodes while being biased against low-degree nodes.

3.2 Doubly Stochastic Matrix Computation

In order to mitigate the node degree-related unfairness, the key idea is to normalize the importance of influence matrices (i.e., the node degrees in $\hat{\mathbf{A}}$) to 1 in Equation (11), such that each node will have equal importance in updating the weight parameters. To achieve that, Equation (11) naturally requires that the rows and columns of $\hat{\mathbf{A}}$ sum up to 1, which is a doubly stochastic matrix.

Computing the doubly stochastic matrix is nontrivial. To achieve that, we adopt the Sinkhorn-Knopp algorithm, which is an iterative algorithm to balance a matrix into doubly stochastic form [26]. Mathematically speaking, given a non-negative $n \times n$ square matrix $\mathbf{A}$, it aims to find two $n \times n$ diagonal matrices $\mathbf{D}_1$ and $\mathbf{D}_2$ such that $\mathbf{D}_1\mathbf{A}\mathbf{D}_2$ is doubly stochastic. The intuition of Sinkhorn-Knopp algorithm is to learn a sequence of matrices whose rows and columns are alternatively normalized. Mathematically, defining $\mathbf{r}_0 = \mathbf{c}_0 = \mathbf{1}$ to be the column vectors of all 1s and $\mathbf{x}^{-1}$ to be the operator for element-wise reciprocal, i.e., $\mathbf{x}^{-1}[i] = 1/\mathbf{x}[i]$, the Sinkhorn-Knopp algorithm alternatively calculates the following equations.

$$\mathbf{c}_{k+1} = (\mathbf{A}^T\mathbf{r}_k)^{-1} \quad \mathbf{r}_{k+1} = (\mathbf{A}\mathbf{c}_{k+1})^{-1} \tag{12}$$

If the algorithm converges after K iterations, the doubly stochastic form of $\mathbf{A}$ is $\mathbf{P} = \text{diag}(\mathbf{r}_K)\mathbf{A}\text{diag}(\mathbf{c}_K)$ where $\text{diag}(\mathbf{r}_K)$ and $\text{diag}(\mathbf{c}_K)$ diagonalize the column vectors $\mathbf{r}_K$ and $\mathbf{c}_K$ into diagonal matrices. The time and space complexities of computing the doubly stochastic matrix $\mathbf{P}$ is linear with respect to the size of input matrix $\mathbf{A}$.

Lemma 1. *(Time and space complexities) Let $\mathbf{A}$ be an $n \times n$ non-negative matrix with m nonzero elements, the time complexity of the Sinkhorn-Knopp algorithm is $O(K(m+n))$ where K is the number of iterations to convergence. It takes an additional $O(m+n)$ space.*

Proof. In the k-th iteration (where $1 \leq k \leq K$) of the Sinkhorn-Knopp algorithm, it takes $O(m)$ time to compute $\mathbf{A}^T\mathbf{r}_k$ and $\mathbf{A}\mathbf{c}_{k+1}$. Then it takes $O(n)$ time for element-wise reciprocal. Thus, the time complexity of an iteration is $O(m+n)$. Since the algorithm takes K iterations to converge, the overall time complexity is $O(K(m+n))$. Regarding the space complexity, it takes an additional $O(n)$ space to store vectors $\mathbf{c}_k$ and $\mathbf{r}_k$ in the k-th iteration and an additional $O(m)$ time to store the resulting doubly stochastic form of matrix $\mathbf{A}$. Thus, the overall space complexity is $O(m+n)$. □

Next, we prove the convergence of the Sinkhorn-Knopp algorithm in our setting. To this end, we first present the definition of the diagonal of a matrix corresponding to a column permutation.

Definition 1. *(Diagonal of a matrix corresponding to a column permutation [26]) Let $\mathbf{A}$ be an $n \times n$ square matrix and δ be a permutation over the set $\{1, \ldots, n\}$.*

(1)The sequence of elements $\mathbf{A}[1, \delta(1)], \mathbf{A}[2, \delta(2)], \ldots, \mathbf{A}[n, \delta(n)]$ is called the diagonal of $\mathbf{A}$ corresponding to δ.

(2)If δ is the identity, the diagonal is the main diagonal of $\mathbf{A}$.

(3)If $\mathbf{A}[i, \delta(i)] > 0, \forall i$, the diagonal is a positive diagonal.

We then provide the formal definition of the support of a matrix in Definition 2.

Definition 2. *(Support of a non-negative square matrix [26]) Let $\mathbf{A}$ be an $n \times n$ non-negative matrix, $\mathbf{A}$ is said to have support if $\mathbf{A}$ contains a positive diagonal. $\mathbf{A}$ is said to have total support if $\mathbf{A} \neq \mathbf{0}$ and if every positive element of $\mathbf{A}$ lies on a positive diagonal, where $\mathbf{0}$ is the zero matrix of the same size as $\mathbf{A}$.*

By Definitions 1 and 2, Sinkhorn and Knopp [26] proved the following result.

Theorem 2. *(Sinkhorn-Knopp theorem [26]) If $\mathbf{A}$ is an $n \times n$ non-negative matrix, the Sinkhorn-Knopp algorithm converges and finds the unique doubly stochastic matrix of the form $\mathbf{D}_1\mathbf{A}\mathbf{D}_2$ if and only if $\mathbf{A}$ has total support, where $\mathbf{D}_1$ and $\mathbf{D}_2$ are diagonal matrices.*

Proof. Omitted. □

Based on Theorem 2, we give Lemma 2 which says that the Sinkhorn-Knopp algorithm always converges in our setting and finds the doubly stochastic matrix with respect to the renormalized graph Laplacian $\hat{\mathbf{A}}$.

Lemma 2. *Given an adjacency matrix $\mathbf{A}$, if $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$ with $\tilde{\mathbf{D}}$ as the degree matrix of $\mathbf{A}+\mathbf{I}$, the Sinkhorn-Knopp algorithm always converges to find the unique doubly stochastic form of $\hat{\mathbf{A}}$.*

Proof. The key idea is to prove that $\hat{\mathbf{A}}$ has total support. Let $deg_{\hat{\mathbf{A}}}(i)$ be the degree of node i in $\hat{\mathbf{A}}$. It is trivial that $\hat{\mathbf{A}}$ has support because its main diagonal is positive. In order to prove that $\hat{\mathbf{A}}$ has total support, for any undirected edge $\mathbf{A}[i, j]$, we define a column permutation δ_{ij} as a permutation that satisfies (1) $\delta_{ij}(i) = j$; (2) $\delta_{ij}(j) = i$ and (3) $\delta_{ij}(k) = k, \forall k \neq i$ and $k \neq j$. Then, for any edge (i, j), by applying the permutation δ_{ij}, the diagonal of $\hat{\mathbf{A}}$ corresponding to δ_{ij} is a positive diagonal due to the non-negativity of $\hat{\mathbf{A}}$. Thus, $\hat{\mathbf{A}}$ has total support because all positive elements of $\hat{\mathbf{A}}$ lie on positive diagonals, which completes the proof. □

Lemma 2 guarantees that we can always calculate the doubly stochastic form of $\hat{\mathbf{A}}$ using the Sinkhorn-Knopp algorithm, in order to ensure the equal importance of node influence in calculating the gradient of the weight parameter $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$ in the l-th layer.

3.3 RawlsGCN Algorithms

If the gradient $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$ is computed using the doubly stochastic matrix $\hat{\mathbf{A}}_{DS}$ with respect to the renormalized graph Laplacian $\hat{\mathbf{A}}$, it is fair with respect to node degrees because all nodes will have equal importance in determining the gradient, i.e., their degrees in $\hat{\mathbf{A}}_{DS}$ are all equal to 1. Thus, a fair gradient with respect to node degrees can be calculated using Equation (7) as follows.

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}}_{\text{fair}} = (\mathbf{H}^{(l-1)})^T \hat{\mathbf{A}}_{DS}^T \frac{\partial J}{\partial \mathbf{E}^{(l)}} \tag{13}$$

where $\mathbf{E}^{(l)} = \hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}$.

Observing the computation of fair gradient in Equation (13), it naturally leads to two methods to mitigate degree-related unfairness, including (1) a pre-processing method named RawlsGCN-Graph which utilizes the doubly stochastic matrix $\hat{\mathbf{A}}_{DS}$ as the input adjacency matrix, and (2) an in-processing method named RawlsGCN-Grad that normalizes the gradient in GCN with Equation (13).

Method #1: Pre-processing with RawlsGCN-Graph. If we are allowed to modify the input of the GCN whereas the model itself are fixed, we can precompute the input renormalized graph Laplacian $\hat{\mathbf{A}}$ into its doubly stochastic form $\hat{\mathbf{A}}_{DS}$ and feed $\hat{\mathbf{A}}_{DS}$ as the input of the GCN. With that, the gradient computed using Equation 7 is equivalent to Equation (13). As a consequence, the Rawlsian difference principle is naturally ensured since all nodes in $\hat{\mathbf{A}}_{DS}$ have the same degree. Given a graph $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{X}\}$ and an L-layer GCN, RawlsGCN-Graph adopts the following 3-step strategy.

1. Precompute $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$ where $\tilde{\mathbf{D}}$ is the diagonal degree matrix of $\mathbf{A}+\mathbf{I}$.
2. Precompute $\hat{\mathbf{A}}_{DS}$ by applying the Sinkhorn-Knopp algorithm on $\hat{\mathbf{A}}$.
3. Input $\hat{\mathbf{A}}_{DS}$ and $\mathbf{X}$ to the GCN for model training.

Method #2: In-processing with RawlsGCN-Grad. If we have access to the model or the model parameters while the input data is fixed, we can precompute the doubly stochastic matrix $\hat{\mathbf{A}}_{DS}$ and use $\hat{\mathbf{A}}_{DS}$ to compute the fair gradient by Equation (13). Then training GCN with the fair gradient ensures the Rawlsian difference principle because nodes of different degrees share the same importance in determining the gradient for gradient descent-based optimization. Given a graph $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{X}\}$ and an L-layer GCN, the general workflow of RawlsGCN-Grad is as follows.

1. Precompute $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$ where $\tilde{\mathbf{D}}$ is the diagonal degree matrix of $\mathbf{A}+\mathbf{I}$.
2. Precompute $\hat{\mathbf{A}}_{DS}$ by applying the Sinkhorn-Knopp algorithm on $\hat{\mathbf{A}}$.

3. Input $\hat{\mathbf{A}}$ and $\mathbf{X}$ to the GCN for model training.
4. For each graph convolution layer $l \in \{1, \ldots, L\}$, compute the fair gradient $\frac{\partial J}{\partial \mathbf{W}^{(l)}}_{\text{fair}}$ for each weight matrix $\mathbf{W}^{(l)}$ using Equation (13).
5. Update the model parameters $\mathbf{W}^{(l)}$ using the fair gradient $\frac{\partial J}{\partial \mathbf{W}^{(l)}}_{\text{fair}}$.
6. Repeat steps 4-5 until the model converges.

An advantage of both RawlsGCN-Graph and RawlsGCN-Grad is that no additional time complexity will be incurred during the learning process of GCN, since we can precompute and store $\hat{\mathbf{A}}_{DS}$. For RawlsGCN-Graph, $\hat{\mathbf{A}}_{DS}$ has the same number of nonzero elements as $\hat{\mathbf{A}}$ because computing $\hat{\mathbf{A}}_{DS}$ is essentially rescaling each nonzero element in $\hat{\mathbf{A}}$. Thus, there is no additional time cost during the graph convolution operation with $\hat{\mathbf{A}}_{DS}$. For RawlsGCN-Grad, computing the fair gradient with Equation (13) enjoys the same time complexity as the vanilla gradient computation (Equation (7)) because $\hat{\mathbf{A}}_{DS}$ and $\hat{\mathbf{A}}$ has exactly the same number of nonzero entries. Thus, there is no additional time cost in big-O notation in optimizing the GCN parameters. In terms of the additional costs in computing $\hat{\mathbf{A}}_{DS}$, it bears a linear time and space complexities with respect to the number of nodes and the number of edges in the graph as stated in Lemma 1.

4 EXPERIMENTAL EVALUATION

In this section, we evaluate our proposed RawlsGCN methods in the task of semi-supervised node classification to answer the following questions:

- **Q1.** How accurate are the RawlsGCN methods in node classification?
- **Q2.** How effective are the RawlsGCN methods in debiasing?
- **Q3.** How efficient are the RawlsGCN methods in time and space?

4.1 Experimental Settings

Datasets. We utilize six publicly available real-world networks for evaluation. Their statistics, including the number of nodes, the number of edges, number of node features, number of classes and the median of node degrees (Median Deg.), are summarized in Table 2. For semi-supervised node classification, we use a fixed random seed to generate the training/validation/test sets for each network. The training set contains 20 nodes per class. The validation set and the test set contain 500 nodes and 1000 nodes, respectively.

Table 2: Statistics of datasets.

Name	Nodes	Edges	Features	Classes	Median Deg.
Cora-ML	2,995	16,316	2,879	7	3
Citeseer	3,327	9,104	3,703	6	2
Coauthor-CS	18,333	163,788	6,805	15	6
Coauthor-Physics	34,493	495,924	8,415	5	10
Amazon-Computers	13,752	491,722	767	10	22
Amazon-Photo	7,650	238,162	745	8	22

Baseline Methods. We compare the proposed RawlsGCN methods with several baseline methods, including GCN [18], DEMO-Net [31], DSGCN [27], Tail-GNN [20], Adversarial Fair GCN (AdvFair) [3] and REDRESS [9]. Detailed description of each baseline method is provided in Appendix.[3]

Metrics. We use cross entropy as the loss function in semi-supervised node classification. To answer **Q1**, we evaluate the accuracy of node classification (i.e., Acc. in Table 3). For metrics in **Q2**, we define the bias w.r.t. the Rawlsian difference principle as the variance of degree-specific average cross entropy (i.e., AvgCE) to be consistent with the definition of Problem 1. Mathematically, it is defined as

$$\begin{aligned} \text{AvgCE}(k) &= \mathbb{E}[\{\text{CE}(u), \forall \text{ node } u \text{ such that } deg(u) = k\}] \\ Bias &= \text{Var}(\{\text{AvgCE}(k), \forall \text{ node degree } k\}) \end{aligned} \tag{14}$$

where $\text{CE}(u)$ and $deg(u)$ are the cross entropy and the degree of node u, respectively. To measure the efficiency (**Q3**), we count the number of learnable parameters (# Param. in Table 4), GPU memory usage in MB (Memory in Table 4) and training time in seconds.

Parameter Settings. For all methods, we use a 2-layer GCN as the backbone model with the hidden dimension as 64. We evaluate all methods on 5 different runs and report their average performance. We train RawlsGCN models for 100 epochs without early stopping. For the purpose of the reproducibility, the random seeds for these 5 runs are varied from 0 to 4. We use the Adam optimizer to train all methods. Unless otherwise specified, the default weight decay of the optimizer is set to 0.0005. The detailed parameter settings (including learning rate, training epochs of baseline methods) are included in Appendix.

Machine Configuration and Reproducibility. All datasets are publicly available. All codes are programmed in Python 3.8.5 and PyTorch 1.9.0. All experiments are performed on a Linux server with 96 Intel Xeon Gold 6240R CPUs at 2.40 GHz and 4 Nvidia Tesla V100 SXM2 GPUs with 32 GB memory. We will release the source code of the proposed methods upon the publication of the paper.

4.2 Main Results

Effectiveness results. The evaluation results are shown in Table 3. We do not report the results of Tail-GNN for the Coauthor-Physics dataset due to the out-of-memory (OOM) error. From the table, we can see that our proposed RawlsGCN-Graph and RawlsGCN-Grad are the only two methods that can consistently reduce the bias across all datasets. Though REDRESS reduces bias more than the proposed methods on the Coauthor-CS dataset, it bears a much higher bias than our proposed methods in other datasets. Surprisingly, on the Amazon-Computers and Amazon-Photo datasets, our proposed RawlsGCN-Graph and RawlsGCN-Grad significantly improve the overall classification accuracy by mitigating the bias of low-degree nodes. This is because, compared with the vanilla GCN, the classification accuracy of low-degree nodes are increased while the classification accuracy of high-degree nodes are largely retained, resulting in the significantly improved overall accuracy.

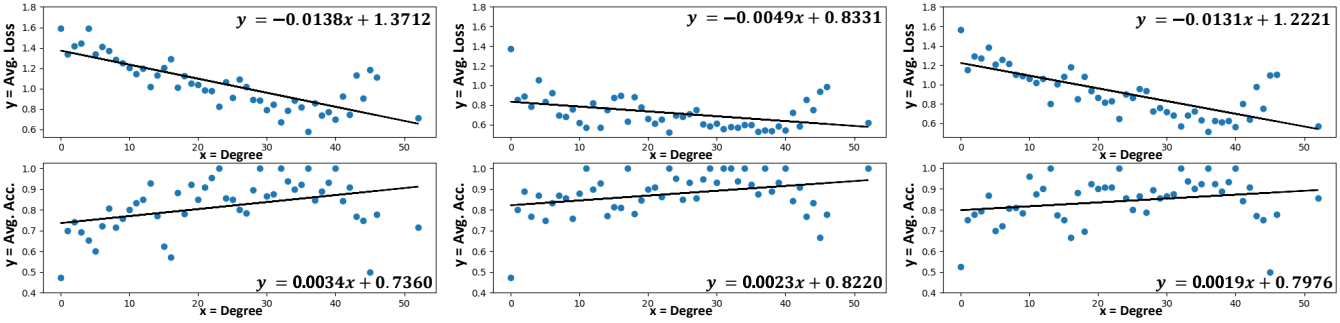
In Figure 3, we visualize how RawlsGCN-Graph and RawlsGCN-Grad benefit the low-degree nodes and balances the performance between low-degree nodes and high-degree nodes. From the figure, we observe that both RawlsGCN-Graph (Figure 3b) and RawlsGCN-Grad (Figure 3c) show lower average loss values and higher average classification accuracy compared with GCN (Figure 3a). Moreover, to visualize how our proposed methods balance the performance between low-degree nodes and high-degree nodes, we perform linear regression on the average loss and average accuracy

[3]We use the official PyTorch implementation of GCN, Tail-GNN and REDRESS for experimental evaluation. For DEMO-Net, we implement our own PyTorch version and consult with the original authors for a sanity check. For DSGCN, we implement our own PyTorch version due to the lack of publicly available implementation.

Table 3: Effectiveness for node classification. Lower is better for bias (in gray). Higher is better for accuracy (Acc., in percentage).

Method	Cora-ML		Citeseer		Coauthor-CS	
	Acc.	Bias	Acc.	Bias	Acc.	Bias
GCN	80.10 ± 0.812	0.392 ± 0.046	68.60 ± 0.341	0.353 ± 0.040	93.28 ± 0.194	0.075 ± 0.004
DEMO-Net	61.60 ± 0.687	0.181 ± 0.015	60.26 ± 0.408	0.315 ± 0.022	65.90 ± 0.583	0.164 ± 0.006
DSGCN	30.26 ± 5.690	8.003 ± 2.766	31.42 ± 3.257	6.887 ± 1.947	44.20 ± 7.155	1.460 ± 0.397
Tail-GNN	78.54 ± 0.582	0.503 ± 0.284	66.34 ± 0.009	0.655 ± 0.382	92.66 ± 0.196	0.052 ± 0.031
AdvFair	67.56 ± 2.594	10.01 ± 2.480	50.26 ± 6.277	3.146 ± 2.425	84.82 ± 2.254	12.26 ± 6.797
REDRESS	75.70 ± 0.620	0.955 ± 0.213	65.80 ± 0.518	0.944 ± 0.077	92.44 ± 0.233	0.028 ± 0.003
RawlsGCN-Graph (Ours)	76.96 ± 1.098	0.105 ± 0.012	69.34 ± 0.745	0.196 ± 0.013	92.52 ± 0.264	0.043 ± 0.002
RawlsGCN-Grad (Ours)	79.34 ± 1.247	0.232 ± 0.065	68.81 ± 0.462	0.283 ± 0.047	92.68 ± 0.240	0.058 ± 0.007

Method	Coauthor-Physics		Amazon-Computers		Amazon-Photo	
	Acc.	Bias	Acc.	Bias	Acc.	Bias
GCN	93.96 ± 0.367	0.023 ± 0.001	64.84 ± 0.641	0.353 ± 0.026	79.58 ± 1.507	0.646 ± 0.038
DEMO-Net	77.50 ± 0.566	0.084 ± 0.010	26.48 ± 3.455	0.456 ± 0.021	39.92 ± 1.242	0.243 ± 0.013
DSGCN	79.08 ± 1.533	0.262 ± 0.075	27.68 ± 1.663	1.407 ± 0.685	26.76 ± 3.387	0.921 ± 0.805
Tail-GNN	OOM	OOM	76.24 ± 1.491	1.547 ± 0.670	86.00 ± 2.715	0.471 ± 0.264
AdvFair	87.44 ± 1.132	0.892 ± 0.502	53.50 ± 5.362	4.395 ± 1.102	75.80 ± 3.563	51.24 ± 39.94
REDRESS	94.48 ± 0.172	0.019 ± 0.001	80.36 ± 0.206	0.455 ± 0.032	89.00 ± 0.369	0.186 ± 0.030
RawlsGCN-Graph (Ours)	94.06 ± 0.196	0.016 ± 0.000	80.16 ± 0.859	0.121 ± 0.010	88.58 ± 1.116	0.071 ± 0.006
RawlsGCN-Grad (Ours)	94.18 ± 0.306	0.021 ± 0.002	74.18 ± 2.530	0.195 ± 0.029	83.70 ± 0.672	0.186 ± 0.068



(a) GCN (b) RawlsGCN-Graph (c) RawlsGCN-Grad

Figure 3: Visualization on how our proposed RawlsGCN algorithms improve the performance of low-degree nodes on the Amazon-Photo dataset. (a) shows the results for vanilla GCN. (b) shows the results for RawlsGCN-Graph. (c) shows the results for RawlsGCN-Grad. Similar to Figure 1, blue dots refer to the average loss (Avg. Loss) and average accuracy (Avg. Acc.) of a specific degree group in the top and bottom figures, respectively. Black lines are the regression lines of the blue dots in each figure. For a more clear visualization, we only consider the degree groups which contain more than five nodes.

with respect to node degree. From the figure, we can see that the slope of the regression lines for RawlsGCN-Graph and RawlsGCN-Grad are flatter than the slope of the regression line in GCN.

Efficiency results. We measure the memory and time consumption of all methods in Table 4. In the table, GCN (100 epochs) and GCN (200 epochs) denote the GCN models trained with 100 epochs and 200 epochs, respectively. From the table, we have two key observations: (1) Compared with other baseline methods, RawlsGCN-Graph and RawlsGCN-Grad have fewer number of parameters to learn and are much more efficient in memory consumption; (2) RawlsGCN-Graph and RawlsGCN-Grad bear almost the same training time as the vanilla GCN with 100 epochs of training (i.e., GCN (100 epochs)) while all other baseline methods significantly increase the training time.

Table 4: Efficiency of training a 2-layer GCN on the Amazon-Photo dataset. Lower is better for all columns. GPU memory usage (Memory) is measured in MB. Training time is measured in seconds.

Method	# Param.	Memory	Training Time
GCN (100 epochs)	48, 264	1, 461	13.335
GCN (200 epochs)	48, 264	1, 461	28.727
DEMO-Net	11, 999, 880	1, 661	9158.5
DSGCN	181, 096	2, 431	2714.8
Tail-GNN	2, 845, 567	2, 081	94.058
AdvFair	89, 280	1, 519	148.11
REDRESS	48, 264	1, 481	291.69
RawlsGCN-Graph (Ours)	48, 264	1, 461	11.783
RawlsGCN-Grad (Ours)	48, 264	1, 461	12.924

Table 5: Ablation study of different matrix normalization techniques on the Amazon-Photo. Lower is better for bias (in gray). Higher is better for accuracy (Acc., in percentage).

Method	Normalization	Acc.	Bias
RawlsGCN-Graph	**Row**	87.98 ± 0.791	0.076 ± 0.006
	Column	88.32 ± 2.315	0.138 ± 0.112
	Symmetric	89.12 ± 0.945	0.071 ± 0.005
	Doubly Stochastic	88.58 ± 1.116	0.071 ± 0.006
RawlsGCN-Grad	**Row**	82.86 ± 1.139	0.852 ± 0.557
	Column	84.96 ± 1.235	0.221 ± 0.064
	Symmetric	82.92 ± 1.121	0.744 ± 0.153
	Doubly Stochastic	83.70 ± 0.672	0.186 ± 0.068

4.3 Ablation Study

To evaluate the effectiveness of the doubly stochastic normalization on the renormalized graph Laplacian, we compare it with three other normalization methods, including row normalization, column normalization and symmetric normalization. As shown in Table 5, while all these normalization methods lead to similar accuracy (within 2% difference), doubly stochastic normalization leads to a much smaller bias than others.

5 RELATED WORK

Graph neural network is an emerging research topic due to its strong empirical performance in many tasks like classification [14], regression [15] and recommendation [30]. Bruna et al. [4] propose the Graph Convolutional Neural Networks (GCNNs) by simulating the convolution operation in the spectrum of graph Laplacian. Kipf et al. [18] propose the Graph Convolutional Networks (GCNs) that aggregates the neighborhood information inspired by the localized first-order approximation of spectral graph convolution. Hamilton et al. [12] propose GraphSAGE that learns node representation in the inductive setting. Atwood et al. [2] propose a diffusion-based graph convolution operation that aggregates the neighborhood information through the graph diffusion process. Veličković et al. [29] introduce the multi-head self-attention mechanism into graph neural networks. Regarding graph neural networks for degree-aware representation learning, Wu et al. [31] propose two methods (i.e., hashing function and degree-specific weight function) to learn degree-specific representations for node and graph classification. Liu et al. [20] learns robust embeddings for low-degree nodes (i.e., tail nodes) by introducing a novel neighborhood translation operation to predict missing information for the tail nodes and utilizing a discriminator to differentiate the head node embeddings and tail node embeddings. Different from [20, 31], our work directly improves the performance on low-degree nodes without relying on any additional degree-specific weights or degree-aware operations.

Fair graph mining has attracted much research attention recently. In terms of group fairness, it has been incorporated into several graph mining tasks, including ranking, clustering, node embedding and graph neural networks. Tsioutsiouliklis et al. [28] propose two fairness-aware PageRank algorithms (i.e., fairness-sensitive PageRank and locally fair PageRank) to ensure a certain proportion of total PageRank mass is assigned to nodes in a specific demographic group. Kleindessner et al. [19] propose fair spectral clustering which aims to ensure each demographic group is represented with the same fraction as in the whole dataset in all clusters. Bose et al. [3] study the compositional fairness for graph embedding, which aims to debias the node embeddings with respect to a combination of sensitive attributes. Rahman et al. [23] modify the random walk procedure in node2vec [10] so that neighbors in the minority demographic group enjoy a higher probability of being reached. Buyl et al. [5] debias the embeddings by injecting bias information in the prior of conditional network embedding [16]. Dai et al. [7] promote statistical parity and equal opportunity in graph neural networks through adversarial learning. Individual fairness is another fundamental fairness notion. Kang et al. [17] present the first effort on individually fair graph mining through Laplacian regularization on the pairwise node similarity matrix. Dong et al. [9] incorporate ranking-based individual fairness into graph neural networks inspired by learning-to-rank. However, neither group fairness nor individual fairness is suitable to solve our problem. For group fairness, the low-degree nodes are often the majority group due to the long-tailed degree distribution. For individual fairness, it only considers fairness in node level by considering the similarity between two nodes, whereas our problem considers fairness in group level in which the groups are defined by node degrees. Many other fairness notions are also studied in graph mining. Agarwal et al. [1] exploit the connection between counterfactual fairness and stability to learn fair and robust node embeddings. Tang et al. [27] propose a RNN-based degree-specific weight generator with self-supervised learning to mitigate the degree-related bias. Our work differs from [27] because we do not change the GCN architecture and do not introduce any degree-specific weights. Rahmattalabi et al. [24] incorporate Rawlsian difference principle to the graph covering problem. Different from [24] that deals with a combinatorial problem, we are the first to introduce such fairness notion in graph neural networks without compromising its differentiable end-to-end paradigm.

6 CONCLUSION

In this paper, we introduce the Rawlsian difference principle to Graph Convolutional Network (GCN), where we aim to mitigate degree-related unfairness. We formally define the problem of enforcing the Rawlsian difference principle on GCN as balancing the loss among groups of nodes with the same degree. Based on that, we reveal the mathematical root cause of degree-related unfairness by studying the computation of the gradient of weight parameters in GCN. Guided by its computation, we propose a pre-processing method named RawlsGCN-Graph and an in-processing method named RawlsGCN-Grad to mitigate the degree-related unfairness. Both methods rely on the Sinkhorn-Knopp algorithm in computing the doubly stochastic matrix of the graph Laplacian, which is guaranteed to converge in the context of GCN. Extensive evaluation on six real-world datasets demonstrate the effectiveness and efficiency of our proposed methods. In the future, we will investigate how to generalize our methods to other graph neural networks.

ACKNOWLEDGEMENT

J. Kang and H. Tong are partially supported by NSF (1947135 and 1939725), DARPA (HR001121C0165) and Army Research Office (W911NF2110088). The content of the information in this document does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

[1] Chirag Agarwal, Himabindu Lakkaraju, and Marinka Zitnik. 2021. Towards a Unified Framework for Fair and Stable Graph Representation Learning. *arXiv preprint arXiv:2102.13186* (2021).

[2] James Atwood and Don Towsley. 2016. Diffusion-Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*. 1993–2001.

[3] Avishek Bose and William Hamilton. 2019. Compositional Fairness Constraints for Graph Embeddings. In *International Conference on Machine Learning*. 715–724.

[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral Networks and Locally Connected Networks on Graphs. *arXiv preprint arXiv:1312.6203* (2013).

[5] Maarten Buyl and Tijl De Bie. 2020. DeBayes: A Bayesian Method for Debiasing Network Embeddings. In *International Conference on Machine Learning*. 1220–1229.

[6] Chi Chen, Weike Ye, Yunxing Zuo, Chen Zheng, and Shyue Ping Ong. 2019. Graph Networks as a Universal Machine Learning Framework for Molecules and Crystals. *Chemistry of Materials* 31, 9 (2019), 3564–3572.

[7] Enyan Dai and Suhang Wang. 2021. Say No to the Discrimination: Learning Fair Graph Neural Networks with Limited Sensitive Attribute Information. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 680–688.

[8] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. 2021. ETA Prediction with Graph Neural Networks in Google Maps. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*.

[9] Yushun Dong, Jian Kang, Hanghang Tong, and Jundong Li. 2021. Individual Fairness for Graph Neural Networks: A Ranking based Approach. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 300–310.

[10] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 855–864.

[11] Kai Guo, Kaixiong Zhou, Xia Hu, Yu Li, Yi Chang, and Xin Wang. 2021. Orthogonal Graph Neural Networks. *arXiv preprint arXiv:2109.11338* (2021).

[12] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*. 1025–1035.

[13] Tatsunori Hashimoto, Megha Srivastava, Hongseok Namkoong, and Percy Liang. 2018. Fairness without Demographics in Repeated Loss Minimization. In *International Conference on Machine Learning*. 1929–1938.

[14] Baoyu Jing, Chanyoung Park, and Hanghang Tong. 2021. Hdmi: High-Order Deep Multiplex Infomax. In *Proceedings of the Web Conference 2021*. 2414–2424.

[15] Baoyu Jing, Hanghang Tong, and Yada Zhu. 2021. Network of Tensor Time Series. In *Proceedings of the Web Conference 2021*. 2425–2437.

[16] Bo Kang, Jefrey Lijffijt, and Tijl De Bie. 2018. Conditional Network Embeddings. In *International Conference on Learning Representations*.

[17] Jian Kang, Jingrui He, Ross Maciejewski, and Hanghang Tong. 2020. InFoRM: Individual Fairness on Graph Mining. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 379–389.

[18] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

[19] Matthäus Kleindessner, Samira Samadi, Pranjal Awasthi, and Jamie Morgenstern. 2019. Guarantees for Spectral Clustering with Fairness Constraints. In *International Conference on Machine Learning*. 3458–3467.

[20] Zemin Liu, Trung-Kien Nguyen, and Yuan Fang. 2021. Tail-GNN: Tail-Node Graph Neural Networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1109–1119.

[21] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2019. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations*.

[22] Sancheng Peng, Guojun Wang, and Dongqing Xie. 2016. Social Influence Analysis in Social Networking Big Data: Opportunities and Challenges. *IEEE Network* 31, 1 (2016), 11–17.

[23] Tahleen A Rahman, Bartlomiej Surma, Michael Backes, and Yang Zhang. 2019. Fairwalk: Towards Fair Graph Embedding. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 3289–3295.

[24] Aida Rahmattalabi, Phebe Vayanos, Anthony Fulginiti, Eric Rice, Bryan Wilder, Amulya Yadav, and Milind Tambe. 2019. Exploring Algorithmic Fairness in Robust Graph Covering Problems. *Advances in Neural Information Processing Systems* 32 (2019), 15776–15787.

[25] John Rawls. 1971. *A Theory of Justice*. Harvard University Press.

[26] Richard Sinkhorn and Paul Knopp. 1967. Concerning Nonnegative Matrices and Doubly Stochastic Matrices. *Pacific J. Math.* 21, 2 (1967), 343–348.

[27] Xianfeng Tang, Huaxiu Yao, Yiwei Sun, Yiqi Wang, Jiliang Tang, Charu Aggarwal, Prasenjit Mitra, and Suhang Wang. 2020. Investigating and Mitigating Degree-Related Biases in Graph Convoltuional Networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1435–1444.

[28] Sotiris Tsioutsiouliklis, Evaggelia Pitoura, Panayiotis Tsaparas, Ilias Kleftakis, and Nikos Mamoulis. 2021. Fairness-Aware PageRank. In *Proceedings of the Web Conference 2021*. 3815–3826.

[29] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[30] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *Proceedings of the 42nd International ACM SIGIR conference on Research and Development in Information Retrieval*. 165–174.

[31] Jun Wu, Jingrui He, and Jiejun Xu. 2019. DEMO-Net: Degree-Specific Graph Neural Networks for Node and Graph Classification. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 406–415.

[32] Si Zhang, Dawei Zhou, Mehmet Yigit Yildirim, Scott Alcorn, Jingrui He, Hasan Davulcu, and Hanghang Tong. 2017. HiDDen: Hierarchical Dense Subgraph Detection with Application to Financial Fraud Detection. In *Proceedings of the 2017 SIAM International Conference on Data Mining*. 570–578.

APPENDIX

A – RawlsGCN-Graph vs. RawlsGCN-Grad

Here, we discuss the advantages and disadvantages of our proposed methods. We list the pros and cons of our proposed methods in Table 6. Moreover, though RawlsGCN-Graph and RawlsGCN-Grad have the same pre-processing procedure in the setting of fixed input graph, we believe that the general idea of normalizing the gradient in RawlsGCN-Grad is useful for distributed training of extremely large graphs, in which a local subgraph of each node is often sampled using a (non-)deterministic sampler for feature aggregation and gradient computation. In this setting, the input graph is not deterministic during training and often asymmetric. Consequently, it is often impossible to precompute the doubly stochastic matrix for RawlsGCN-Graph. However, we can still use the sampling distribution of local subgraph to calculate the normalized gradient using Eqs. (10) and (11).

Table 6: Pros and cons of our proposed methods.

	RawlsGCN-Graph	RawlsGCN-Grad
Pros	• No need to modify the GNN model; • Higher accuracy on graph with more diverse degree empirically; • Smaller bias than RawlsGCN-Grad.	• Higher accuracy on graph with less diversity in node degree empirically; • Able to work in use cases like distributed training of large graphs.
Cons	• Lower accuracy on smaller graph/graph with less diversity in node degree empirically; • May be unable to work in use cases like distributed training on extremely large graphs.	• Slightly higher bias than RawlsGCN-Graph; • Need to change the optimizer of model.

B – Descriptions of Baseline Methods

- **GCN** [18] refers to the original Graph Convolutional Network (GCN) without fairness considerations. In our experiment, we adopt the same architecture as in [18] but increasing the hidden dimension to 64 for a fair comparison.
- **DEMO-Net** [31] uses multi-task graph convolution where each task learns degree-specific node representations in order to preserve the degree-specific graph structure. We use DEMO-Net with degree-specific weight function instead of hashing function due to its higher classification accuracy and better stability. For a fair comparison, we remove the components for order-free and seed-oriented representation learning as they are irrelevant to fairness w.r.t. node degree.
- **DSGCN** [27] mitigates degree-related bias by degree-specific graph convolution, which infers the degree-specific weights using a Recurrent Neural Network (RNN). We use 2 degree-specific graph convolution layers in DSGCN with the same hidden dimension settings as the vanilla GCN. We set the number of RNN cell to 10 (i.e., nodes with degree larger than 10 will share the same degree-specific weight), which is consistent with [27]. We set the activation function of the RNN cell to tanh function. For a fair comparison, we only use the degree-specific graph convolution module (i.e., DSGCN in [27]) for our experiments.
- **Tail-GNN** [20] learns robust embedding for low-degree nodes (i.e., tail nodes) in an adversarial learning fashion with the novel neighborhood translation mechanism. It first generates forged tail nodes from nodes with degree higher than a certain threshold k. Then the neighborhood translation operation predicts the missing information of tail nodes and forged tail nodes by a translation model learned from head nodes. After that, a discriminator is applied to predict whether a node is head or tail based on the node representations. In our experiment, we set $k = 5$ for forged tail nodes generation. If a training node u has degree less than k, we do not generate the forged tail node using this training node.
- **Adversarial Fair GCN** (AdvFair) is a variant of [3] which ensures group fairness for graph embeddings in the compositional setting (i.e., for different combinations of sensitive attributes). We set the node degree as the sensitive attribute, i.e., nodes of the same degree form a demographic group. For a fair comparison, we compute the node embeddings using 2 graph convolution layers with ReLU activation, each of which has 64 hidden dimension. The 'filtered' embeddings are computed by the filter, which is a 2-layer multi-layer perceptron (MLP) with 128 and 64 hidden dimensions, respectively. The discriminator is a 2-layer MLP where the first layer contains 64 hidden dimensions and the second layer predicts the sensitive attribute of each node. Both the filter and the discriminator use leaky ReLU as the activation function. A multi-class logistic regression is applied on the 'filtered' embeddings for node classification.
- **REDRESS** [9] ensures individual fairness of graph neural network (GNN) by optimizing the similarity between the ranking lists of model input and output. In our experiment, we set the backbone GNN model as the vanilla GCN model described above.

Table 7: Additional effectiveness results for node classification on Chameleon dataset. Lower is better for bias (the gray column). Higher is better for accuracy (Acc., in percentage).

Method	Chameleon	
	Acc.	Bias
GCN	60.09 ± 1.047	0.504 ± 0.118
DEMO-Net	63.77 ± 0.955	0.352 ± 0.015
DSGCN	47.76 ± 1.978	0.129 ± 0.019
Tail-GNN	58.73 ± 1.794	1.040 ± 0.654
AdvFair	42.54 ± 8.499	0.276 ± 0.194
REDRESS	23.77 ± 2.745	0.020 ± 0.005
RawlsGCN-Graph (Ours)	50.61 ± 0.526	0.098 ± 0.007
RawlsGCN-Grad (Ours)	45.92 ± 2.741	0.138 ± 0.062

C – Parameter Settings

In this section, we provide additional parameter settings for the purpose of reproducibility, including the settings for learning rate and training epochs of baseline methods. For RawlsGCN-Graph,

Table 8: Additional ablation study of different matrix normalization techniques. Lower is better for bias (i.e., the gray column). Higher is better for accuracy (Acc.).

Method	Normalization	Cora-ML		Citeseer		Coauthor-CS	
		Acc.	Bias	Acc.	Bias	Acc.	Bias
RawlsGCN-Graph	**Row**	79.74 ± 0.320	0.098 ± 0.004	69.18 ± 0.595	0.240 ± 0.013	92.78 ± 0.331	0.052 ± 0.002
	Column	76.78 ± 1.360	0.260 ± 0.330	69.12 ± 0.781	0.243 ± 0.093	92.44 ± 0.609	0.049 ± 0.012
	Symmetric	77.04 ± 1.606	0.109 ± 0.015	69.20 ± 0.735	0.196 ± 0.014	92.56 ± 0.120	0.042 ± 0.001
	Doubly Stochastic	76.98 ± 1.098	0.105 ± 0.012	69.34 ± 0.745	0.196 ± 0.013	92.52 ± 0.264	0.043 ± 0.002
RawlsGCN-Grad	**Row**	79.78 ± 0.349	0.230 ± 0.017	68.64 ± 0.215	0.274 ± 0.036	92.92 ± 0.440	0.069 ± 0.006
	Column	79.94 ± 0.599	0.253 ± 0.077	68.48 ± 0.204	0.302 ± 0.049	92.78 ± 0.407	0.058 ± 0.006
	Symmetric	79.68 ± 0.458	0.199 ± 0.008	68.68 ± 0.248	0.286 ± 0.042	93.00 ± 0.341	0.063 ± 0.006
	Doubly Stochastic	79.34 ± 1.247	0.232 ± 0.065	68.81 ± 0.462	0.283 ± 0.047	92.68 ± 0.240	0.058 ± 0.007

Method	Normalization	Coauthor-Physics		Amazon-Computers	
		Acc.	Bias	Acc.	Bias
RawlsGCN-Graph	**Row**	94.36 ± 0.488	0.013 ± 0.000	78.54 ± 1.125	0.092 ± 0.013
	Column	93.98 ± 0.508	0.016 ± 0.003	78.18 ± 4.354	0.196 ± 0.106
	Symmetric	93.98 ± 0.248	0.016 ± 0.000	80.22 ± 0.803	0.126 ± 0.012
	Doubly Stochastic	94.06 ± 0.196	0.016 ± 0.000	80.16 ± 0.859	0.121 ± 0.010
RawlsGCN-Grad	**Row**	94.08 ± 0.204	0.027 ± 0.001	63.46 ± 1.376	0.453 ± 0.039
	Column	94.26 ± 0.294	0.020 ± 0.002	75.48 ± 1.273	0.218 ± 0.033
	Symmetric	94.30 ± 0.346	0.021 ± 0.001	66.42 ± 0.584	0.353 ± 0.021
	Doubly Stochastic	94.18 ± 0.306	0.021 ± 0.002	74.18 ± 2.530	0.195 ± 0.029

RawlsGCN-Grad and Adversarial Fair GCN, we search the learning rate that achieves the highest average classification accuracy in the set of $\{0.075, 0.05, 0.025, 0.01, 0.0075, 0.005, 0.0025\}$. For DSGCN, due to its long running time, we search the learning rate in the set of $\{0.05, 0.025, 0.01, 0.005\}$. For GCN, DEMO-Net, Tail-GNN and REDRESS, we use the suggested hyperparameters (including learning rate, weight decay, number of epochs and early stopping conditions) in the released source code. In addition, for REDRESS, we search the best choice of α (see details in [9]) in the range of $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. For Adversarial Fair GCN, we train the model for 1000 epochs with a patience of 200 for early stopping. For DSGCN, due to long training time, we train the model for 200 epochs without early stopping, which is consistent with the settings in GCN.

D – Additional Results on Heterophilic Graph

Different from datasets listed in Section 4.1, a heterophilic graph consists of linked nodes that are likely to have dissimilar features or different class labels. We conduct additional experiments on a commonly used heterophilic graph named Chameleon dataset [21]. It contains 2, 277 nodes which are Wikipedia pages about chameleons. Each node has 2, 325 features which correspond to informative nouns in the Wikipedia pages. Two nodes are connected if they have mutual link(s) between two pages, which forms 36, 101 edges. We follow the same experimental settings in Section 4.1 for (1) generating training/validation/test sets, (2) training model and (3) evaluating results. The experimental result on Chameleon dataset is listed in Table 7. From the table, we have the following observations. (1) Though REDRESS achieves smaller bias than RawlsGCN-Graph and RawlsGCN-Grad, its accuracy is severely reduced compared to GCN; (2) Though DSGCN outperforms RawlsGCN-Grad with slightly higher accuracy and smaller bias, it fails to outperform RawlsGCN-Graph; (3) Except for these two cases, our proposed methods achieve the smallest bias compared with all other baseline methods. Overall, our methods still achieve the best trade-off between the accuracy and bias.

E – Additional Ablation Study Results

We provide additional ablation study on other datasets listed in Section 4.1. From the Table 8, we observe that, although row normalization and symmetric normalization outperforms the doubly stochastic normalization in some cases, it can also increase the bias in other cases, e.g., Amazon-Computers and Amazon-Photo (shown in Table 5 in Section 4.1). All in all, the doubly stochastic normalization is the best one that (1) consistently mitigates bias for both RawlsGCN-Graph and RawlGCN-Grad, and (2) achieves good balance between accuracy and fairness.