Graph Neural Networks with Heterophily

Jiong Zhu,[1] **Ryan A. Rossi,**[2] **Anup Rao,**[2] **Tung Mai,**[2]
Nedim Lipka,[2] **Nesreen K. Ahmed,**[3] **Danai Koutra**[1]
[1]University of Michigan, Ann Arbor, USA
[2]Adobe Research, San Jose, USA
[3]Intel Labs, Santa Clara, USA
jiongzhu@umich.edu, {ryrossi, anuprao, tumai, lipka}@adobe.com, nesreen.k.ahmed@intel.com, dkoutra@umich.edu

Abstract

Graph Neural Networks (GNNs) have proven to be useful for many different practical applications. However, many existing GNN models have implicitly assumed homophily among the nodes connected in the graph, and therefore have largely overlooked the important setting of heterophily, where most connected nodes are from different classes. In this work, we propose a novel framework called CPGNN that generalizes GNNs for graphs with either homophily or heterophily. The proposed framework incorporates an interpretable compatibility matrix for modeling the heterophily or homophily level in the graph, which can be learned in an end-to-end fashion, enabling it to go beyond the assumption of strong homophily. Theoretically, we show that replacing the compatibility matrix in our framework with the identity (which represents pure homophily) reduces to GCN. Our extensive experiments demonstrate the effectiveness of our approach in more realistic and challenging experimental settings with significantly less training data compared to previous works: CPGNN variants achieve state-of-the-art results in heterophily settings with or without contextual node features, while maintaining comparable performance in homophily settings.

1 Introduction

As a powerful approach for learning and extracting information from relational data, Graph Neural Network (GNN) models have gained wide research interest (Scarselli et al. 2008) and been adapted in applications including recommendation systems (Ying et al. 2018), bioinformatics (Zitnik, Agrawal, and Leskovec 2018; Yan et al. 2019), fraud detection (Dou et al. 2020), and more. While many different GNN models have been proposed, existing methods have largely overlooked several limitations in their formulations: (1) *implicit homophily assumptions*; (2) *heavy reliance on contextual node features*. First, many GNN models, including the most popular GNN variant proposed by Kipf and Welling (2017), implicitly assume *homophily* in the graph, where most connections happen among nodes in the same class or with alike features (McPherson, Smith-Lovin, and Cook 2001). This assumption has affected the design of many GNN models, which tend to generate similar representations for nodes within close proximity, as studied in previous works (Ahmed et al. 2018; Rossi et al. 2020; Wu et al. 2019). However, there are also many instances in the real world where nodes of different classes are more likely to connect with one another — in idiom, this phenomenon can be described as "opposites attract". As we observe empirically, many GNN models which are designed under implicit homophily assumptions suffer from poor performance in heterophily settings, which can be problematic for applications like fraudster detection (Pandit et al. 2007; Dou et al. 2020) or analysis of protein structures (Fout et al. 2017), where heterophilous connections are common. Second, many existing models rely heavily on contextual node features to derive intermediate representations of each node, which is then propagated within the graph. While in a few networks like citation networks, node features are able to provide powerful node-level contextual information for downstream applications, in more common cases the contextual information are largely missing, insufficient or incomplete, which can significantly degrade the performance for some models. Moreover, complex transformation of input features usually requires the model to adopt a large number of learnable parameters, which need more data and computational resources to train and are hard to interpret.

In this work, we propose CPGNN, a novel approach that incorporates into GNNs a compatibility matrix that captures both heterophily and homophily by modeling the likelihood of connections between nodes in different classes. This novel design overcomes the drawbacks of existing GNNs mentioned above: it enables GNNs to appropriately learn from graphs with either homophily *or* heterophily, and is able to achieve satisfactory performance even in the cases of missing and incomplete node features. Moreover, the end-to-end learning of the class compatibility matrix effectively recovers the ground-truth underlying compatibility information, which is hard to infer from limited training data, and provides insights for understanding the connectivity patterns within the graph. Finally, the key idea proposed in this work can naturally be used to generalize many other GNN-based methods by incorporating and learning the heterophily compatibility matrix $\mathbf{H}$ in a similar fashion.

We summarize the main contributions as follows:

- **Heterophily Generalization of GNNs**. We describe a generalization of GNNs to heterophily settings by incorporating a compatibility matrix $\mathbf{H}$ into GNN-based methods,

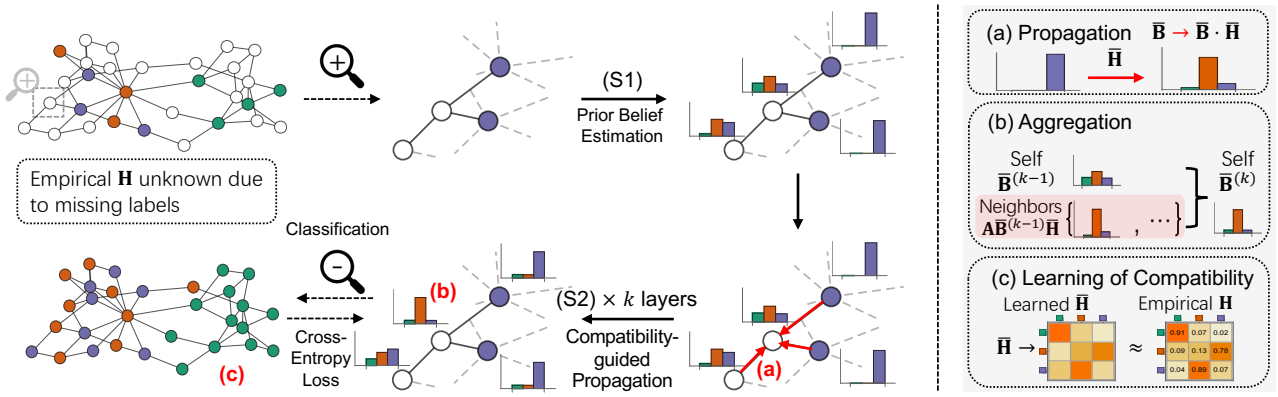


Figure 1: The general pipeline of the proposed framework (CPGNN) with k propagation layers (§2.2). As an example, we use a graph with mixed homophily and heterophily, with node colors representing class labels: nodes in green show strong *homophily*, while nodes in orange and purple show strong *heterophily*. CPGNN framework first generates prior belief estimations using an off-the-shelf neural network classifier, which utilizes node features if available (S1). The prior beliefs are then propagated within their neighborhoods guided by the learned compatibility matrix $\mathbf{H}$, and each node aggregates beliefs sent from its neighbors to update its own beliefs (S2). We describe the backward training process, including how $\mathbf{H}$ can be learned end-to-end in §2.3.

which is learned in an end-to-end fashion.

- **CPGNN Framework**. We propose CPGNN, a novel approach that directly models and learns the class compatibility matrix $\mathbf{H}$ in GNN-based methods. This formulation gives rise to many advantages including better effectiveness for graphs with either homophily or heterophily, and for graphs with or without node features. We release CPGNN at https://github.com/GemsLab/CPGNN.
- **Comprehensive Evaluation**. We conduct extensive experiments to compare the performance of CPGNN with baseline methods under a more *realistic* experimental setup by using significantly fewer training data comparing to the few previous works which address heterophily (Pei et al. 2020; Zhu et al. 2020). These experiments demonstrate the effectiveness of incorporating the compatibility matrix $\mathbf{H}$ into GNN-based methods.

2 Framework

In this section we introduce our CPGNN framework, after presenting the problem setup and important definitions.

2.1 Preliminaries

Problem Setup. We focus on the problem of semi-supervised node classification on a simple graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the node- and edge-sets respectively, and $\mathcal{Y}$ is the set of possible class labels (or types) for $v \in \mathcal{V}$. Given a training set $\mathcal{T}_\mathcal{V} \subset \mathcal{V}$ with known class labels y_v for all $v \in \mathcal{T}_\mathcal{V}$, and (optionally) contextual feature vectors $\mathbf{x}_v$ for $v \in \mathcal{V}$, we aim to infer the unknown class labels y_u for all $u \in (\mathcal{V} - \mathcal{T}_\mathcal{V})$. For subsequent discussions, we use $\mathbf{A} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$ for the adjacency matrix *with self-loops removed*, $\mathbf{y} \in \mathcal{Y}^{|\mathcal{V}|}$ as the ground-truth class label vector for all nodes, and $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}|\times F}$ for the node feature matrix.

Definitions. We now introduce two key concepts for modeling the homophily level in the graph with respect to the class labels: (1) *homophily ratio*, and (2) *compatibility matrix*.

Definition 1 (Homophily Ratio h). *Let* $\mathbf{C} \in \mathbb{R}^{|\mathcal{Y}|\times|\mathcal{Y}|}$ *where* $C_{ij} = |\{(u,v) : (u,v) \in \mathcal{E} \wedge y_u = i \wedge y_v = j\}|$, $\mathbf{D} = \mathrm{diag}(\{C_{ii} : i = 1, \ldots, |\mathcal{Y}|\})$, *and* $\mathbf{e} \in \mathbb{R}^{|\mathcal{V}|}$ *be an all-ones vector. The homophily ratio is defined as* $h = \frac{\mathbf{e}^\top \mathbf{D}\mathbf{e}}{\mathbf{e}^\top \mathbf{C}\mathbf{e}}$.

The homophily ratio h defined above is good for measuring the *overall* homophily level in the graph. By definition, we have $h \in [0,1]$: graphs with h closer to 1 tend to have more edges connecting nodes within the same class, or *stronger homophily*; on the other hand, graphs with h closer to 0 have more edges connecting nodes in different classes, or a *stronger heterophily*. However, the actual homophily level is not necessarily uniform within all parts of the graph. One common case is that the homophily level varies among different pairs of classes, where it is more likely for nodes between some pair of classes to connect than some other pairs. To measure the variability of the homophily level, we define the *compatibility matrix* $\mathbf{H}$ as follows:

Definition 2 (Compatibility Matrix H). *Let* $\mathbf{Y} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{Y}|}$ *where* $Y_{vj} = 1$ *if* $y_v = j$, *and* $Y_{vj} = 0$ *otherwise. Then, the compatibility matrix* $\mathbf{H}$ *is defined as:*

$$\mathbf{H} = (\mathbf{Y}^\top \mathbf{A}\mathbf{Y}) \oslash (\mathbf{Y}^\top \mathbf{A}\mathbf{E}) \tag{1}$$

where $\oslash$ *is Hadamard (element-wise) division and* $\mathbf{E}$ *is a* $|\mathcal{V}| \times |\mathcal{V}|$ *all-ones matrix.*

In node classification settings, compatibility matrix $\mathbf{H}$ models the (empirical) probability of nodes belonging to each pair of classes to connect. More generally, $\mathbf{H}$ can be used to model any discrete attribute; in that case, $\mathbf{H}_{ij}$ is the probability that a node with attribute value i connects with a node with value j. Modeling $\mathbf{H}$ in GNNs is beneficial for heterophily settings, but calculating the exact $\mathbf{H}$ would require knowledge to the class labels of all nodes in the graph, which violates the semi-supervised node classification setting. Therefore, it is not possible to incorporate exact $\mathbf{H}$ into graph neural networks. In the following sections, we propose CPGNN, which is capable of learning $\mathbf{H}$ in an end-to-end way based on a rough initial estimation.

2.2 Framework Design

The CPGNN framework consists of two stages: (S1) prior belief estimation; and (S2) compatibility-guided propagation. We visualize the CPGNN framework in Fig. 1.

(S1) Prior Belief Estimation The goal for the first step is to estimate per node $v \in \mathcal{V}$ a prior belief $\mathbf{b}_v \in \mathbb{R}^{|\mathcal{Y}|}$ of its class label $y_v \in \mathcal{Y}$ from the node features $\mathbf{X}$. This separate, explicit prior belief estimation stage enables the use of any off-the-shelf neural network classifier as the estimator, and thus can accommodate different types of node features. In this work, we consider the following models as the estimators:

- `MLP`, a graph-agnostic multi-layer perceptron. Specifically, the k-th layer of the MLP can be formulated as following:
$$\mathbf{R}^{(k)} = \sigma(\mathbf{R}^{(k-1)}\mathbf{W}^{(k)}), \tag{2}$$
where $\mathbf{W}^{(k)}$ are learnable parameters, and $\mathbf{R}^{(0)} = \mathbf{X}$. We call our method with MLP-based estimator `CPGNN-MLP`.
- `GCN-Cheby` (Defferrard, Bresson, and Vandergheynst 2016). We instantiate the model using a 2[nd]-order Chebyshev polynomial, where the k-th layer is parameterized as:
$$\mathbf{R}^{(k)} = \sigma\Big(\textstyle\sum_{i=0}^{2} T_i(\tilde{\mathbf{L}})\mathbf{R}^{(k-1)}\mathbf{W}_i^{(k)}\Big). \tag{3}$$
$\mathbf{W}_i^{(k)}$ are learnable parameters, $\mathbf{R}^{(0)} = \mathbf{X}$, and $T_i(\tilde{\mathbf{L}})$ is the i-th order of the Chebyshev polynomial of $\tilde{\mathbf{L}} = \mathbf{L} - \mathbf{I}$ defined recursively as:
$$T_i(\tilde{\mathbf{L}}) = 2\tilde{\mathbf{L}}T_{i-1}(\tilde{\mathbf{L}}) - T_{i-2}(\tilde{\mathbf{L}})$$
with $T_0(\tilde{\mathbf{L}}) = \mathbf{I}$ and $T_1(\tilde{\mathbf{L}}) = \tilde{\mathbf{L}} = -\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$. We refer to our Cheby-based method as `CPGNN-Cheby`.

We note that the performance of CPGNN is affected by the choice of the estimator. It is important to choose an estimator that is *not* constrained by the homophily assumption (e.g., our above-mentioned choices), so that it does not hinder the performance in heterophilous graphs.

Denote the output of the final layer of the estimator as $\mathbf{R}^{(K)}$, then the prior belief $\mathbf{B}_p$ of nodes can be given as
$$\mathbf{B}_p = \mathrm{softmax}(\mathbf{R}^{(K)}) \tag{4}$$
To facilitate subsequent discussions, we denote the trainable parameters of a general prior belief estimator as $\mathbf{\Theta}_p$, and the prior belief of node v derived by the estimator as $\mathcal{B}_p(v; \mathbf{\Theta}_p)$.

(S2) Compatibility-guided Propagation We propagate the prior beliefs of nodes within their neighborhoods using a parameterized, end-to-end trainable compatibility matrix $\bar{\mathbf{H}}$.

To propagate the belief vectors through linear formulations, following Gatterbauer et al. (2015), we first center $\mathbf{B}_p$ with
$$\bar{\mathbf{B}}^{(0)} = \mathbf{B}_p - \tfrac{1}{|\mathcal{Y}|} \tag{5}$$
We parameterize the compatibility matrix as $\bar{\mathbf{H}}$ to replace the weight matrix $\mathbf{W}$ in traditional GNN models as the end-to-end trainable parameter. We formulate each layer as:
$$\bar{\mathbf{B}}^{(k)} = \bar{\mathbf{B}}^{(0)} + \mathbf{A}\bar{\mathbf{B}}^{(k-1)}\bar{\mathbf{H}} \tag{6}$$
Each layer propagates and updates the current belief per node in its neighborhood. After K layers, we have the final belief
$$\mathbf{B}_f = \mathrm{softmax}(\bar{\mathbf{B}}^{(K)}). \tag{7}$$
We similarly denote $\mathcal{B}_f(v; \bar{\mathbf{H}}, \mathbf{\Theta}_p)$ as the final belief for node v, where parameters $\mathbf{\Theta}_p$ are from the prior belief estimator.

2.3 Training Procedure

Pretraining of Prior Belief Estimator. We pretrain the prior belief estimator for β_1 iterations so that $\bar{\mathbf{H}}$ can then be better initialized with informative prior beliefs. Specifically, the pretraining process aims to minimize the loss function
$$\mathcal{L}_p(\mathbf{\Theta}_p) = \textstyle\sum_{v\in\mathcal{T}_\mathcal{V}} \mathcal{H}\left(\mathcal{B}_p(v; \mathbf{\Theta}_p), y_v\right) + \lambda_p\|\mathbf{\Theta}_p\|_2, \tag{8}$$
where $\mathcal{H}$ corresponds to the cross entropy, and λ_p is the L2 regularization weight for the prior belief estimator. Through an ablation study (App. §D, Fig. 5), we show that pretraining prior belief estimator helps increase the final performance.

Initialization and Regularization of $\bar{\mathbf{H}}$. We empirically found that initializing the parameters $\bar{\mathbf{H}}$ with an estimation of the unknown compatibility matrix $\mathbf{H}$ can lead to better performance (cf. §4.4, Fig. 4a). We derive the estimation using node labels in the training set $\mathcal{T}_\mathcal{V}$, and prior belief $\mathbf{B}_p$ estimated in Eq. (4) after the pretraining stage. More specifically, denote the training mask matrix $\mathbf{M}$ as:
$$[\mathbf{M}]_{i,:} = \begin{cases} \mathbf{1}, & \text{if } i \in \mathcal{T}_\mathcal{V} \\ \mathbf{0}, & \text{otherwise} \end{cases} \tag{9}$$
and the enhanced belief matrix $\tilde{\mathbf{B}}$, which makes use of known node labels $\mathbf{Y}_{\text{train}} = \mathbf{M} \circ \mathbf{Y}$ in the training set $\mathcal{T}_\mathcal{V}$, as
$$\tilde{\mathbf{B}} = \mathbf{Y}_{\text{train}} + (1 - \mathbf{M}) \circ \mathbf{B}_p \tag{10}$$
where $\circ$ is the element-wise product. The estimation $\hat{\mathbf{H}}$ of the unknown compatibility matrix $\mathbf{H}$ is derived as
$$\hat{\mathbf{H}} = \text{Sinkhorn-Knopp}\left(\mathbf{Y}_{\text{train}}^\top \mathbf{A}\tilde{\mathbf{B}}\right) \tag{11}$$
where the use of the Sinkhorn and Knopp (1967) algorithm is to ensure that $\hat{\mathbf{H}}$ is doubly stochastic. We find that a doubly-stochastic and symmetric initial value for $\bar{\mathbf{H}}$ boosts the training when using multiple propagation layers. Thus, we initialize the parameter $\bar{\mathbf{H}}$ as $\bar{\mathbf{H}}_0 = \frac{1}{2}(\hat{\mathbf{H}} + \hat{\mathbf{H}}^\top) - \frac{1}{|\mathcal{Y}|}$, where $\hat{\mathbf{H}}$ is centered around 0 (similar to $\mathbf{B}_p$). To ensure the rows of $\bar{\mathbf{H}}$ remain centered around 0 throughout the training process, we adopt the following regularization term $\Phi(\bar{\mathbf{H}})$ for $\bar{\mathbf{H}}$:
$$\Phi(\bar{\mathbf{H}}) = \textstyle\sum_i \left|\sum_j \bar{\mathbf{H}}_{ij}\right| \tag{12}$$

Loss Function for CPGNN Training. Putting everything together, we obtain the loss function for training CPGNN:
$$\mathcal{L}_f(\bar{\mathbf{H}}, \mathbf{\Theta}_p) = \sum_{v\in\mathcal{T}_\mathcal{V}} \mathcal{H}\left(\mathcal{B}_f(v; \bar{\mathbf{H}}, \mathbf{\Theta}_p), y_v\right) + \eta\mathcal{L}_p(\mathbf{\Theta}_p) + \Phi(\bar{\mathbf{H}}) \tag{13}$$
The loss function consists of three parts: (1) the cross entropy loss from the CPGNN output; (2) the co-training loss from the prior belief estimator; and (3) a regularization term that keeps $\bar{\mathbf{H}}$ centered around 0 throughout the training process. The latter two terms are novel for the CPGNN formulation, and help increase the performance of CPGNN, as we show later through an ablation study (§4.4). Intuitively, our separate co-training term for the prior belief estimator measures the distance of *prior* beliefs to the ground-truth distribution for nodes in the training set while also optimizing the *final* beliefs. In other words, the second term helps keep the accuracy of the *prior* beliefs throughout the training process.

2.4 Interpretation of Parameters $\bar{\mathbf{H}}$

Unlike the hard-to-interpret weight matrix $\mathbf{W}$ in classic GNNs, parameter $\bar{\mathbf{H}}$ in CPGNN can be easily understood: it captures the probability that node pairs in specific classes connect with each other. Through an inverse of the initialization process, we can obtain an estimation of the compatibility matrix $\hat{\mathbf{H}}$ after training from learned parameter $\bar{\mathbf{H}}$ as follows:

$$\hat{\mathbf{H}} = \text{Sinkhorn-Knopp}(\tfrac{1}{\alpha}\bar{\mathbf{H}} + \tfrac{1}{|\mathcal{Y}|}) \tag{14}$$

where $\alpha = \min\{a \geq 1 : 1 \geq \frac{1}{a}\bar{\mathbf{H}} + \frac{1}{|\mathcal{Y}|} \geq 0\}$ is a recalibration factor ensuring that the obtained $\hat{\mathbf{H}}$ is a valid stochastic matrix. In §4.5, we provide an example of the estimated $\hat{\mathbf{H}}$ after training, and show the improvements in estimation error compared to the initial estimation by Eq. (11).

3 Theoretical Analysis

Theoretical Connections. Theorem 1 establishes the theoretical result that CPGNN can be reduced to a simplified version of GCN when $\mathbf{H} = \mathbf{I}$. Intuitively, replacing $\mathbf{H}$ with $\mathbf{I}$ indicates a pure homophily assumption, and thus shows exactly the reason that GCN-based methods have a strong homophily assumption built-in, and therefore perform worse for graphs *without* strong homophily.

Theorem 1. *The forward pass formulation of a 1-layer SGC (Wu et al. 2019), a simplified version of GCN without the non-linearities and adjacency matrix normalization,*

$$\mathbf{B}_f = \text{softmax}\left((\mathbf{A} + \mathbf{I})\,\mathbf{X}\boldsymbol{\Theta}\right) \tag{15}$$

where $\boldsymbol{\Theta}$ denotes the model parameter, can be treated as a special case of CPGNN with compatibility matrix $\mathbf{H}$ fixed as $\mathbf{I}$ and non-linearity removed in the prior belief estimator.

Proof The formulation of CPGNN with 1 aggregation layer can be written as follows:

$$\mathbf{B}_f = \text{softmax}(\bar{\mathbf{B}}^{(1)}) = \text{softmax}\left(\bar{\mathbf{B}}^{(0)} + \mathbf{A}\bar{\mathbf{B}}^{(0)}\mathbf{H}\right) \tag{16}$$

Now consider a 1-layer MLP (Eq. (2)) as the prior belief estimator. Since we assumed that the non-linearity is removed in the prior belief estimator, we can assume that $\mathbf{B}_p$ is already centered. Therefore, we have

$$\bar{\mathbf{B}}^{(0)} = \mathbf{B}_p = \mathbf{R}^{(K)} = \mathbf{R}^{(0)}\mathbf{W}^{(0)} = \mathbf{X}\mathbf{W}^{(0)} \tag{17}$$

where $\mathbf{W}^{(0)}$ is the trainable parameter for MLP. Plug in Eq. (17) into Eq. (16), we have

$$\mathbf{B}_f = \text{softmax}\left(\mathbf{X}\mathbf{W}^{(0)} + \mathbf{A}\mathbf{X}\mathbf{W}^{(0)}\mathbf{H}\right) \tag{18}$$

Fixing compatibility matrix $\mathbf{H}$ fixed as $\mathbf{I}$, and we have

$$\mathbf{B}_f = \text{softmax}\left((\mathbf{A} + \mathbf{I})\mathbf{X}\mathbf{W}^{(0)}\right) \tag{19}$$

As $\mathbf{W}^{(0)}$ is a trainable parameter equivalent to $\boldsymbol{\Theta}$ in Eq. (15), the notation is interchangeable. Thus, the simplified GCN formulation as in Eq. (15) can be reduced to a special case of CPGNN with compatibility matrix $\mathbf{H} = \mathbf{I}$. ■

Time and Space Complexity of CPGNN Let $|\mathcal{E}|$ and $|\mathcal{V}|$ denote the number of edges and nodes in $\mathcal{G}$, respectively. Further, let $|\mathcal{E}_i|$ denote the number of node pairs in $\mathcal{G}$ within i-hop distance (*e.g.*, $|\mathcal{E}_1| = |\mathcal{E}|$) and $|\mathcal{Y}|$ denotes the number of unique class labels. We assume the graph adjacency matrix $\mathbf{A}$ and node feature matrix $\mathbf{X}$ are stored as sparse matrices.

CPGNN only introduces $O(|\mathcal{E}||\mathcal{Y}|^2)$ extra time over the selected prior belief estimator in the propagation stage (S2). Therefore, the overall complexity for CPGNN is largely determined by the time complexity of the selected prior belief estimator: when using MLP as prior belief estimator (Stage S1), the overall time complexity of CPGNN-MLP is $O(|\mathcal{E}||\mathcal{Y}|^2 + |\mathcal{V}||\mathcal{Y}| + \text{nnz}(\mathbf{X}))$, while the overall time complexity of an α-order CPGNN-Cheby is $O(|\mathcal{E}||\mathcal{Y}|^2 + |\mathcal{V}||\mathcal{Y}| + \text{nnz}(\mathbf{X}) + |\mathcal{E}_{\alpha-1}|d_{\max} + |\mathcal{E}_\alpha|)$, where $d_{\max}$ is the max degree of a node in $\mathcal{G}$ and $\text{nnz}(\mathbf{X})$ is the number of nonzeros in $\mathbf{X}$.

The overall space complexity of CPGNN is $O(|\mathcal{E}| + |\mathcal{V}||\mathcal{Y}| + |\mathcal{Y}|^2 + \text{nnz}(\mathbf{X}))$, which also takes into account the space complexity for the two discussed prior belief estimators above (MLP and GCN-Cheby).

4 Experiments

We design experiments to investigate the effectiveness of the proposed framework for node classification with *and* without contextual features using both synthetic and real-world graphs with heterophily and strong homophily.

4.1 Methods and Datasets

Methods. We test the two formulations discussed in §2.2: `CPGNN-MLP` and `CPGNN-Cheby`. Each formulation is tested with either 1 or 2 aggregation layers, leading to 4 variants in total. We compared our methods with the following baselines, some of which are reported to be competitive under heterophily (Zhu et al. 2020): GCN (Kipf and Welling 2017), GAT (Veličković et al. 2018), GCN-Cheby (Defferrard, Bresson, and Vandergheynst 2016; Kipf and Welling 2017), GraphSAGE (Hamilton, Ying, and Leskovec 2017), MixHop (Abu-El-Haija et al. 2019), and H_2GCN (Zhu et al. 2020). We also consider MLP as a graph-agnostic baseline. We provide hardware and software specifications and details on hyperparameter tuning in App. B and C.

Datasets. We investigate CPGNN using both synthetic and real-world graphs. For synthetic benchmarks, we generate graphs and node labels following an approach similar to Karimi et al. (2017) and Abu-El-Haija et al. (2019), which expands the Barabási-Albert model with configurable class compatibility settings. We assign to the nodes feature vectors from the recently announced Open Graph Benchmark (Hu

Table 1: Statistics for our synthetic and real graphs.

Dataset	#Nodes $\lvert\mathcal{V}\rvert$	#Edges $\lvert\mathcal{E}\rvert$	#Classes $\lvert\mathcal{Y}\rvert$	#Features F	Homophily h
syn-products	10,000	59,640–59,648	10	100	[0, 0.1, …, 1]
Texas	183	295	5	1703	0.11
Squirrel	5,201	198,493	5	2,089	0.22
Chameleon	2,277	31,421	5	2,325	0.23
CiteSeer	3,327	4,676	7	3,703	0.74
Pubmed	19,717	44,327	3	500	0.8
Cora	2,708	5,278	6	1,433	0.81

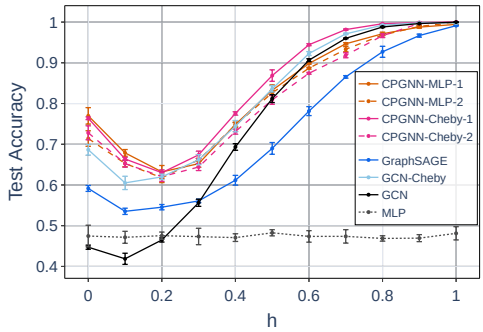


Figure 2: Mean classification accuracy of CPGNN and baselines on synthetic benchmark `syn-products` (cf. Table A.1 for detailed results).

et al. 2020), which includes only graphs with homophily. We detail the algorithms for generating synthetic benchmarks in App. A. For real-world graph data, we consider graphs with heterophily and homophily. We use 3 heterophily graphs, namely Texas, Squirrel and Chameleon (Rozemberczki, Allen, and Sarkar 2019), and 3 widely adopted graphs with strong homophily, which are Cora, Pubmed and Citeseer (Sen et al. 2008; Namata et al. 2012). We use the features and class labels provided by Pei et al. (2020).

4.2 Node Classification *with* Contextual Features

Experimental Setup. For synthetic experiments, we generate 3 synthetic graphs for every heterophily level $h \in \{0, 0.1, 0.2, \ldots, 0.9, 1\}$. We then randomly select 10% of nodes in each class for training, 10% for validation, and 80% for testing, and report the average classification accuracy as performance of each model on all instances with the same level of heterophily. Using synthetic graphs for evaluation enables us to better understand how the model performance changes as a function of the level of heterophily in the graph. Hence, we vary the level of heterophily in the graph going from strong heterophily all the way to strong homophily while holding other factors constant such as degree distribution and differences in contextual features. On real-world graphs, we generate 10 random splits for training, validation and test sets; for each split we randomly select 10% of nodes in each class to form the training set, with another 10% for the validation set and the remaining as the test set. Notice that we are using a significantly smaller fraction of training samples compared to previous works that address heterophily (Pei et al. 2020; Zhu et al. 2020). This is a more realistic assumption in many real-world applications.

Synthetic Benchmarks. We compare the performance of CPGNN to the state-of-the-art methods in Fig. 2. Notably, we observe that CPGNN-Cheby-1 consistently outperforms the baseline methods across the full spectrum of low to high homophily (or high to low heterophily). Furthermore, compared to our CPGNN variants, it performs the best in all settings with $h \geq 0.3$. For $h < 0.3$, CPGNN-MLP-1 outperforms it, and in fact performs the best overall for graphs with strong heterophily. More importantly, CPGNN has a significant performance improvement over the baseline methods. In particular, by incorporating and learning the class compatibility matrix $\mathbf{H}$ in an end-to-end fashion, we find that CPGNN-Cheby-1 achieves a gain of up to 7% compared to GCN-Cheby in heterophily settings, while CPGNN-MLP-1 performs up to 30% better in heterophily and 50% better in homophily compared to the graph-agnostic MLP model.

Real-World Graphs with Heterophily. Results for graphs with heterophily are presented in Table 2. Overall, we observe that the top-3 methods in mean accuracy across all the graphs are all based on CPGNN, which demonstrates the importance of incorporating and learning the compatibility matrix $\mathbf{H}$ into GNNs. Notably, CPGNN-Cheby-1 performs the best overall and significantly outperforms the other baseline methods, achieving improvements between 1.68% and 10.64% in mean accuracy compared to GNN baselines. These results demonstrate the effectiveness of CPGNN in heterophily settings on real-world benchmarks. We note that our empirical analysis also confirms the small time complexity overhead of CPGNN: on the Squirrel dataset, the runtimes of CPGNN-MLP-1 and CPGNN-Cheby-1 are 39s and 697s, respectively, while the prior belief estimators, MLP and GCN-Cheby, run in 29s and 592s in our implementation.

Real-World Graphs with Homophily. For the real-world graphs with homophily, we report the results for each method in Table 3. Recall that our framework generalizes GNN for both homophily and heterophily. We find in Table 3, the methods from the proposed framework perform better or comparable to the baselines, including those which have an implicit assumption of strong homophily. Therefore, our methods are more universal while able to maintain the same level of performance as those that are optimized under a strict homophily assumption. As an aside, we observe that CPGNN-Cheby-1 is the best performing method on Pubmed.

Summary. For the common settings of semi-supervised node classification with contextual features available, the above results show that CPGNN variants have the best performance in heterophily settings while maintaining comparable performance in the homophily settings. Considering both the heterophily and homophily settings, CPGNN-Cheby-1 is the best method overall, which ranked first in the heterophily settings and second in homophily settings.

4.3 Node Classification *without* Features

Most previous work on semi-supervised node classification have focused only on graphs that have contextual features on the nodes. However, the vast majority of graph data does not have such node-level features (Rossi and Ahmed 2015), which greatly limits the utility of the methods proposed in prior work that assume such features are available. Therefore, we conduct extensive experiments on semi-supervised node classification without contextual features using the same real-world graphs as before.

Experimental Setup. To investigate the performance of CPGNN and baselines when contextual feature vectors are not available for nodes in the graph, we follow the approach as Kipf and Welling (2017) by replacing the node features $\mathbf{X}$ in each benchmark with an identity matrix $\mathbf{I}$. We use the training, validation and test splits provided by Pei et al. (2020).

Table 2: Accuracy on heterophily graphs *with* features.

Hom. ratio h	Texas 0.11	Squirrel 0.22	Chameleon 0.23	Mean Acc
CPGNN-MLP-1	63.75±4.74	32.70±1.90	51.08±2.29	49.18
CPGNN-MLP-2	70.42±2.97	26.64±1.23	55.46±1.42	50.84
CPGNN-Cheby-1	63.13±5.72	37.03±1.23	53.90±2.61	51.35
CPGNN-Cheby-2	65.97±8.78	27.92±1.53	56.93±2.03	50.27
H_2GCN	71.39±2.57	29.50±0.77	48.12±1.96	49.67
GraphSAGE	67.36±3.05	34.35±1.09	45.45±1.97	49.05
GCN-Cheby	58.96±3.04	26.52±0.92	36.66±1.84	40.71
MixHop	62.15±2.48	36.42±3.43	46.84±3.47	48.47
GCN	55.90±2.05	33.31±0.89	52.00±2.30	47.07
GAT	55.83±0.67	31.20±2.57	50.54±1.97	45.86
MLP	64.65±3.06	25.50±0.87	37.36±2.05	42.50

Table 3: Accuracy on homophily graphs *with* features.

Hom. ratio h	Citeseer 0.74	Pubmed 0.8	Cora 0.81	Mean Acc
CPGNN-MLP-1	71.30±1.11	86.40±0.36	77.40±1.10	78.37
CPGNN-MLP-2	71.48±1.85	85.31±0.70	81.24±1.26	79.34
CPGNN-Cheby-1	72.04±0.53	86.68±0.20	83.64±1.31	80.79
CPGNN-Cheby-2	72.06±0.51	86.66±0.24	81.62±0.97	80.11
H_2GCN	71.76±0.64	85.93±0.40	83.43±0.95	80.37
GraphSAGE	71.74±0.66	85.66±0.53	81.60±1.16	79.67
GCN-Cheby	72.04±0.58	86.43±0.31	83.29±1.20	80.58
MixHop	73.23±0.60	85.12±0.29	85.34±1.23	81.23
GCN	72.27±0.52	86.42±0.27	83.56±1.21	80.75
GAT	72.63±0.87	84.48±0.22	79.57±2.12	78.89
MLP	66.52±0.99	84.70±0.33	64.81±1.20	72.01

Table 4: Accuracy on heterophily graphs *without* features.

Hom. ratio h	Texas 0.11	Squirrel 0.22	Chameleon 0.23	Mean Acc
CPGNN-MLP-1	64.05±7.65	55.19±1.88	68.38±3.48	62.54
CPGNN-MLP-2	65.14±9.99	36.37±2.08	70.18±2.64	57.23
CPGNN-Cheby-1	63.78±7.67	54.76±2.01	67.19±2.18	61.91
CPGNN-Cheby-2	70.27±8.26	26.42±1.20	68.25±1.57	54.98
H_2GCN	68.38±6.98	50.91±1.71	62.41±2.14	60.57
GraphSAGE	67.03±4.90	36.90±2.36	58.53±2.20	54.15
GCN-Cheby	50.00±8.08	12.62±0.73	14.93±1.53	25.85
MixHop	57.57±5.56	33.54±2.08	50.15±2.78	47.08
GCN	51.08±7.48	43.78±1.39	62.04±2.17	52.30
GAT	57.03±4.31	42.46±2.08	60.31±2.61	53.26
MLP	44.86±9.29	19.77±0.80	20.57±2.29	28.40

Table 5: Accuracy on homophily graphs *without* features.

Hom. ratio h	Citeseer 0.74	Pubmed 0.8	Cora 0.81	Mean Acc
CPGNN-MLP-1	65.70±2.96	81.98±0.36	81.97±1.24	76.55
CPGNN-MLP-2	67.66±2.29	82.33±0.39	82.37±1.70	77.46
CPGNN-Cheby-1	67.93±2.86	82.44±0.58	83.76±1.81	78.04
CPGNN-Cheby-2	67.39±2.69	82.27±0.54	83.02±1.29	77.56
H_2GCN	68.37±2.93	82.97±0.37	83.22±1.56	78.19
GraphSAGE	66.71±3.27	77.86±3.84	81.77±2.00	75.45
GCN-Cheby	67.56±3.24	79.14±0.38	83.66±1.02	76.79
MixHop	68.38±3.06	82.72±0.75	84.73±1.80	78.61
GCN	67.14±3.15	82.28±0.50	83.34±1.38	77.59
GAT	68.64±3.27	81.92±0.33	81.79±2.21	77.45
MLP	19.78±1.35	39.58±0.69	21.61±1.92	26.99

Heterophily. We report results on graphs with strong heterophily under the featureless settings in Table 4. We observe that the best performing methods for each dataset are all CPGNN variants. From the mean accuracy perspective, all CPGNN variants outperform all baselines except H_2GCN, which is also proposed to handle heterophily, in the overall performance; CPGNN-MLP-1 has the best *overall* performance, followed by CPGNN-Cheby-1. It is also worth noting that the performance of GCN-Cheby and MLP, upon which our prior belief estimator is based on, are significantly worse than other methods. This demonstrates the effectiveness of incorporating the class compatibility matrix $\mathbf{H}$ in GNN models and learning it in an end-to-end fashion.

Homophily. We report the results in Table 5. The featureless setting for graphs with strong homophily is a fundamentally easier task compared to graphs with strong heterophily, especially for methods with implicit homophily assumptions, as they tend to yield highly similar prediction within the proximity of each node. Despite this, the CPGNN variants still perform comparably to the state-of-the-art methods.

Summary. Under the featureless settings, the above results show that CPGNN variants achieve state-of-the-art performance in heterophily settings, while achieving comparable performance in the homophily settings. Considering both the heterophily and homophily settings, CPGNN-Cheby-1 is again the best method overall.

4.4 Ablation Study

To evaluate the effectiveness of our model design, we conduct an ablation study by examining variants of CPGNN-MLP-1 with one design element removed at a time. Fig. 4 presents the results for the ablation study, with more detailed results presents in Table A.2 in Appendix. We also discussed the effectiveness of co-training and pretraining in Appendix §D.

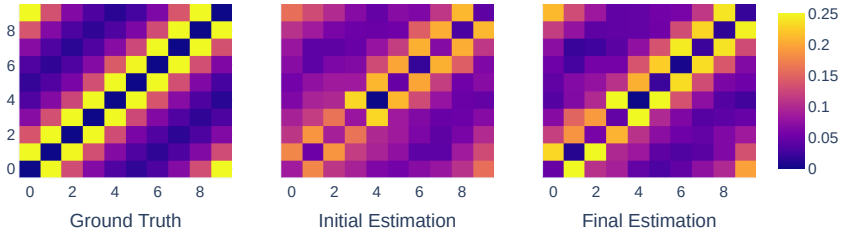
Initialization and Regularization of $\bar{\mathbf{H}}$. Here we study 2 variants of CPGNN-MLP-1: (1) No $\bar{\mathbf{H}}$ initialization, when $\bar{\mathbf{H}}$ is initialized using glorot initialization (similar to other GNN formulations) instead of our initialization process described in § 2.3. (2) No $\bar{\mathbf{H}}$ regularization, where we remove the regularization term $\Phi(\bar{\mathbf{H}})$ as defined in Eq. (12) from the overall loss function (Eq. (13)). In Fig. 4a, we see that replacing the initializer can lead to up to 30% performance drop for the model, while removing the regularization term can cause up to 6% decrease in performance. These results support our claim that initializing $\bar{\mathbf{H}}$ using pretrained prior beliefs and known labels in the training set and regularizing the $\bar{\mathbf{H}}$ around 0 lead to better overall performance.

End-to-end Training of $\bar{\mathbf{H}}$. To demonstrate the performance gain through end-to-end training of CPGNN after the initialization of $\bar{\mathbf{H}}$, we compare the final performance of CPGNN-MLP-1 with the performance after $\bar{\mathbf{H}}$ is initialized; Fig. 4b shows the results. From the results, we see that the end-to-end training process of CPGNN has contributed up to 21% performance gain. We believe such performance gain is due to a more accurate $\bar{\mathbf{H}}$ learned through the training process, as demonstrated in the next subsection.

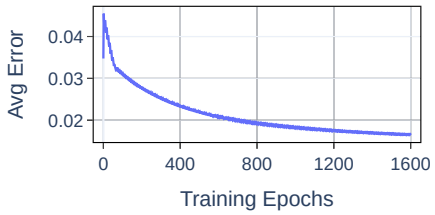
4.5 compatibility matrix Estimation

As described in §2.4, we can obtain an estimation $\hat{\mathbf{H}}$ of the class compatiblity matrix $\mathbf{H} \in [0,1]^{|\mathcal{Y}|\times|\mathcal{Y}|}$ through the learned parameter $\bar{\mathbf{H}}$. To measure the accuracy of the estimation $\hat{\mathbf{H}}$, we calculate the average error of each element for the estimated $\hat{\mathbf{H}}$ as following: $\bar{\delta}_{\mathbf{H}} = \frac{|\hat{\mathbf{H}}-\mathbf{H}|}{|\mathcal{Y}|^2}$.

Fig. 3 shows an example of the obtained estimation $\hat{\mathbf{H}}$ on the synthetic benchmark `syn-products` with homophily

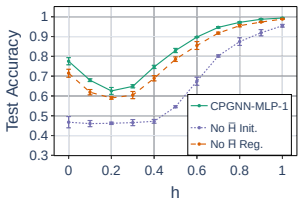


(a) Heterophily matrices $\mathbf{H}$ for empirical (ground truth), initial and final estimation.

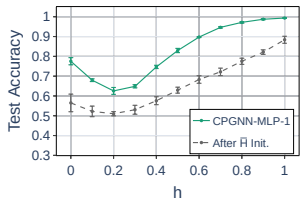


(b) Error of compatibility matrix estimation $\hat{\mathbf{H}}$ throughout training process.

of $\mathbf{H}$ for a $h = 0$ instance of `syn-products` dataset.



(a) Accuracy without initialization or regularization of $\bar{\mathbf{H}}$.



(b) Accuracy after end-to-end training vs. after initializing $\bar{\mathbf{H}}$.

Figure 4: Ablation Study: Mean accuracy as a function of h. (a): When replacing $\bar{\mathbf{H}}$ initialization with glorot or removing $\bar{\mathbf{H}}$ regularization, the performance of CPGNN drops significantly; (b): The significant increase in performance shows the effectiveness of the end-to-end training in our framework.

ratio $h = 0$ using heatmaps, along with the initial estimation derived following §2.3 which CPGNN optimizes upon, and the ground truth empirical compatibility matrix as defined in Def. 2. From the heatmap, we can visually observe the improvement of the final estimation upon the initial estimation. The curve of the estimation error with respect to the number of training epochs also shows that the estimation error decreases throughout the training process, supporting the observations through the heatmaps. These results illustrate the interpretability of parameters $\bar{\mathbf{H}}$, and effectiveness of our modeling of compatibility matrix.

5 Related Work

SSL before GNNs. The problem of semi-supervised learning (SSL) or *collective classification* (Sen et al. 2008; McDowell, Gupta, and Aha 2007; Rossi et al. 2012) can be solved with iterative methods (J. Neville 2000; Lu and Getoor 2003), graph-based regularization and probabilistic graphical models (London and Getoor 2014). Among these methods, our approach is related to belief propagation (BP) (Yedidia, Freeman, and Weiss 2003; Rossi et al. 2018), a message-passing approach where each node iteratively sends its neighboring nodes estimations of their beliefs based on its current belief, and updates its own belief based on the estimations received from its neighborhood. Koutra et al. (2011) and Gatterbauer et al. (2015) have proposed *linearized* versions which are faster to compute. However, these approaches require the class-compatibility matrix to be determined before the inference stage, and cannot support end-to-end training.

GNNs. In recent years, graph neural networks (GNNs) have become increasingly popular for graph-based semi-supervised node classification problems thanks to their ability to learn through end-to-end training. Defferrard, Bresson, and Vandergheynst (2016) proposed an early version of GNN by generalizing convolutional neural networks (CNNs) from regular grids (e.g., images) to irregular grids (e.g., graphs). Kipf and Welling (2017) introduced GCN, a popular GNN model which simplifies the previous work. Other GNN models that have gained wide attention include Planetoid (Yang, Cohen, and Salakhudinov 2016) and GraphSAGE (Hamilton, Ying, and Leskovec 2017). More recent works have looked into designs which strengthen the effectiveness of GNN to capture graph information: GAT (Veličković et al. 2018) and AGNN (Thekumparampil et al. 2018) introduced an edge-level attention mechanism; MixHop (Abu-El-Haija et al. 2019) and Geom-GCN (Pei et al. 2020) designed aggregation schemes which go beyond the immediate neighborhood of each node; the jumping knowledge network (Xu et al. 2018) leverages representations from intermediate layers; GAM (Stretcu et al. 2019) and GMNN (Qu, Bengio, and Tang 2019) use a separate model to capture the agreement or joint distribution of labels in the graph. To capture more graph information, recent works trained very deep networks with 100+ layers (Li et al. 2019, 2020; Rong et al. 2020).

Although many of these GNN methods work well when the data exhibits strong homophily, none of these methods (except Geom-GCN) was proposed to address the challenging and largely overlooked setting of heterophily, and many of them perform poorly in this setting. Recently, Zhu et al. (2020) discussed effective designs which improve the representation power of GNNs under heterophily through theoretical and empirical analysis. Going beyond these designs that prior GNN works have leveraged, we propose a new GNN framework that elegantly combines the powerful notion of compatibility matrix $\mathbf{H}$ from belief propagation with end-to-end training.

6 Conclusion

We propose CPGNN, an approach that models an interpretable class compatibility matrix into the GNN framework, and conduct extensive empirical analysis under more realistic settings with fewer training samples and a featureless setup. Through theoretical and empirical analysis, we have shown that the proposed model overcomes the limitations of existing GNN models, especially in the complex settings of heterophily graphs without contextual features.

Acknowledgments

We would like to thank Mark Heimann, Yujun Yan and Leman Akoglu for engaging discussions during the early stage of this work, and the reviewers for their constructive feedback. This material is based upon work supported by the National Science Foundation under CAREER Grant No. IIS 1845491, Army Young Investigator Award No. W911NF1810397, an Adobe Digital Experience research faculty award, an Amazon faculty award, a Google faculty award, and AWS Cloud Credits for Research. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Quadro P6000 GPU used for this research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding parties.

References

Abu-El-Haija, S.; Perozzi, B.; Kapoor, A.; Harutyunyan, H.; Alipourfard, N.; Lerman, K.; Steeg, G. V.; and Galstyan, A. 2019. MixHop: Higher-Order Graph Convolution Architectures via Sparsified Neighborhood Mixing. In *International Conference on Machine Learning (ICML)*.

Ahmed, N. K.; Rossi, R.; Lee, J. B.; Willke, T. L.; Zhou, R.; Kong, X.; and Eldardiry, H. 2018. Learning Role-based Graph Embeddings. In *IJCAI*.

Barabasi, A. L.; and Albert, R. 1999. Emergence of scaling in random networks. *Science* 286(5439): 509–512.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, 3844–3852.

Dou, Y.; Liu, Z.; Sun, L.; Deng, Y.; Peng, H.; and Yu, P. S. 2020. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 315–324.

Fout, A.; Byrd, J.; Shariat, B.; and Ben-Hur, A. 2017. Protein interface prediction using graph convolutional networks. In *Advances in neural information processing systems*, 6530–6539.

Gatterbauer, W.; Günnemann, S.; Koutra, D.; and Faloutsos, C. 2015. Linearized and single-pass belief propagation. *Proceedings of the VLDB Endowment* 8(5): 581–592.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687* .

J. Neville, D. J. 2000. Iterative classification in relational data. In *In Proc. AAAI*, 13–20. AAAI Press.

Karimi, F.; Génois, M.; Wagner, C.; Singer, P.; and Strohmaier, M. 2017. Visibility of minorities in social networks. *arXiv preprint arXiv:1702.00150* .

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

Koutra, D.; Ke, T.-Y.; Kang, U.; Chau, D. H. P.; Pao, H.-K. K.; and Faloutsos, C. 2011. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 245–260. Springer.

Li, G.; Müller, M.; Thabet, A.; and Ghanem, B. 2019. DeepGCNs: Can GCNs Go as Deep as CNNs? In *The IEEE International Conference on Computer Vision (ICCV)*.

Li, G.; Xiong, C.; Thabet, A.; and Ghanem, B. 2020. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739* .

London, B.; and Getoor, L. 2014. Collective Classification of Network Data. *Data Classification: Algorithms and Applications* 399.

Lu, Q.; and Getoor, L. 2003. Link-Based Classification. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning (ICML)*, 496–503. AAAI Press.

McDowell, L. K.; Gupta, K. M.; and Aha, D. W. 2007. Cautious inference in collective classification. In *AAAI*, volume 7, 596–601.

McPherson, M.; Smith-Lovin, L.; and Cook, J. M. 2001. Birds of a feather: Homophily in social networks. *Annual review of sociology* 27(1): 415–444.

Namata, G.; London, B.; Getoor, L.; and Huang, B. 2012. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8.

Pandit, S.; Chau, D. H.; Wang, S.; and Faloutsos, C. 2007. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *Proceedings of the 16th international conference on World Wide Web*, 201–210.

Pei, H.; Wei, B.; Chang, K. C.-C.; Lei, Y.; and Yang, B. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations*.

Qu, M.; Bengio, Y.; and Tang, J. 2019. GMNN: Graph Markov Neural Networks. In *International Conference on Machine Learning*, 5241–5250.

Rong, Y.; Huang, W.; Xu, T.; and Huang, J. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*. URL https://openreview.net/forum?id=Hkx1qkrKPr.

Rossi, R. A.; and Ahmed, N. K. 2015. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 4292–4293. URL http://networkrepository.com.

Rossi, R. A.; Jin, D.; Kim, S.; Ahmed, N. K.; Koutra, D.; and Lee, J. B. 2020. On Proximity and Structural Role-based Embeddings in Networks: Misconceptions, Techniques, and Applications. In *Transactions on Knowledge Discovery from Data (TKDD)*, 36.

Rossi, R. A.; McDowell, L. K.; Aha, D. W.; and Neville, J. 2012. Transforming Graph Data for Statistical Relational Learning. *Journal of Artificial Intelligence Research (JAIR)* 45: 363–441.

Rossi, R. A.; Zhou, R.; Ahmed, N. K.; and Eldardiry, H. 2018. Relational Similarity Machines (RSM): A Similarity-based Learning Framework for Graphs. In *IEEE BigData*, 10.

Rozemberczki, B.; Allen, C.; and Sarkar, R. 2019. Multi-scale attributed node embedding. *arXiv preprint arXiv:1909.13021* .

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20(1): 61–80.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine* 29(3): 93–93.

Sinkhorn, R.; and Knopp, P. 1967. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics* 21(2): 343–348.

Stretcu, O.; Viswanathan, K.; Movshovitz-Attias, D.; Platanios, E.; Ravi, S.; and Tomkins, A. 2019. Graph Agreement Models for Semi-Supervised Learning. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 8713–8723.

Thekumparampil, K. K.; Wang, C.; Oh, S.; and Li, L.-J. 2018. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735* .

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning*, 6861–6871.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.; and Jegelka, S. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, volume 80, 5449–5458. PMLR.

Yan, Y.; Zhu, J.; Duda, M.; Solarz, E.; Sripada, C.; and Koutra, D. 2019. Groupinn: Grouping-based interpretable neural network for classification of limited, noisy brain data. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 772–782.

Yang, Z.; Cohen, W.; and Salakhudinov, R. 2016. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, 40–48.

Yedidia, J. S.; Freeman, W. T.; and Weiss, Y. 2003. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium* 8: 236–239.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 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.

Zhu, J.; Yan, Y.; Zhao, L.; Heimann, M.; Akoglu, L.; and Koutra, D. 2020. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. *Advances in Neural Information Processing Systems* 33.

Zitnik, M.; Agrawal, M.; and Leskovec, J. 2018. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics* 34(13): i457–i466.

Appendix

A Synthetic Graph Generation

We generate synthetic graphs in a way improved upon Abu-El-Haija et al. (2019) by following a modified preferential attachment process (Barabasi and Albert 1999), which allows us to control the compatibility matrix $\mathbf{H}$ in generated graph while keeping a power law degree distribution. We detail the algorithm of synthetic graph generation in Algorithm 1.

For the synthetic graph `syn-products` used in our experiments, we use `ogbn-products` (Hu et al. 2020) as the reference graph $\mathcal{G}_r$, with parameters $C = 10$, $n_0 = 70$, $m = 6$ and the total number of nodes as 10000; all 10 classes share the same size of 1000. For the compatibility matrix, we set the diagonal elements of $\mathbf{H}$ to be the same, which we denote as h, and we follow the approach in Abu-El-Haija et al. (2019) to set the off-diagonal elements.

B More Experimental Setups

Baseline Implementations. We use the official implementation released by the authors on GitHub for all baselines besides MLP.

- **GCN & GCN-Cheby** (Kipf and Welling 2017): https://github.com/tkipf/gcn
- **GraphSAGE** (Hamilton, Ying, and Leskovec 2017): https://github.com/williamleif/graphsage-simple (PyTorch implementation)
- **MixHop** (Abu-El-Haija et al. 2019): https://github.com/samihaija/mixhop
- **GAT** (Veličković et al. 2018): https://github.com/PetarV-/GAT
- **H_2GCN** (Zhu et al. 2020): https://github.com/GemsLab/H2GCN

Hardware and Software Specifications. We run all experiments on a workstation which features an AMD Ryzen 9 3900X CPU with 12 cores, 64GB RAM, a Nvidia Quadro P6000 GPU with 24GB GPU Memory and a Ubuntu 20.04.1 LTS operating system. We implement CPGNN using TensorFlow 2.2 with GPU support.

C Hyperparameter Tuning

Below we list the hyperparameters tested on each benchmark per model on real-world graphs. As the hyperparameters defined by each baseline model differ significantly, we list the combinations of non-default command line arguments we tested, without explaining them in detail. We refer the interested reader to the corresponding original implementations for further details on the arguments, including their definitions. When multiple hyperparameters are listed, the results reported for each benchmark are based on the hyperparameters which yield the best validation accuracy in average.

To ensure a fair evaluation of the performance improvement brought by CPGNN, the MLP and GCN-Cheby prior belief estimator in CPGNN-MLP and CPGNN-Cheby share the same network architecture (including numbers and sizes of hidden layers) as our MLP and GCN-Cheby baselines.

Algorithm 1: Synthetic Graph Generation

Input: $C \in \mathbb{N}$: Number of classes in generated graph;
$\mathbf{N} \in \mathbb{N}^C$: target size of each class;
$n_0 \in \mathbb{N}$: Number of nodes for the initial bootstrapping graph, which should be much smaller than the total number of nodes;
$m \in \mathbb{N}$: Number of edges added with each new node;
$\mathbf{H} \in [0,1]^{C \times C}$: Target compatibility matrix for generated graph;
$\mathcal{G}_r = (\mathcal{V}_r, \mathcal{E}_r)$: Reference graph with node set $\mathcal{V}_r$ and edge set $\mathcal{E}_r$;
$\mathbf{y}_r$: mapping from each node $v \in \mathcal{V}_r$ to its class label $\mathbf{y}_r[v]$ in the reference graph $\mathcal{G}_r$;
$\mathbf{X}_r$: mapping from each node $v \in \mathcal{V}_r$ to its node feature vector $\mathbf{X}_r[v]$ in the reference graph $\mathcal{G}_r$;

Output: Generated synthetic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathbf{y} : \mathcal{V} \to \mathcal{Y}$ as mapping from each node $v \in \mathcal{V}$ to its class label $\mathbf{y}[v]$, and $\mathbf{X} : \mathcal{V} \to \mathbb{R}^F$ as mapping from $v \in \mathcal{V}$ to its node feature vector $\mathbf{X}[v]$.

begin

- Initialize class label set $\mathcal{Y} \leftarrow \{0, \ldots, C-1\}$, node set $\mathcal{V} \leftarrow \phi$, edge set $\mathcal{E} \leftarrow \phi$;
- Calculate the target number of nodes n in generated graph by summing up all elements in $\mathbf{N}$;
- Generate node label vector $\mathbf{y}$, such that class label $y \in \mathcal{Y}$ appears exactly $\mathbf{N}[y]$ times in $\mathbf{y}$, and shuffle $\mathbf{y}$ randomly after generation;
- **for** $v \in \{0, 1, \ldots, n_0 - 1\}$ **do**
  - Add new node v with class label $\mathbf{y}[v]$ into the set of nodes $\mathcal{V}$;
  - If $v \neq 0$, add new edge $(v-1, v)$ into the set of edges $\mathcal{E}$;
- **for** $v \in \{n_0, n_0+1, \ldots, n-1\}$ **do**
  - Initialize weight vector $\mathbf{w} \leftarrow \mathbf{0}$ and set $\mathcal{T} \leftarrow \phi$;
  - **for** $u \in \mathcal{V}$ **do**
    - $\mathbf{w}[u] \leftarrow \mathbf{H}[\mathbf{y}[v], \mathbf{y}[u]] \times \mathbf{d}[u]$, where $\mathbf{d}[u]$ is the current degree of node u;
  - Normalize vector $\mathbf{w}$ such that $\|\mathbf{W}\|_1 = 1$;
  - Randomly sample m nodes *without replacement* from $\mathcal{V}$ with probabilities weighted by $\mathbf{w}$, and add the sampled nodes into set $\mathcal{T}$;
  - Add new node v with class label $\mathbf{y}[v]$ into the set of nodes $\mathcal{V}$;
  - **for** $t \in \mathcal{T}$ **do**
    - Add new edge (t, v) into the set of edges $\mathcal{E}$;
- Find an valid injection $\Gamma : \mathcal{V} \to \mathcal{V}_r$ such that $\forall u, v \in \mathcal{V}, \Gamma(u) = \Gamma(v) \Rightarrow u = v$ and $\mathbf{y}[u] = \mathbf{y}[v] \Leftrightarrow \mathbf{y}_r[\Gamma(u)] = \mathbf{y}_r[\Gamma(v)]$;
- **for** $v \in \mathcal{V}$ **do**
  - $\mathbf{X}[v] \leftarrow \mathbf{X}_r[\Gamma(v)]$;

- **GraphSAGE** (Hamilton, Ying, and Leskovec 2017):
  - `hid_units`: 64
  - `lr`: $a \in \{0.1, 0.7\}$
  - `epochs`: 500
- **GCN-Cheby** (Kipf and Welling 2017):
  - `hidden1`: 64

Table A.1: Node classification *with* features on synthetic graph (§4.2, Fig. 2): mean classification accuracy per method and homophily ratio h on `syn-products`.

Methods	Homophily ratio h										
	0.00	**0.10**	**0.20**	**0.30**	**0.40**	**0.50**	**0.60**	**0.70**	**0.80**	**0.90**	**1.00**
CPGNN-MLP-1	76.98±2.01	67.88±0.82	63.21±1.57	65.23±0.85	74.86±0.60	83.25±1.30	89.83±0.16	94.67±0.27	97.13±0.40	98.77±0.27	99.43±0.07
CPGNN-MLP-2	71.40±1.90	65.28±0.80	61.76±1.18	66.17±0.40	74.67±0.61	82.63±0.56	88.68±0.21	93.33±0.63	96.81±0.14	98.94±0.18	99.90±0.12
CPGNN-Cheby-1	76.37±0.33	66.38±0.39	63.01±0.68	67.39±0.94	77.57±0.39	86.86±1.40	94.44±0.24	98.16±0.16	99.60±0.06	99.89±0.13	100.00±0.00
CPGNN-Cheby-2	72.80±1.72	65.26±0.87	62.05±1.17	64.40±0.86	73.12±0.62	80.90±1.09	87.43±0.22	91.85±0.62	96.66±0.47	99.67±0.09	100.00±0.00
GraphSAGE	59.15±0.73	53.53±0.77	54.54±0.66	56.08±0.45	61.17±1.19	68.98±1.44	78.14±1.10	86.55±0.30	92.71±1.35	96.69±0.38	99.12±0.11
GCN-Cheby	68.65±1.30	60.51±1.64	61.98±0.68	66.20±1.24	74.43±1.40	83.60±0.77	92.28±0.47	97.11±0.18	99.25±0.09	99.81±0.06	99.80±0.18
MixHop	10.66±1.73	11.29±0.61	12.40±1.86	11.87±2.38	13.91±3.21	19.72±1.06	20.33±1.11	21.72±2.07	21.88±3.04	21.32±1.91	22.60±2.14
GCN	44.72±0.51	41.87±1.37	46.49±0.50	55.63±0.88	69.33±0.80	81.21±0.97	90.65±0.35	96.01±0.10	98.80±0.14	99.64±0.01	99.99±0.01
GAT	19.59±5.96	21.74±2.06	25.67±1.77	30.34±2.90	39.42±7.60	50.62±5.45	64.68±5.01	88.01±3.71	98.01±0.65	99.06±0.80	99.94±0.02
MLP	47.46±2.66	47.15±1.47	47.55±0.90	47.35±2.02	47.07±0.94	48.25±0.76	47.37±1.41	47.38±1.64	46.87±0.65	46.94±0.86	48.12±1.63

Table A.2: Ablation study of CPGNN-MLP-1 (§4.4 and App. §D): mean classification accuracy per method and homophily ratio h on `syn-products`.

Variants	Homophily ratio h										
	0.00	**0.10**	**0.20**	**0.30**	**0.40**	**0.50**	**0.60**	**0.70**	**0.80**	**0.90**	**1.00**
CPGNN-MLP-1	77.52±1.82	67.95±0.68	62.55±1.73	64.85±0.81	74.67±0.82	82.95±1.07	89.75±0.24	94.64±0.38	97.17±0.34	98.73±0.30	99.35±0.10
No $\bar{H}$ Init. (Fig. 4a)	46.74±2.86	46.03±1.56	46.20±0.61	46.53±1.55	47.20±0.94	54.54±0.50	67.47±2.06	80.18±0.53	87.44±1.93	91.89±1.59	95.47±0.68
No $\bar{H}$ Reg. (Fig. 4a)	71.49±2.01	61.88±1.37	58.98±0.64	60.43±1.82	68.91±1.35	78.55±1.14	85.47±1.99	91.76±0.49	95.48±0.59	97.31±0.03	98.82±0.19
After $\bar{H}$ Init. (Fig. 4b)	56.49±4.48	52.22±2.70	51.02±1.08	53.05±2.25	57.58±1.99	62.95±1.51	68.31±1.88	72.18±1.00	77.53±1.56	82.13±1.11	88.38±1.77
No Cotrain (Fig. 5)	75.63±1.33	65.85±0.79	61.96±1.84	64.52±1.57	73.67±0.64	8					
No Pretrain (Fig. 5)	75.67±2.65	65.45±0.47	60.23±0.68	64.15±0.97	73.59±0.79	8					

  - `weight_decay`: $a \in \{$`1e-5, 5e-4`$\}$
  - `max_degree`: 2
  - `early_stopping`: 40
- **Mixhop** (Abu-El-Haija et al. 2019):
  - `adj_pows`: 0, 1, 2
  - `hidden_dims_csv`: 64
- **GCN** (Kipf and Welling 2017):
  - `hidden1`: 64
  - `early_stopping`: $a \in \{40, 100, 200\}$
  - `epochs`: 2000
- **GAT** (Veličković et al. 2018):
  - `hid_units`: 8
  - `n_heads`: 8
- **H_2GCN** (Zhu et al. 2020):
  - `network_setup`: `M64-T1-G-V-T2-G-V-C1-C2-D-MO` or `M64-R-T1-G-V-T2-G-V-C1-C2-D-MO` (H_2GCN-2 with or without `ReLU`)
  - `dropout`: 0 or 0.5
  - `l2_regularize_weight`: `1e-5`
- **MLP**:
  - Dimension of Feature Embedding: 64
  - Number of hidden layer: 1
  - Non-linearity Function: `ReLU`
  - Dropout Rate: 0

D Detailed Results

Node Classification with Contextual Features. Table A.1 provides the detailed results on `syn-products`, as illustrated in Fig. 2 in §4.2.

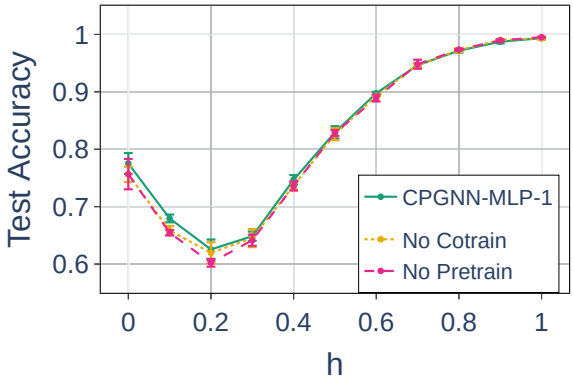


Figure 5: Ablation Study for co-training and pretraining: Mean accuracy as a function of h. Co-training and pretraining contribute up to 2% performance gain (cf. App. §D).

Ablation Study. Table A.2 presents more detailed results for the ablation study (cf. §4.4), which complements Fig. 4. In addition, we also conduct an ablation study to examine the effectiveness of co-training and pretraining. We test a variant where co-training is removed by setting $\eta = 0$ for the co-training loss term $\eta\mathcal{L}_p(\mathbf{\Theta}_p)$ in Eq. (13). We also test another variant where we skip the pretraining for prior belief estimator. We refer to these 2 variants as "No Cotrain" and "No Pretrain" respectively. Fig. 5 and Table A.2 reveal that, though the differences in performance are small, the adoption of co-training and pretraining has led to up to 2% increase for the performance in heterophily settings.