Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Towards adaptive graph neural networks via solving prior-data conflicts*

Xugang WU†, Huijun WU, Ruibo WANG†, Xu ZHOU, Kai LU†‡

College of Computer, National University of Defense Technology, Changsha 410073, China

†E-mail: wuxugang13@nudt.edu.cn; ruibo@nudt.edu.cn; lukainudt@163.com

Received Mar. 20, 2023; Revision accepted June 8, 2023; Crosschecked Feb. 4, 2024

Abstract: Graph neural networks (GNNs) have achieved remarkable performance in a variety of graph-related tasks. Recent evidence in the GNN community shows that such good performance can be attributed to the homophily prior; i.e., connected nodes tend to have similar features and labels. However, in heterophilic settings where the features of connected nodes may vary significantly, GNN models exhibit notable performance deterioration. In this work, we formulate this problem as prior-data conflict and propose a model called the mixture-prior graph neural network (MPGNN). First, to address the mismatch of homophily prior on heterophilic graphs, we introduce the non-informative prior, which makes no assumptions about the relationship between connected nodes and learns such relationship from the data. Second, to avoid performance degradation on homophilic graphs, we implement a soft switch to balance the effects of homophily prior and non-informative prior by learnable weights. We evaluate the performance of MPGNN on both synthetic and real-world graphs. Results show that MPGNN can effectively capture the relationship between connected nodes, while the soft switch helps select a suitable prior according to the graph characteristics. With these two designs, MPGNN outperforms state-of-the-art methods on heterophilic graphs without sacrificing performance on homophilic graphs.

Key words: Graph neural networks; Heterophily; Prior-data conflict
https://doi.org/10.1631/FITEE.2300194 **CLC number:** TP18

1 Introduction

Graphs are used to represent structured data in a variety of real-world scenarios, including social networks, biological networks, and citation networks (Tang et al., 2009; Wang et al., 2016; Ma et al., 2019; Zhang ZW et al., 2022). In recent years, graph neural networks (GNNs) (Hamilton et al., 2017; Kipf and Welling, 2017; Veličković et al., 2018) have demonstrated remarkable success in various graph-related tasks, such as node classification, link prediction, and graph classification (Grover and Leskovec, 2016; Ying et al., 2018; Zhang MH and Chen, 2018; Chiang et al., 2019; Wu ZH et al., 2020).

There is a key prior in the design of most GNN models, the homophily prior (McPherson et al., 2001; Ciotti et al., 2016), which posits that nodes with similar features or the same class labels are more likely to be connected in the graph. For example, papers from the same area tend to cite each other in citation networks. The homophily prior provides a strong regularization that prevents GNN models from overfitting when the amount of training data is limited. Consequently, incorporating such an assumption into GNN models boosts their performance on homophilic graphs significantly.

‡ Corresponding author

* Project supported by the National University of Defense Technology Foundation (Nos. ZK20-09 and ZK21-17), the Natural Science Foundation of Hunan Province, China (No. 2021JJ40692), and the National Key R&D Program of China (No. 2021YFB0300101)

ORCID: Xugang WU, https://orcid.org/0000-0003-4715-6785; Ruibo WANG, https://orcid.org/0000-0001-7952-3784; Kai LU, https://orcid.org/0000-0002-6378-7002

It should be noted, however, that graphs in the real world can be heterophilic (Pei et al., 2020). In these heterophilic settings, connected nodes typically share dissimilar features and labels. In transaction networks, scammers prefer to link to normal customers to conceal their activities. In dating graphs, people tend to connect with someone of the opposite gender. As demonstrated in Zhu et al. (2020), the mismatch between the prior knowledge and the data characteristics leads to suboptimal GNN performance in heterophilic settings. In some cases, even a multilayer perceptron (MLP) model (which ignores the graph structure and makes predictions exclusively based on the features of the nodes) could outperform most existing GNNs.

In this paper, we focus on solving such prior-data conflicts in GNNs. Because the conflict stems from the inconsistencies between the homophily prior and the uncertain graph characteristics, our first attempt is to remove the homophily prior in the model. Therefore, instead of using the homophily prior, which assumes strong similarity between connected nodes, we introduce a non-informative prior to capture the relationship between connected nodes. Specifically, we formulate the relationship as a latent variable and incorporate it into the networks. Starting with a non-informative prior, we apply the expectation-maximization (EM) algorithm to learn the model parameters and the inter-node relationship simultaneously. This design enables our method to discover the arbitrary relationship between connected nodes beyond the assumption of the homophily prior.

However, adopting the above non-informative prior alone results in poor performance on homophilic graphs. Due to the non-informative prior, no assumptions are made about the relationships between connected nodes, which can result in a broader posterior distribution and less precise estimates of model parameters. In contrast, an informative prior can narrow the posterior distribution and provide more precise estimates, especially when the sample size is small. Therefore, instead of simply discarding the homophily prior, we design a model with mixed priors. Specifically, we combine two priors in the model: (1) a homophily prior, which assumes that connected nodes in the graph are similar to one another; (2) a non-informative prior, which makes no assumptions about the relationship between connected nodes and learns the relationship from the data using the EM algorithm. The challenge is, then, how to determine the weight for each prior based on the characteristics of the graphs. Particularly, the model should put most of its weight on the homophily prior when tackling a homophilic graph, and vice versa. A straightforward way is to check which prior can better fit the training data. However, because most current GNNs are over-parameterized, the models can easily "memorize" all the training data even with a false prior (Arpit et al., 2017; Zhang CY et al., 2017; Feldman, 2020). To overcome this problem, we design a soft switch module. The intuition behind the soft switch is that models with proper priors will fit the training data faster in the early stages of training. Therefore, instead of training the model until it memorizes all the training data, the soft switch monitors which prior fits the training data better in the warm-up stage and then puts more weight on the suitable prior. Combining all these designs, we propose the mixture-prior GNN (MPGNN).

The contributions of this paper can be summarized as follows:

1. To address the prior-data conflicts of GNNs in heterophilic settings, we incorporate a non-informative prior in the model by formulating the relationship between connected nodes as a learnable transition matrix, thus enabling the inter-node relationship and the model parameters to be optimized simultaneously through an EM algorithm.

2. We adopt a scheme of mixture priors and implement a soft switch module to adjust the weight of the priors according to the characteristics of the graphs. Such a design allows the model to converge quickly to the proper prior without requiring much training data.

3. Combining these two designs, we propose the MPGNN model and evaluate it on both synthetic and real-world benchmarks. Results show that: (1) MPGNN can effectively capture the relationship between connected nodes and the soft switch can choose the suitable prior adaptively; (2) MPGNN achieves state-of-the-art performance on a range of heterophilic graphs without sacrificing its performance on homophilic graphs.

2 Related works

2.1 Graph neural networks

GNNs normally adopt a message-passing mechanism, where nodes in the graphs aggregate messages from their neighbor and are updated based on the aggregated messages. Generally, the message passing in layer l can be formulated as

$$m_v^{(l)} = \text{AGGREGATE}\left(\left\{h_u^{(l-1)} : u \in \mathcal{N}(v)\right\}\right), \quad (1)$$

$$h_v^{(l)} = \text{UPDATE}\left(h_v^{(l-1)}, m_v^{(l)}\right), \quad (2)$$

where $h_u^{(l)}$ denotes the representation of node u in layer l, and $\mathcal{N}(v)$ denotes the neighbors of node v. Different choices of the AGGREGATE and UPDATE functions yield a variety of GNNs. Graph convolutional network (GCN) (Kipf and Welling, 2017) aggregates the message from the neighbors and averages the aggregated and ego messages. Graphsage (Hamilton et al., 2017) uses more aggregation methods, such as sum and long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997). Graph attention network (GAT) (Veličković et al., 2018) calculates the attention score of each edge and assigns it as the edge weight during aggregation. Jumping knowledge network (JKNet) (Xu et al., 2018) concatenates the node representations in each layer to capture information in different neighboring hops. Adaptive diffusion convolution graph neural net (ADC-GNN) (Zhao et al., 2021) uses the adaptive diffusion convolution layer, which automatically learns the optimal neighborhood size from the data. There are some other works that attempt to separate the feature transformation step with the propagation step. Approximated personalized propagation of neural predictions (APPNP) (Klicpera et al., 2019) propagates the information of each node via personalized PageRank (Jeh and Widom, 2003). C&S (Huang et al., 2021) adopts an error correlation step and a prediction correlation step to enhance the propagation performance. Propagation then training adaptively (PTA) (Dong et al., 2021) uses the pseudo labels from the label propagation and assigns an adaptive weight to each pseudo label. From the spectral perspective, most GNNs can be regarded as low-pass filters (Wu F et al., 2019). As a result, these models implicitly assume that high-frequency components are noise and should be ignored. However, in heterophilic settings, it is possible for the high-frequency components to carry meaningful information, which hinders the generalization of traditional GNNs on heterophilic graphs.

2.2 Graph neural networks with heterophily

Recent advances in GNNs attempt to improve the performance of GNNs in heterophilic settings. Geom-GCN (Pei et al., 2020) computes the graph structure based on the unsupervised node embeddings and aggregates messages in a bi-level manner. Homophily and heterophily GCN (H_2GCN) (Zhu et al., 2020) makes targeted designs to boost the accuracy of GNNs under heterophily, such as the separation of ego and neighbor embeddings, higher-order neighborhoods, and multi-layer combination. Compatibility propagation GNN (CPGNN) (Zhu et al., 2021) introduces a compatibility matrix in GNN to model the level of heterophily or homophily in the graph. Generalized PageRank GNN (GPR-GNN) (Chien et al., 2021) first learns the embedding for each node and then propagates the embeddings through the generalized PageRank. Universal graph convolutional network (UGCN) (Jin et al., 2021) fuses the information from 1-hop, 2-hop, and K-nearest neighbors (KNN) networks via a discriminative aggregation. Graph pointer neural network (GPNN) (Yang et al., 2022) uses a pointer network to select the most relevant nodes, sorts them into a sequence, and learns from the sequence using LSTM (Hochreiter and Schmidhuber, 1997). BernNet (He et al., 2021) constructs an adaptive filter using an order-K approximation of the Bernstein polynomial and determines its spectral characteristics by configuring the coefficients of the Bernstein basis. Compared to the previous works, MPGNN is the first work to formulate the weakness of GNNs in heterophilic settings as a prior-data conflict problem and to solve the conflict via a mixture-prior scheme.

3 Preliminaries

3.1 Notations and problem setting

This work focuses on the semi-supervised node classification task in graphs. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{X})$ with n nodes, where $\mathcal{V}$ is the set of nodes $\{v_1, v_2, \cdots, v_n\}$ with $|\mathcal{V}| = n$, $\mathcal{E}$ is the set of edges,

and $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n)^{\mathrm{T}} \in \mathbb{R}^{n\times m}$ is the corresponding feature matrix. The feature of each node $v \in \mathcal{V}$ can be denoted as an m-dimensional vector $\boldsymbol{x}_v \in \mathbb{R}^m$. The adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ can be constructed by setting $A_{ij} = 1$ if $(v_i, v_j) \in \mathcal{E}$ and $A_{ij} = 0$ otherwise. $\boldsymbol{y} \in \mathcal{Y}^{|\mathcal{V}|}$ denotes the ground-truth label for each node. Our goal is to predict the classes of unlabeled nodes.

3.2 Measure of homophily in graphs

Currently, there are mainly two metrics for measuring the homophily ratio $\mathcal{H}$: average node homophily $\mathcal{H}_{\text{node}}$ (Pei et al., 2020) and average edge homophily $\mathcal{H}_{\text{edge}}$ (Zhu et al., 2020). $\mathcal{H}_{\text{node}}$ measures the label similarity among the neighbors of each node, while $\mathcal{H}_{\text{edge}}$ measures the label similarity between the members of each connected node pair.

Definition 1 (Average node homophily) The average node homophily is defined as

$$\mathcal{H}_{\text{node}}(\mathcal{G}) = \frac{1}{|\mathcal{V}|}\sum_{v\in\mathcal{V}}\frac{|\{y_v = y_u : u \in \mathcal{N}(v)\}|}{|\mathcal{N}(v)|}. \quad (3)$$

Definition 2 (Average edge homophily) The average edge homophily is defined as

$$\mathcal{H}_{\text{edge}}(\mathcal{G}) = \frac{1}{|\mathcal{E}|}\sum_{(u,v)\in\mathcal{E}}|y_v = y_u|. \quad (4)$$

These two metrics share the same trend in describing homophily. Their ranges are both [0, 1]. Higher values indicate stronger homophily. In this paper, we use the average node homophily ($\mathcal{H}_{\text{node}}$) as the measure of homophily and use $\mathcal{H}$ to denote it for simplicity.

3.3 Expectation-maximization algorithm

The EM algorithm is an approach for learning maximum likelihood parameters for models with unobserved latent variables (Dempster et al., 1977; McLachlan and Krishnan, 1997). Instead of optimizing the latent variable and the model parameters at the same time, the EM algorithm breaks the learning process into two steps: estimation step (E-step) and maximization step (M-step). First, the algorithm initializes both the latent variables and model parameters with a set of preset values. Then it turns into an EM loop until the learning converges. In the E-step, the algorithm estimates the latent variables using the observed data, whereas in the M-step, the algorithm updates the hypothesis with the latent variables learned in the expectation step. Commonly, the basic two steps of the EM algorithm, i.e., the E-step and M-step, more easily converge compared to the whole optimization problem, and therefore the EM algorithm can stabilize the training process, especially when the labeled data is limited.

3.4 Prior-data conflicts in GNNs

Incorporating prior knowledge into machine learning models is a well-known approach for improving the efficiency and accuracy of learning. When the prior knowledge aligns with the true distribution of the data, models designed based on such knowledge can achieve high performance. For instance, in the image processing field, the design of convolutional neural networks (CNNs) incorporates prior knowledge of the locality and translational invariance of image data (Krizhevsky et al., 2017). Similarly, in natural language processing, recurrent neural networks (RNNs) are designed to model the sequential correlation in text data (Hochreiter and Schmidhuber, 1997). In the graph domain, traditional GNNs incorporate the homophily prior, leading to high efficiency in homophilic settings (Kipf and Welling, 2017). However, attempting to learn all knowledge from data using a general model can lead to a low model learning efficiency and a high risk of overfitting noise information in the data, especially with limited data volume.

Nevertheless, when the prior knowledge does not align with the true distribution of data, it may result in prior-data conflicts. For example, using a model based on homophily prior to learn heterophilic data can significantly degrade the model's performance (Pei et al., 2020; Zhu et al., 2020). To address this issue, we propose a graph model that can adapt to both homophilic and heterophilic settings, without the need for manual selection of the prior assumptions. To achieve this, there are two important challenges: (1) How to design a model that can capture various node relationships and learn such relationships from the graphs, without relying on the homophily prior? (2) How to enable the model to adopt the suitable prior according to the characteristics of the graphs?

To address the first challenge, we design a non-informative propagating module that removes the homophily assumption from traditional graph

models and models the relationships between graph nodes as a learnable transition matrix, allowing the model to learn the true relationships between nodes from the data. Although the non-informative module eases the conflict and improves performance on heterophilic data, it may result in suboptimal performance on homophilic graphs, because it is less data-efficient compared to the homophily module in homophilic settings.

Therefore, it is crucial to select a suitable prior based on the graph's characteristics to achieve optimal performance, which is the second challenge of this work. A straightforward solution is to check which priors fit the training data better. However, this simple fit-check method has two drawbacks: (1) Over-parameterized models can easily "memorize" all the training data even with a false prior; (2) Manual comparison and selection of models are required, making it difficult to achieve end-to-end learning. To solve these problems, we design a soft switch to select a suitable prior based on the characteristics of the graphs. The soft switch is sensitive to the initial speed of model fitting, enabling it to quickly select a prior that aligns with the data in the early stages of training. Furthermore, such fitting speeds are reflected in the gradient corresponding to the modules, which enables us to integrate it into the training pipeline in an end-to-end manner, eliminating the need for multiple training runs to select a suitable prior.

4 Framework and designs

In this section, we present the overall MPGNN framework. We first introduce the overall MPGNN framework in Section 4.1. In Section 4.2, we demonstrate how we integrate a non-informative prior to formulate the relationship between connected nodes and how such relationships can be learned by using an EM algorithm. In Section 4.3, we introduce the mixture-prior scheme to avoid the performance degradation caused by discarding the homophily prior on the graphs with strong homophily. Finally, some details, including the initialization and the regularization of MPGNN, are demonstrated in Section 4.4.

4.1 Overall framework

The overall MPGNN framework is as illustrated in Fig. 1. The MPGNN model is built on a generalized PageRank GNN (GPR-GNN) (Chien et al., 2021). If we eliminate the non-informative propagation and soft switch components of the MPGNN model, we obtain the GPR-GNN model.

First, MPGNN maps the node features $\boldsymbol{X}$ into the node embeddings $\boldsymbol{H}^{(0)} \in \mathbb{R}^{|\mathcal{V}|\times C}$, where C

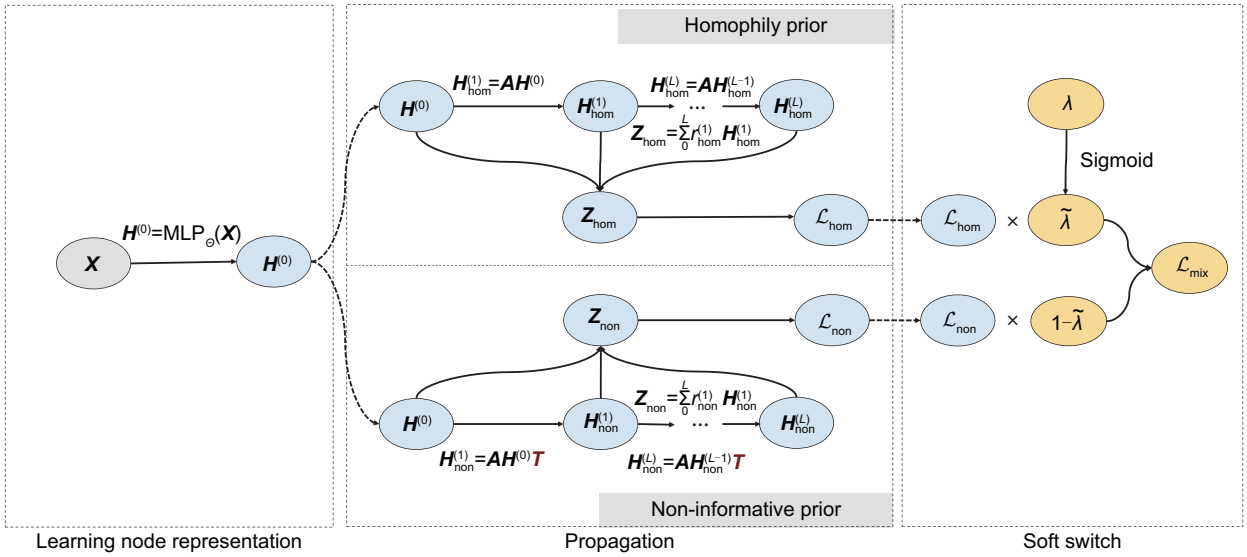

Fig. 1 The overall framework of MPGNN. First, we use an MLP to obtain the representation of each node. After that, the node representations are propagated with two priors separately: homophily prior and non-informative prior. Finally, a soft switch is used to combine their results according to the characteristics of the graph

denotes the number of classes:

$$\boldsymbol{H}^{(0)} = \mathrm{MLP}_{\theta}(\boldsymbol{X}). \tag{5}$$

After that, MPGNN propagates the node embeddings with two priors: homophily prior and non-informative prior. When propagating with the homophily prior, we assume that connected nodes tend to share similar features and labels. Therefore, we propagate the messages and calculate the next-layer embedding of each node by averaging the messages from its neighbors:

$$\boldsymbol{H}_{\mathrm{hom}}^{(l)} = \tilde{\boldsymbol{A}}\boldsymbol{H}_{\mathrm{hom}}^{(l-1)}. \tag{6}$$

Here, $\tilde{\boldsymbol{A}}$ denotes the normalized adjacency matrix. Let $\boldsymbol{D}$ denote the degree matrix of $\boldsymbol{A}+\boldsymbol{I}$. $\tilde{\boldsymbol{A}}$ is calculated as $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}(\boldsymbol{A}+\boldsymbol{I})\boldsymbol{D}^{-1/2}$, and $\boldsymbol{H}_{\mathrm{hom}}^{(l-1)}$ denotes the node embedding after l-layer propagation.

In comparison, when propagating with the non-informative prior, we integrate a transition matrix $\boldsymbol{T} \in \mathbb{R}^{C\times C}$ to model the relationship between connected nodes. The transition probability is given by

$$\tilde{\boldsymbol{T}} = \mathrm{softmax}(\boldsymbol{T}). \tag{7}$$

$\tilde{T}_{ij}$ indicates the probability that the neighbor of a node of class i is a node of class j. $\boldsymbol{T}$ is initialized with an all-zero matrix and learned via the EM algorithm. The detailed designs of the propagation with a non-informative prior are described in Section 4.2. The node embeddings in layer l can be derived by

$$\boldsymbol{H}_{\mathrm{non}}^{(l)} = \tilde{\boldsymbol{A}}\boldsymbol{H}_{\mathrm{non}}^{(l-1)}\tilde{\boldsymbol{T}}. \tag{8}$$

Here we obtain the node embedding in each layer l with two priors $\boldsymbol{H}_{\mathrm{hom}}^{(l)}$ and $\boldsymbol{H}_{\mathrm{non}}^{(l)}$. To capture the information from different hop ranges, we combine the node embedding in each layer with the weight $\gamma^{(l)}$ to derive the final node representations:

$$\boldsymbol{Z}_{\mathrm{hom}} = \sum_{l=0}^{L} \gamma_{\mathrm{hom}}^{(l)} \boldsymbol{H}_{\mathrm{hom}}^{(l)}, \tag{9}$$

$$\boldsymbol{Z}_{\mathrm{non}} = \sum_{l=0}^{L} \gamma_{\mathrm{non}}^{(l)} \boldsymbol{H}_{\mathrm{non}}^{(l)}, \tag{10}$$

and we can compute the loss for each propagation method as follows:

$$\mathcal{L}_{\mathrm{hom}} = \frac{1}{|\mathcal{D}_L|} \sum_{(v_i,y_i)\in\mathcal{D}_L} \ell\left(\mathrm{softmax}\left(\boldsymbol{Z}_{\mathrm{hom}}\right)_i, y_i\right), \tag{11}$$

$$\mathcal{L}_{\mathrm{non}} = \frac{1}{|\mathcal{D}_L|} \sum_{(v_i,y_i)\in\mathcal{D}_L} \ell\left(\mathrm{softmax}\left(\boldsymbol{Z}_{\mathrm{non}}\right)_i, y_i\right), \tag{12}$$

where $\mathcal{D}_L$ denotes the training dataset. Combining the loss with homophily prior and non-informative prior, the loss of the model with mixed priors is

$$\mathcal{L}_{\mathrm{mix}} = \tilde{\lambda}\mathcal{L}_{\mathrm{hom}} + (1-\tilde{\lambda})\mathcal{L}_{\mathrm{non}}, \tag{13}$$

where $\tilde{\lambda} = \mathrm{sigmoid}(\lambda)$ denotes the weight of the homophily prior and $1-\tilde{\lambda}$ denotes the weight of the non-informative prior accordingly. The value of λ depends on how homophilic the graph is. The reasons that we use the sigmoid function here and how MPGNN learns λ dynamically during the training process are illustrated in detail in Section 4.3.

The overall training pipeline of MPGNN is demonstrated in Algorithm 1. In the training stage, $\mathcal{L}_{\mathrm{mix}}$ is back-propagated. The MLP parameters θ, the transition matrix $\boldsymbol{T}$, the weight of each layer's embedding γ, and the weight of mixed priors λ are learned via gradient descent in an alternative manner. Specifically, in the E-step, MPGNN updates only the transition matrix $\boldsymbol{T}$, with all other parameters being fixed, whereas in the M-step, MPGNN updates the rest of the parameters with the transition matrix being fixed.

4.2 Non-informative prior

In MPGNN, propagation with the non-informative prior is used to solve the prior-data conflict. Instead of following the homophily prior, we learn the inter-node relationships by formulating the transition probability as a latent variable $\boldsymbol{T}$ and integrating it into the propagation process, as demonstrated in Section 4.1. Because our task is to estimate the model with an unobserved latent variable $\boldsymbol{T}$, we adopt the EM algorithm to learn the model parameters and the transition matrix simultaneously. First, we initialize $\boldsymbol{T}$ with an all-zero matrix, which means that nodes have the same probability of being connected to nodes of any class. Then, we enter an EM loop to learn the transition matrix and the model parameters alternately. In the E-step, we learn the transition matrix that minimizes $\mathcal{L}_{\mathrm{non}}$ with the current model parameters θ and γ being fixed:

$$\boldsymbol{T}^{[t]} = \arg\min_{\boldsymbol{T}} \mathcal{L}_{\mathrm{non}}(\boldsymbol{X}, \boldsymbol{A}; \theta^{[t-1]}, \gamma^{[t-1]}, \boldsymbol{T}), \tag{14}$$

whereas in the M-step, the model parameters θ and the weights for each propagation layer γ are updated

with the transition matrix $\boldsymbol{T}$ being fixed:

$$\theta^{[t]}, \gamma^{[t]} = \arg\min_{\theta,\gamma} \mathcal{L}_{\text{non}}(\boldsymbol{X}, \boldsymbol{A}; \theta, \gamma, \boldsymbol{T}^{[t]}). \quad (15)$$

Using the transition matrix, MPGNN can capture the relationships between connected nodes in the graph, thus eliminating the prior-data conflict that is inherent in traditional GNNs in heterophilic settings. In the equations presented above, we specifically illustrate the update of the heterophilic component while leaving out the training process for the homophilic component. Algorithm 1 provides a comprehensive overview of the complete training pipeline.

4.3 Mixed priors and soft switch

The non-informative prior and EM learning enhance the performance of MPGNN on the graphs with little homophily. However, we observe a performance drop when applying them to homophilic graphs. Empirically, it is observed that although the optimized transition matrix approaches an identity matrix in homophilic settings as is expected, the propagation would aggregate information following an unstable transition matrix before the identity transition matrix is learned, resulting in sub-optimal accuracy on homophilic graphs. Therefore, instead of simply replacing the homophily prior with the non-informative prior, we propose to use a mixture of these two priors and adjust their weights dynamically according to the intrinsic inter-node relationship of the graph.

The most intuitive way to choose a suitable weight is to check which prior fits the data better. However, because most GNNs are over-parameterized, they can easily memorize all the training data after training (Arpit et al., 2017; Zhang CY et al., 2017; Feldman, 2020). Fig. 2 shows the training loss curve of models with homophily and non-informative priors. After about 100 epochs of training, models with either prior can fit the training data completely (the training loss approaches zero). Meanwhile, we observe that the model with suitable prior fits the data faster in the early training stage. Based on this observation, the soft switch should be more sensitive at the initial stage and less sensitive when the training stabilizes. In MPGNN, we adopt the sigmoid function as the soft switch. The sigmoid function is shown in Fig. 3. In the initial stage, the soft switch lies in an activation region, where the module adapts to the data quickly by adjusting the weights to bias a suitable prior. Once the output approaches a tail of either 0 or 1, the soft switch turns into the saturation region, where gradients vanish and the soft switch stops learning.

Algorithm 1 MPGNN training pipeline

Require: graph $G = (V, E)$, input features $\boldsymbol{X}$, output labels $\boldsymbol{Y}$, and number of layers L.

1: Initialize the MLP parameters θ, the transition matrix $\boldsymbol{T}$, the weight for each layer γ, and the parameter of the soft switch λ.
2: Set learning rate α and number of epochs N.
3: **for** $t = 1$ to N **do**
4: Compute node embeddings using the MLP layer to derive the initial node embedding $\boldsymbol{H}^{(0)}$ according to Eq. (5).
5: Starting from $\boldsymbol{H}^{(0)}$, propagate the node embeddings with homophily and heterophily priors to derive the embeddings $\boldsymbol{H}^{(l)}_{\text{hom}}$ and $\boldsymbol{H}^{(l)}_{\text{non}}$ in each layer according to Eqs. (6)–(8).
6: Combine the node embedding in each layer to derive the final node embeddings $\boldsymbol{Z}_{\text{hom}}$ and $\boldsymbol{Z}_{\text{non}}$ according to Eqs. (9) and (10).
7: Compute loss according to Eqs. (11)–(13) and obtain the final loss $\mathcal{L}_{\text{mix}}$.
8: Compute the gradients $\nabla_\theta \mathcal{L}_{\text{mix}}$, $\nabla_\lambda \mathcal{L}_{\text{mix}}$, $\nabla_{\gamma^{(l)}_{\text{hom}}} \mathcal{L}_{\text{mix}}$, $\nabla_{\gamma^{(l)}_{\text{non}}} \mathcal{L}_{\text{mix}}$, $\nabla_{\boldsymbol{T}} \mathcal{L}_{\text{mix}}$.
9: Update the parameters using the Adam optimizer. We use the alternating training mechanism to implement the EM training process. In the E-step, MPGNN updates only the transition matrix $\boldsymbol{T}$, with all other parameters being fixed, whereas in the M-step, MPGNN updates the rest of the parameters with the transition matrix being fixed. These two steps are conducted alternately:
E step:

$$\boldsymbol{T} \leftarrow \boldsymbol{T} - \alpha \nabla_{\boldsymbol{T}} \mathcal{L}$$

M step:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$$

$$\lambda \leftarrow \lambda - \alpha \nabla_\lambda \mathcal{L}$$

$$\gamma^{(l)}_{\text{hom}} \leftarrow \gamma^{(l)}_{\text{hom}} - \alpha \nabla_{\gamma^{(l)}_{\text{hom}}} \mathcal{L}$$

$$\gamma^{(l)}_{\text{non}} \leftarrow \gamma^{(l)}_{\text{non}} - \alpha \nabla_{\gamma^{(l)}_{\text{non}}} \mathcal{L}$$

10: **end for**
11: **return** MPGNN with optimized parameters.

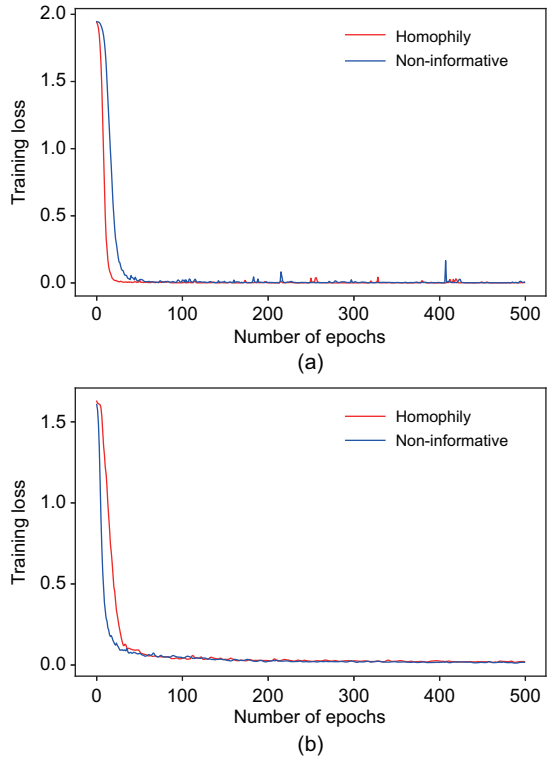

Fig. 2 The training loss of the models with the homophily prior and the non-informative prior on a homophilic graph (Cora) (a) and a heterophilic graph (Chameleon) (b). References to color refer to the online version of this figure

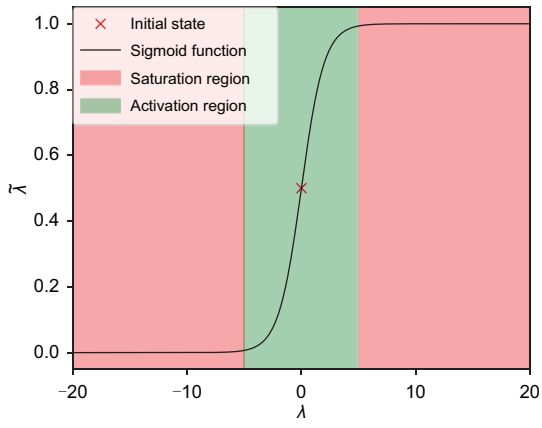

Fig. 3 The sigmoid soft switch. In the early training stage, the soft switch lies in the activation region and $\tilde{\lambda}$ approaches a tail of either 0 or 1. After that, the soft switch turns into the saturation region, where gradient vanishing happens and the soft switch stops learning. References to color refer to the online version of this figure

4.4 Initialization and regularization

4.4.1 Initialization

MPGNN is designed to be insensitive to graph types. It is important to avoid introducing any bias through the initialization. The transition matrix $\boldsymbol{T}$ is, therefore, initialized as an all-zero matrix. The weight γ for the embedding in each layer is set to $1/L$. λ is initialized as zero so that the initial weight of both priors is set to $\text{sigmoid}(0) = 0.5$.

4.4.2 Regularization

We adopt dropout regularization during the MPGNN training procedure. Dropout randomly drops a certain proportion of features during the training stage, which prevents the learned transition matrix from overfitting. For the node representation learning stage, we apply weight decay to improve the generalizability.

5 Experiments

In our experiments, we first demonstrated the performance of MPGNN on both synthetic data (Section 5.1) and real-world data (Section 5.2). We showed that the non-informative design in MPGNN can effectively learn the relationships between connected nodes under both homophily and heterophily settings in Section 5.3. An empirical analysis of the soft switch in MPGNN is presented in Section 5.4.

5.1 Evaluation on synthetic data

5.1.1 Generation of synthetic data

For the evaluation of synthetic data, we used cSBM (Deshpande et al., 2018) to generate the synthetic graphs and adopted the setting used in the evaluation of GPR-GNN (Chien et al., 2021). In this setting, the informativeness of node features and graph structure was controlled by a parameter ϕ, which took a range of $[-1, 1]$. The absolute value of ϕ indicates how much information is carried by the graph topology, while the sign of ϕ indicates whether the graphs are homophilic or heterophilic. Specifically, $\phi = 0$ indicates that only node features are informative, $\phi = 1$ indicates a strong homophilic graph, while $\phi = -1$ corresponds to a strong heterophilic graph. In terms of data splits, we

considered two random splits of the data. In sparse splitting, the training/validation/testing ratios were 2.5%/2.5%/95%, similar to the setting in Kipf and Welling (2017). In dense splitting, the ratios were 60%/20%/20%, corresponding to the setting in Pei et al. (2020).

5.1.2 Baseline models

On synthetic data, we compared MPGNN with the following baseline models: MLP, GCN (Kipf and Welling, 2017), GAT (Veličković et al., 2018), JKNet (Xu et al., 2018), APPNP (Klicpera et al., 2019), and GPR-GNN (Chien et al., 2021). For all baselines except GPR-GNN, we adopted the implementation in the PyTorch Geometric Library (Fey and Lenssen, 2019). For GPR-GNN, we used the implementation provided by the authors.

5.1.3 Model setup

For fair comparisons, we kept the setup of MPGNN consistent with that of GPR-GNN. Specifically, a two-layer MLP with 64 hidden units was used to learn the node representations, and the propagating path length L was set to 10. For the learning rate and weight decay, we tuned them in $\{0.01, 0.05\}$ and $\{0, 0.0005\}$, respectively. For the weight of each layer γ, we used the default random initialization in PyTorch. As we demonstrated in Section 4.4, the transition matrix $\boldsymbol{T}$ was initialized as an all-zero matrix. We ran each experiment 100 times with different data splits and random seeds.

5.1.4 Results

Table 1 shows the performance of MPGNN and all baseline methods on cSBM under sparse splitting. When tackling the heterophilic graphs ($\phi < 0$), MPGNN outperformed all baselines by a significant margin. Because GPR-GNN had the best performance among all baselines with the heterophilic settings, we compared our results with mainly the performance of GPR-GNN. Specifically, when $\phi = -0.50, -0.75$, and -1.00, indicating strong heterophily, MPGNN outperformed GPR-GNN by 27.42%, 7.88%, and 8.70%, respectively. We conjecture that there are two reasons for the significant improvement. First, GPR-GNN expresses the inter-node relationships using the sign of λ, which therefore allows only a bi-directional relationship. In comparison, MPGNN formulates the relationship as a transition matrix, which captures more diversified inter-node relationships. Second, compared to GPR-GNN, in which the node relationships and layer amplitudes are coupled in one parameter λ, MPGNN separates them and formulates the inter-node relationships explicitly. Therefore, MPGNN is more sensitive when capturing the relationships between nodes, leading to better accuracy under sparse splitting. Results under dense splitting confirm the assumption from another perspective. In Fig. 4, we show the test accuracy of MPGNN and GPR-GNN under dense splitting. Although both models achieved good accuracy in the end, we observed that the test accuracy of GPR-GNN fluctuated in the early stage of learning and used about 200 epochs to achieve optimal results. In comparison, MPGNN converged faster and remained stable with high accuracy in fewer than 50 epochs. However, in weak homophilic settings, we noticed that APPNP outperformed MPGNN when $\phi = 0.25, 0.50$. Under these settings, the amount of information carried by the graph structure was relatively small, thereby increasing the difficulty in capturing the weak homophily between nodes with MPGNN. As a result, MPGNN achieved a sub-optimal result compared with APPNP, which makes a homophily assumption on the node relationships.

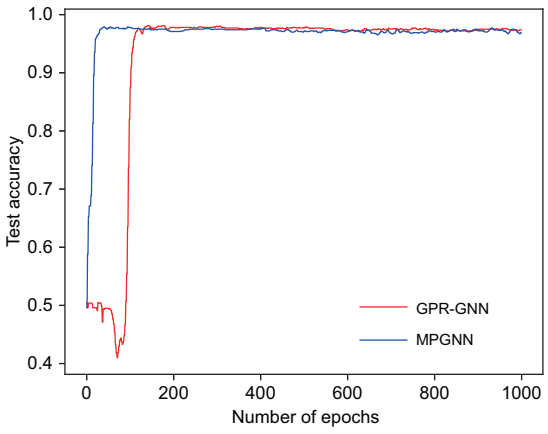

Fig. 4 The performance of MPGNN and GPR-GNN under dense splitting (ϕ=−0.75). Although both GPR-GNN and MPGNN achieved good performance after they were fully trained, GPR-GNN experienced instability in the early training stage. References to color refer to the online version of this figure

5.2 Evaluation on real-world data

5.2.1 Real-world datasets

To evaluate real-world data, we adopted three homophilic graphs and three heterophilic graphs. Specifically, for homophilic graphs, we chose three citation graphs, namely Cora (McCallum et al., 2000), Citeseer (Sen et al., 2008), and PubMed (Sen et al., 2008). In terms of heterophilic graphs, we chose three benchmark datasets used in Pei et al. (2020), including Chameleon, Squirrel, and Actor. The statistics of the datasets are summarized in Table 2. We adopted the typical data split as in related works, where the training/validation/testing splits were 60%, 20%, and 20%, respectively.

5.2.2 Baseline models

On real-world data, we compared MPGNN with the following baseline models. For GNNs without special designs for heterophilic graphs, we adopted three classic models, GCN (Kipf and Welling, 2017), GAT (Veličković et al., 2018), and APPNP (Klicpera et al., 2019), as the baselines. For GNNs designed for heterophilic graphs, we adopted GEOM-GCN (Pei et al., 2020), H_2GCN (Zhu et al., 2020), UGCN (Jin et al., 2021), BERNNET (He et al., 2021), and GPR-GNN (Chien et al., 2021) as the baselines. For each model, we ran the experiments 10 times and reported the mean accuracy with standard deviation.

5.2.3 Results

The accuracices of MPGNN and all baselines on real-world datasets are demonstrated in Table 3. In heterophilic settings, MPGNN outperformed all baselines by a notable margin. In particular, UGCN, BERNNET, GPR-GNN, and MPGNN achieved better results on the heterophilic graphs. The reason is that these four methods are designed to capture information from multi-hop neighbors, whereas other models can only reach neighbors within a two-hop radius. Among these methods, GPR-GNN and MPGNN outperformed other baselines on the heterophilic graphs, whereas compared with GPR-GNN, MPGNN still improved the accuracy by 3.64%, 7.36%, and 1.46% on Chameleon, Squirrel, and Actor, respectively. We noticed that the performance of MPGNN on homophilic graphs was slightly inferior to that of GPR-GNN. This is due to the fact that, while the soft switch can aid the model in selecting a suitable prior quickly, it has a slight impact on the model during the warm-up period, which makes it less effective on homophilic graphs than on heterophilic graphs.

Overall, MPGNN performed similarly on both synthetic and real-world datasets, with its main advantage lying in heterogeneous graph learning. Because it can adaptively learn the relationships between nodes from the data without relying on priors, it avoided the problem of prior-data conflicts and achieved good performance. However, on homophilic datasets, especially with small data volumes and insignificant homophily, MPGNN suffered from a small performance loss. The reason for this is that even though the soft switch can quickly select a suitable prior for MPGNN, an incorrect prior during the warm-up phase can cause slight perturbations in model optimization, ultimately resulting in a slight decrease in performance. This loss is difficult to avoid in practice because we cannot know which prior the training data follows in advance unless we conduct multi-step training to determine the data characteristics first and select the appropriate prior accordingly, which would introduce additional labor and time costs.

Table 1 Results for cSBM under sparse splitting

Model	Accuracy (%)								
	$\phi=-1.00$	$\phi=-0.75$	$\phi=-0.50$	$\phi=-0.25$	$\phi=0$	$\phi=0.25$	$\phi=0.50$	$\phi=0.75$	$\phi=1.00$
MPGNN	**97.96±0.76**	**96.37±0.55**	**94.92±1.04**	**62.98±2.19**	59.23±1.42	67.81±1.03	84.60±1.25	**94.60±0.67**	**95.40±1.08**
GPR-GNN	89.26±0.58	88.49±1.26	67.50±0.52	58.30±1.09	58.09±0.74	67.65±1.33	84.65±0.75	93.86±0.53	95.02±1.03
APPNP	49.60±0.21	51.52±0.26	56.08±0.31	59.10±0.49	60.34±1.07	**68.01±1.57**	**85.65±1.52**	94.24±0.48	92.82±0.63
MLP	50.03±0.94	53.13±0.48	56.28±0.62	59.25±0.49	**62.03±0.74**	60.48±0.59	57.81±0.37	53.23±0.21	50.31±0.28
GCN	55.88±0.50	62.54±0.75	56.32±0.44	52.33±0.51	54.38±0.32	67.03±0.35	84.38±0.20	93.34±1.53	83.64±0.58
GAT	53.44±0.32	57.70±0.51	53.19±0.39	51.55±0.35	53.36±0.63	63.38±0.15	81.03±0.35	88.65±0.41	82.08±0.33
JKNet	51.41±0.40	58.32±0.30	52.66±0.71	50.63±0.15	52.30±0.25	65.66±0.38	84.61±0.15	93.01±0.60	90.63±0.41

The highest accuracies are in bold

Table 2 Statistics of the real-world datasets

Parameter	Value					
	Cora	Citeseer	PubMed	Chameleon	Squirrel	Actor
Number of nodes	2708	3327	19 717	2277	5201	7600
Number of edges	5278	4552	44 324	31 371	198 353	26 659
Number of features	1433	3703	500	2325	2089	7600
Number of classes	7	6	5	5	5	5
Homophily ($\mathcal{H}$)	0.656	0.578	0.644	0.024	0.055	0.008

5.3 Visualization of the transition matrix

In this subsection, we visualize the learned transition matrix to show that the design of the non-informative prior can effectively capture the relationship between connected nodes.

Fig. 5 shows the learned transition matrix on cSBM in both homophily and heterophily cases. MPGNN could capture the corresponding relationships effectively. When $\phi = 1.0$, the transition matrix is a diagonal matrix, indicating the large probability that connected nodes belong to the same class. When $\phi = -1.0$, the transition matrix is an anti-diagonal matrix, meaning that nodes of one class tend to connect to nodes of the other class.

Fig. 6 shows the learned transition matrix on two different heterophilic graphs, Chameleon and Squirrel. Although both graphs are heterophilic graphs, the relationships between connected nodes were quite different. On the Chameleon dataset, MPGNN captured a block-like relationship. Specifically, nodes of class 0 preferred to connect to the nodes of class 1, and vice versa, while nodes of classes 2–4 were more likely to be connected. As for the Squirrel dataset, we observed that nodes of all classes tended to connect with nodes of class 4. These learned relationships were consistent with the ground-truth relationships. For example, on the Chameleon dataset, among edges that had one vertex of class 0, 41% of them had the other vertex of class 1, which was three times more likely than other

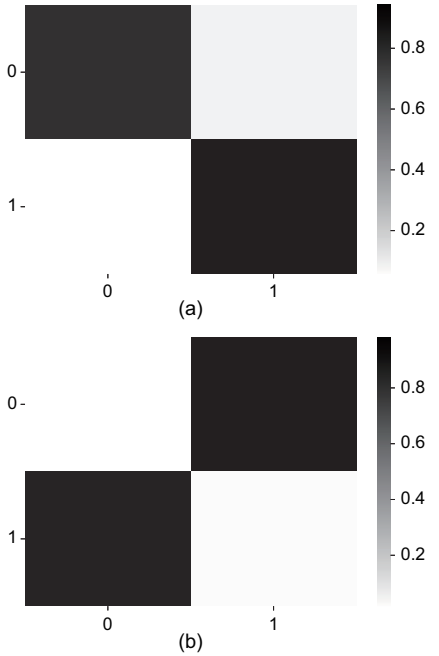

Fig. 5 The learned transition matrix on synthetic dataset cSBM: (a) homophily, ϕ=1.00; (b) heterophily, ϕ=−1.00

Table 3 Accuracy on the real-world datasets

Model	Accuracy (%)					
	Cora	Citeseer	PubMed	Chameleon	Squirrel	Actor
GCN	87.63±0.49	78.38±0.28	86.92±0.43	60.96±0.78	45.66±0.39	30.59±0.23
GAT	87.50±0.36	80.13±0.30	86.30±0.13	63.90±0.46	42.72±0.33	35.98±0.23
APPNP	88.10±0.29	80.33±0.26	89.15±0.35	51.91±0.56	34.77±0.34	38.86±0.24
GEOM-GCN	85.43±0.49	78.38±0.36	86.92±0.43	61.06±0.51	38.28±0.27	31.81±0.24
H_2GCN	87.90±0.32	76.77±0.36	88.60±0.31	58.38±1.76	32.33±1.94	34.49±1.63
GPR-GNN	**88.65±0.26**	80.01±0.15	**89.19±0.09**	67.48±0.40	49.93±0.53	39.30±0.27
UGCN	87.35±0.18	79.90±0.27	87.22±0.09	64.30±0.57	44.83±0.31	40.09±0.87
BERNNET	88.52±0.95	80.10±0.45	88.21±0.16	68.52±0.43	51.35±0.50	40.55±1.01
MPGNN	88.39±0.28	**80.38±0.20**	89.12±0.11	**71.12±1.63**	**57.29±1.38**	**40.76±1.32**

The highest accuracies are in bold

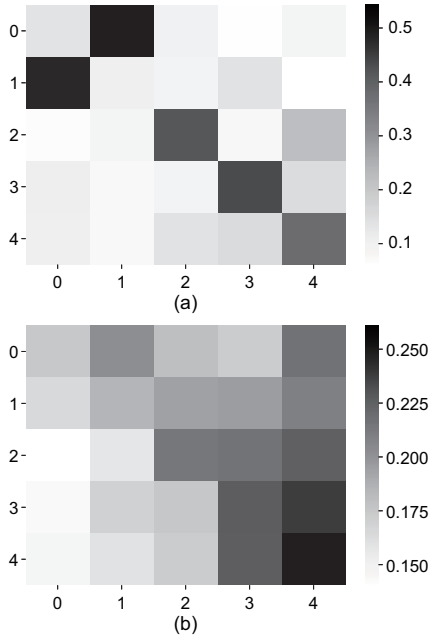

Fig. 6 The learned transition matrix on real-world datasets: (a) Chameleon; (b) Squirrel

classes. On the Squirrel dataset, for nodes of all classes, the edges that connected to nodes of class 4 occupied the largest proportion of all edges.

5.4 Analysis of the soft switch

In this subsection, we report an empirical analysis of the soft switch focusing on two aspects: (1) Can the soft switch correctly adjust the weights according to the degree of homophily? (2) Can the soft switch contribute to a performance boost?

To answer question (1), we inspected the derivative of $\mathcal{L}_{\text{mix}}$ with respect to λ:

$$\begin{aligned}\frac{\partial \mathcal{L}_{\text{mix}}}{\partial \lambda} &= \frac{\partial \mathcal{L}_{\text{mix}}}{\partial \tilde{\lambda}} \cdot \frac{\partial \tilde{\lambda}}{\partial \lambda} \\ &= (\mathcal{L}_{\text{hom}} - \mathcal{L}_{\text{non}}) \cdot \tilde{\lambda} \cdot (1-\tilde{\lambda}).\end{aligned} \tag{16}$$

This derivative reveals two soft switch behaviors. First, because $\tilde{\lambda} \in (0,1)$, the sign of the gradient depends on the magnitudes of $\mathcal{L}_{\text{hom}}$ and $\mathcal{L}_{\text{non}}$. When $\mathcal{L}_{\text{hom}} > \mathcal{L}_{\text{non}}$, the derivative is positive and the soft switch will put more weight on the non-informative prior via gradient descent. Second, $\tilde{\lambda} \cdot (1-\tilde{\lambda})$ takes the maximum at the initial point where $\tilde{\lambda} = 0.5$, which stands for the activation region. When $\tilde{\lambda}$ approaches a tail of either 0 or 1, $\tilde{\lambda} \cdot (1-\tilde{\lambda})$ approaches 0. The soft switch enters the saturation region and stops learning, which meets the design requirements we mentioned in Section 4.3.

To evaluate the performance improvement brought by the soft switch, we tested the performance of MPGNN with/without the soft switch on the Cora and Citeseer datasets with sparse splitting. As Table 4 shows, without the soft switch, training MPGNN with a non-informative prior on homophilic graphs led to sub-optimal accuracy, which verifies the need to combine the homophily prior and the non-informative prior.

Table 4 Accuracy on Cora and Citeseer under sparse splitting

Model	Accuracy (%)	
	Cora	Citeseer
With soft switch	80.75±0.16	67.88±0.38
Without soft switch	78.59±0.13	59.91±0.26

5.5 Computational complexity and convergence analysis

Because MPGNN uses GPR-GNN as the base model, we discuss their theoretical and empirical time complexity in this subsection. Theoretically, MPGNN introduces additional overhead compared to GPR-GNN in two aspects: (1) The non-informative part of MPGNN requires an extra transition matrix multiplication during training; (2) MPGNN needs to train the homophilic part and the non-informative part simultaneously, which will introduce extra time overhead. The former overhead is relatively low because the transition matrix is a small $C \times C$ matrix, where C denotes the number of classes in the data. However, the latter costs around $2\times$ overhead because MPGNN needs to train two sub-models. The empirical results in Table 5 confirmed the theoretical analysis. The non-informative part of MPGNN costed about $1.12\times$ time compared to GPR-GNN, while the whole MPGNN framework costed $2.14\times$ time compared to GPR-GNN. Nevertheless, note that there are some potential techniques to reduce the running time of the whole MPGNN framework. For instance, the propagation of two priors can be executed in parallel. Furthermore, because the soft switch will quickly disable one of the propagation paths, we can discard the unselected prior when the soft switch enters saturation to avoid executing these two propagation paths simultaneously. This technique can be implemented using the dynamic programming feature of PyTorch.

Table 5 Empirical computational complexity for 1000 epochs

Model	Computation time (s)	
	Cora	Chameleon
GPR-GNN	11.71	11.41
MPGNN (non-info)*	13.21	12.88
MPGNN	25.11	24.52

* Non-informative part of MPGNN

Another important aspect is the speed of convergence. MPGNN uses the EM algorithm to learn the model parameters and the transition matrix simultaneously. To address the concern about the low convergence rate of the EM algorithm, we compared the models with the homophily prior (M_{hom}) and the non-informative prior (M_{non}) on both homophilic and heterophilic graphs. The comparison metric we used was the number of iterations required for the model to achieve its optimal validation performance. Results are shown in Table 6. From the results, we can observe that M_{non} converged more slowly than M_{hom} on homophilic data. However, when dealing with heterophilic data that did not comply with the homophilic prior, the EM method had a higher convergence rate than the one based on the homophilic assumption.

Table 6 Number of epochs to achieve optimal validation performance

Prior type	Number of epochs	
	Cora	Chameleon
M_{hom}	32	154
M_{non}	58	83

This phenomenon can be explained by the two-sided effect of the EM algorithm. On one hand, the EM method requires alternating optimization of model parameters and the transition matrix. At the initial stage of training, the model parameters and transition matrices are not accurate enough, leading to an inaccurate optimization direction and a decrease in the speed of convergence. On the other hand, the EM method can capture the relationships between connected nodes, which helps the model converge faster. When the homophilic prior is not suitable for heterophilic data, the EM method can capture the relationships between connected nodes more accurately. Such benefits overwhelm the initial low convergence speed of the EM method, leading to a higher convergence rate on heterophilic data.

5.6 Results on heterophilic graphs at larger scales

To further verify the effectiveness of MPGNN in capturing the relationships between connected nodes, we tested the performance of MPGNN on three larger-scale heterophilic graphs proposed in Lim et al. (2021), namely Pokec (Leskovec and Krevl, 2014), arXiv-year (Hu et al., 2020), and snap-patents (Leskovec et al., 2005; Leskovec and Krevl, 2014). The statistics of these three datasets are shown in Table 7. The experiments were run on an NVIDIA RTX 3090 with 24 GB GPU memory. For baselines, we followed the best-tuned hyperparameters of Lim et al. (2021). For MPGNN, we set the learning rate as 0.05, the weight decay as 0, and the dropout as 0.2. The results are shown in Table 8. From the results, we observed that MPGNN achieved state-of-the-art performance on all three datasets. The improvement over GPR-GNN demonstrated that MPGNN can capture the relationships between connected nodes effectively, which is crucial for the performance of GNNs on heterophilic graphs.

Table 7 Statistics of the larger-scale datasets

Parameter	Value		
	Pokec	arXiv-year	snap-patents
Number of nodes	1 632 803	169 343	2 923 922
Number of edges	30 622 564	1 166 243	13 975 788
Number of features	65	128	269
Number of classes	2	5	5
$(\mathcal{H})$	0.445	0.222	0.073

$(\mathcal{H})$ denotes the average edge homophily

Table 8 Performance on the larger-scale datasets

Model	Accuracy (%)		
	Pokec	arXiv-year	snap-patents
GCN	75.45±0.17	46.02±0.26	45.65±0.04
GAT	71.77±6.18	46.05±0.51	45.37±0.44
APPNP	62.58±0.08	38.15±0.26	32.19±0.07
H_2GCN	–	49.09±0.10	–
GPR-GNN	78.83±0.05	45.07±0.21	40.19±0.03
MPGNN	**80.39±0.31**	**49.64±0.19**	**52.48±0.04**

– denotes out of memory. The highest accuracies are in bold

6 Conclusions

In this work, we demonstrate that the suboptimal performance of GNNs on heterophilic graphs is due to prior-data conflicts. We propose MPGNN, which solves such conflicts via (1) integrating a non-informative prior and (2) using a soft switch to balance the homophily prior and non-informative prior adaptively. The integrated non-informative prior can effectively capture the relationships between connected nodes, therefore boosting the performance of MPGNN on heterophilic graphs. Meanwhile, the design of the soft switch can adjust the weights of the homophily prior and non-informative prior adaptively, which helps MPGNN choose a suitable prior according to the intrinsic inter-node affinities of the graphs. With these two designs, MPGNN achieves state-of-the-art performance on both homophilic and heterophilic graphs.

Contributors

Xugang WU designed the research. Kai LU, Huijun WU, Ruibo WANG, and Xu ZHOU improved the design. Xugang WU implemented the method and drafted the paper. Huijun WU and Kai LU helped organize the paper. Xugang WU revised and finalized the paper.

Conflict of interest

All the authors declare that they have no conflict of interest.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

Arpit D, Jastrzebski S, Ballas N, et al., 2017. A closer look at memorization in deep networks. Proc 34th Int Conf on Machine Learning, p.233-242.

Chiang WL, Liu XQ, Si S, et al., 2019. Cluster-GCN: an efficient algorithm for training deep and large graph convolutional networks. Proc 25th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining, p.257-266. https://doi.org/10.1145/3292500.3330925

Chien E, Peng JH, Li P, et al., 2021. Adaptive universal generalized PageRank graph neural network. Proc 9th Int Conf on Learning Representations.

Ciotti V, Bonaventura M, Nicosia V, et al., 2016. Homophily and missing links in citation networks. *EPJ Data Sci*, 5(1):7. https://doi.org/10.1140/epjds/s13688-016-0068-2

Dempster AP, Laird NM, Rubin DB, 1977. Maximum likelihood from incomplete data via the EM algorithm. *J R Stat Soc Ser B (Methodol)*, 39(1):1-22. https://doi.org/10.1111/j.2517-6161.1977.tb01600.x

Deshpande Y, Montanari A, Mossel E, et al., 2018. Contextual stochastic block models. Proc 32nd Int Conf on Neural Information Processing Systems, p.8590-8602.

Dong HD, Chen JW, Feng FL, et al., 2021. On the equivalence of decoupled graph convolution network and label propagation. Proc Web Conf, p.3651-3662. https://doi.org/10.1145/3442381.3449927

Feldman V, 2020. Does learning require memorization? A short tale about a long tail. Proc 52nd Annual ACM SIGACT Symp on Theory of Computing, p.954-959. https://doi.org/10.1145/3357713.3384290

Fey M, Lenssen JE, 2019. Fast graph representation learning with PyTorch geometric. https://arxiv.org/abs/1903.02428

Grover A, Leskovec J, 2016. node2vec: scalable feature learning for networks. Proc 22nd ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining, p.855-864. https://doi.org/10.1145/2939672.2939754

Hamilton WL, Ying R, Leskovec J, 2017. Inductive representation learning on large graphs. Proc 31st Int Conf on Neural Information Processing Systems, p.1025-1035.

He MG, Wei ZW, Huang ZF, et al., 2021. BernNet: learning arbitrary graph spectral filters via Bernstein approximation. Proc 35th Int Conf on Neural Information Processing Systems, p.14239-14251.

Hochreiter S, Schmidhuber J, 1997. Long short-term memory. *Neur Comput*, 9(8):1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hu WH, Fey M, Zitnik M, et al., 2020. Open graph benchmark: datasets for machine learning on graphs. Proc 34th Int Conf on Neural Information Processing Systems, Article 1855.

Huang Q, He H, Singh A, et al., 2021. Combining label propagation and simple models outperforms graph neural networks. Proc 9th Int Conf on Learning Representations.

Jeh G, Widom J, 2003. Scaling personalized web search. Proc 12th Int Conf on World Wide Web, p.271-279. https://doi.org/10.1145/775152.775191

Jin D, Yu ZZ, Huo CY, et al., 2021. Universal graph convolutional networks. Proc 35th Int Conf on Neural Information Processing Systems, p.10654-10664.

Kipf TN, Welling M, 2017. Semi-supervised classification with graph convolutional networks. Proc 5th Int Conf on Learning Representations.

Klicpera J, Bojchevski A, Günnemann S, 2019. Predict then propagate: graph neural networks meet personalized PageRank. Proc 7th Int Conf on Learning Representations.

Krizhevsky A, Sutskever I, Hinton GE, 2017. ImageNet classification with deep convolutional neural networks. *Commun ACM*, 60(6):84-90. https://doi.org/10.1145/3065386

Leskovec J, Krevl A, 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data/

Leskovec J, Kleinberg J, Faloutsos C, 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. Proc 11th ACM SIGKDD Int Conf on Knowledge Discovery in Data Mining, p.177-187. https://doi.org/10.1145/1081870.1081893

Lim D, Hohne F, Li XY, et al., 2021. Large scale learning on non-homophilous graphs: new benchmarks and strong simple methods. Proc 35th Int Conf on Neural Information Processing Systems, p.20887-20902.

Ma JX, Zhou C, Cui P, et al., 2019. Learning disentangled representations for recommendation. Proc 33rd Int Conf on Neural Information Processing Systems, Article 513.

McCallum AK, Nigam K, Rennie J, et al., 2000. Automating the construction of Internet portals with machine learning. *Inform Retr*, 3(2):127-163. https://doi.org/10.1023/A:1009953814988

McLachlan GJ, Krishnan T, 1997. The EM Algorithm and Extensions. John Wiley & Sons, New York, USA.

McPherson M, Smith-Lovin L, Cook JM, 2001. Birds of a feather: homophily in social networks. *Ann Rev Sociol*, 27:415-444. https://doi.org/10.1146/annurev.soc.27.1.415

Pei HB, Wei BZ, Chang KCC, et al., 2020. Geom-GCN: geometric graph convolutional networks. Proc 8th Int Conf on Learning Representations.

Sen P, Namata G, Bilgic M, et al., 2008. Collective classification in network data. *AI Mag*, 29(3):93-106. https://doi.org/10.1609/aimag.v29i3.2157

Tang J, Sun JM, Wang C, et al., 2009. Social influence analysis in large-scale networks. Proc 15th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining, p.807-816. https://doi.org/10.1145/1557019.1557108

Veličković P, Cucurull G, Casanova A, et al., 2018. Graph attention networks. Proc 6th Int Conf on Learning Representations.

Wang Z, Wang CK, Pei JS, et al., 2016. Causality based propagation history ranking in social network. Proc 25th Int Joint Conf on Artificial Intelligence, p.3917-3923.

Wu F, Souza AHJr, Zhang TY, et al., 2019. Simplifying graph convolutional networks. Proc 36th Int Conf on Machine Learning, p.6861-6871.

Wu ZH, Pan SR, Long GD, et al., 2020. Connecting the dots: multivariate time series forecasting with graph neural networks. Proc 26th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining, p.753-763. https://doi.org/10.1145/3394486.3403118

Xu K, Li CT, Tian YL, et al., 2018. Representation learning on graphs with jumping knowledge networks. Proc 35th Int Conf on Machine Learning, p.5449-5458.

Yang TM, Wang YJ, Yue ZH, et al., 2022. Graph pointer neural networks. Proc 36th AAAI Conf on Artificial Intelligence, p.8832-8839. https://doi.org/10.1609/aaai.v36i8.20864

Ying R, He RN, Chen KF, et al., 2018. Graph convolutional neural networks for web-scale recommender systems. Proc 24th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining, p.974-983. https://doi.org/10.1145/3219819.3219890

Zhang CY, Bengio S, Hardt M, et al., 2017. Understanding deep learning requires rethinking generalization. Proc 5th Int Conf on Learning Representations.

Zhang MH, Chen YX, 2018. Link prediction based on graph neural networks. Proc 32nd Int Conf on Neural Information Processing Systems, p.5171-5181.

Zhang ZW, Cui P, Zhu WW, 2022. Deep learning on graphs: a survey. *IEEE Trans Knowl Data Eng*, 34(1):249-270. https://doi.org/10.1109/TKDE.2020.2981333

Zhao JL, Dong YX, Ding M, et al., 2021. Adaptive diffusion in graph neural networks. Proc 35th Int Conf on Neural Information Processing Systems, p.23321-23333.

Zhu J, Yan YJ, Zhao LX, et al., 2020. Beyond homophily in graph neural networks: current limitations and effective designs. Proc 34th Int Conf on Neural Information Processing Systems, Article 653.

Zhu J, Rossi RA, Rao A, et al., 2021. Graph neural networks with heterophily. Proc 35th AAAI Conf on Artificial Intelligence, p.11168-11176. https://doi.org/10.1609/aaai.v35i12.17332