Graph Neural Network leveraging Higher-order Class Label Connectivity for Heterophilous Graphs

Takuto Takahashi ✉, Itsuki Nakayama, Takahiro Mitani, Ryosuke Kikuchi, Yuya Sasaki, and Makoto Onizuka

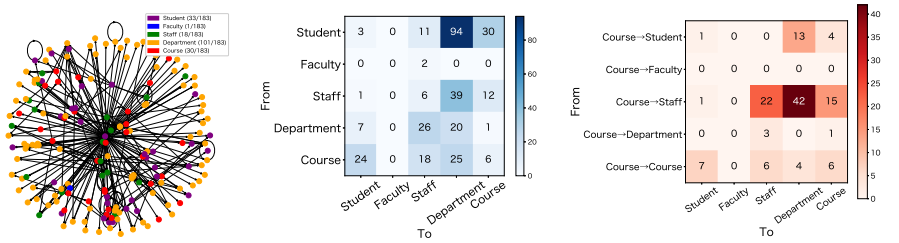
The University of Osaka, 1-5 Yamadaoka, Suita, Osaka, Japan {takahashi.takuto, nakayama.itsuki, mitani.takashiro, kikuchi.ryosuke, sasaki, onizuka}@ist.osaka-u.ac.jp

Abstract. Node classification in graph neural networks (GNNs) has been widely applied in various fields of graph analysis. GNNs achieve high-accuracy node classification in homophilous graphs, where nodes with the same class label tend to be connected. However, their performance remains limited in heterophilous graphs, where nodes with different class labels are more likely to be connected. In particular, current GNNs derived from graph convolutional networks cannot capture higher-order class label connectivity, which is frequently observed in real-world heterophilous graphs. To address this issue, we propose a novel classifier, Label Context Classifier (LCC), designed to capture higher-order class label connectivity in directed graphs. LCC estimates the class label of a target node by leveraging *label context* embeddings that are generated through four distinct types of walks. In addition, our approach allows the integration of LCC and any GNN by adaptively learning their importance. Experimental results demonstrate that GNNs integrated with LCC outperform SOTA methods and the label context embeddings improve the node classification performance in heterophilous directed graphs.

Keywords: Graph neural networks · Node classification · Heterophilous graphs

1 Introduction

Node classification in graphs is one of the important tasks in graph analysis, aiming to predict the class labels of nodes. This task has a wide range of applications, including the analysis of social networks and biological networks, such as genes and proteins [3, 16, 17, 25]. A representative approach for node classification is Graph Neural Networks (GNNs) [1, 4, 6, 7, 9, 11, 18, 21–23, 28]. Traditional GNNs such as Graph Convolutional Network (GCN) [7] are designed for homophilous graphs, where nodes with the same class labels/features are more likely to be connected. However, their effectiveness is limited for heterophilous graphs, where nodes with different class labels/features tend to be connected [26, 27]. To improve the performance in heterophilous graphs, GNNs that capture the characteristics of heterophilous graphs have been actively proposed [1, 8–10, 22, 26, 28]. Nevertheless, there are still cases where the accuracy of these GNNs does not surpass that of multilayer perceptrons (MLPs), which rely only on node features without using edges. This result indicates that these GNNs do not fully leverage the structural information of graphs [10].



(a) Graph visualization (b) 1st-order connectivity (c) 2nd-order connectivity: we only show five rows of Course→* paths out of 25 rows

Fig. 1: The class label connectivity in the Texas dataset. (a) visualization with five class labels. (b) 1st-order connectivity (#edges) from class label in y-axis to class label in x-axis. (c) 2nd-order connectivity from class label path (e.g., Course→Student) in y-axis to class label in x-axis.

Motivation. An interesting observation we found on the weakness of current GNNs derived from GCN is that they fail to capture the *higher-order class label connectivity* in graphs, that is, how class labels are connected through multiple hops using directed edges. This weakness is caused by the fact that GCN transforms the node classification problem into a simple classification problem using graph convolution operation: after applying the graph convolution operations, GCN trains the model from embedding space to class label space, so it ignores the class label connectivity among nodes, particularly appeared in the training sets. This limitation also applies to more advanced methods such as [24, 26]. While they utilize a class compatibility matrix corresponding to 1st-order class label connectivity, they do not capture higher-order class connectivity.

Strong higher-order class label connectivity often appears in real-world graphs. Figure 1 illustrates such examples in the heterophilous directed graph of the Texas dataset. Figure 1 (a) visualizes the graph with five class labels (i.e., Student, Faculty, Staff, Department, and Course). Figure 1 (b) shows the 1st-order connectivity, depicting the number of directed edges from the class label on the y-axis to the one on the x-axis. We observe that there is strong connectivity from Student nodes to Department nodes, whereas there is no connectivity to Faculty nodes. Similarly, Figure 1 (c) also reveals strong/weak 2nd-order class label connectivity. For example, the connectivity from Course→Staff class label path to Department nodes is strong, whereas no connectivity to Faculty nodes.

Contribution. In order to capture such higher-order class label connectivity, we propose a novel classifier, Label Context Classifier (LCC), which trains the model using *label walks*, which are sequences of class labels on walks. First, we extract four fundamental types of label walks from a target node: forward walk, backward walk, sibling walk, and guardian walk. They are mutually exclusive, and each walk type captures a different aspect of class label connectivity. We generate *label context embeddings* that capture the label context by training a model using the label walks, an idea inspired by word2vec [13]. Then, we train LCC using the concatenation of node features and the label context embeddings obtained from different types of label walks. Since LCC complements the capability of GNNs, we integrate LCC and any GNN by adaptively learning their importance using validation loss without additional model training.

The contributions of this paper are as follows:

- We propose the Label Context Classifier (LCC), which estimates the class label of a target node by capturing the higher-order class label connectivity across directed graphs. LCC estimates the class label using the label context embeddings generated from four types of label walks.
- We can integrate LCC and any GNN by adaptively learning their importance weights without additional model training.
- Experimental results confirm that our proposal outperforms SOTA methods and the label context embeddings actually enhance node classification performance in heterophilous directed graphs.

The structure of this paper is as follows. Section 2 describes related work, and then Section 3 explains preliminary knowledge. Sections 4 and 5 present the details of the Label Context Classifier and how to integrate it and any GNN, respectively. Section 6 conducts experiments to demonstrate the effectiveness of the proposed method for heterophilous directed graphs. Finally, Section 7 concludes this paper.

2 Related Work

Traditional GNNs Since Graph Convolutional Network (GCN) [7] has emerged, numerous methods have been proposed for node classification using graph convolution. The graph convolution ensures 1st-order node proximity, which makes the adjacent node representations similar. Therefore, a family of GCN is suitable for homophily graphs but not for heterophilous graphs.

Graph Attention Networks (GAT) [18] improve accuracy by employing a self-attention mechanism that learns relative weights between connected node pairs. JK-Net [21] is a method that aggregates outputs from multiple layers of a GNN to integrate information at different layer levels. APPNP [4] first transforms the initial node features using an MLP and then applies a personalized PageRank-based iterative propagation mechanism to distribute information among nodes. In addition, sampling-based GNNs [6, 23] are scalable methods that compute node representations using subgraphs extracted from the input graph. GraphSAGE [6] samples a fixed number of neighbors uniformly for each node. GraphSAINT [23] samples subgraphs and learns graph representations by combining information from multiple subgraphs.

GNNs for Heterophilous Graphs Real-world graphs sometimes exhibit heterophily, where nodes of different attributes/classes are more likely to be connected. There have been several GNNs designed for heterophilous graphs [1, 8, 9, 12, 22, 28]. However, these methods also suffer from the same limitation as the GNNs derived from GCN in the sense that they also transform the node classification problem into a simple classification problem by training the model from embedding space to class label space, so they ignore the class label connectivity.

H2GCN [28], a representative GNN designed for heterophilous graphs, separates the processing of a node's own features from those of its neighbors, utilizes high-order

neighborhood information, and employs an appropriate aggregation function to capture complex relationships between nodes. LINKX [9] independently processes node features and adjacency matrix information, learning both in parallel to effectively leverage multiple sources of information. GloGNN [8] combines local node features with global features that capture relationships between distant but structurally similar nodes. Recent studies have further advanced this area. CAGNNs [1] introduce a novel metric based on the distinguishability of neighboring nodes, decomposing node features into representation and aggregation components. A mixer module is then used to adaptively evaluate neighboring information for each node. Adaptive Channel Mixing (ACM) [10] is a GNN framework designed to address heterophily by adaptively combining aggregation, diversification, and identity channels at the node level. ACM allows nodes to learn different weights for each channel, effectively capturing local heterophily without requiring high-order filters or increased computational resources. LG-GNN [22] achieves high-accuracy node classification for heterophilous graphs by adaptively integrating global structural similarity and local feature similarity between nodes. This design effectively considers node relationships during the information aggregation and propagation process.

Other related work There are methods [24, 26] that utilize the class compatibility matrix, which corresponds to 1st-order class label connectivity. As a specific example, CPGNN [26] trains the class compatibility matrix using training sets and a prior belief estimator. However, it has a weakness in that it does not capture higher-order class connectivity.

A meta-path is a predefined sequence of node and edge types in order to capture higher-order connectivity and semantic relationships between different entities in heterogeneous information networks (HIN). For example, MetaPath2Vec [2] generates constrained random walks for learning embeddings, while HAN [20] employs attention mechanisms to aggregate multiple meta-paths. Recent studies [19] focus on automatic meta-path discovery to enhance model generalization. These methods assume that all nodes and edges are typed, and meta-paths are defined based on those types. In contrast, our approach is applicable to graphs without predefined types, and we define four types of direction-aware fundamental walks.

node2vec [5] is a method to generate node embedding in order to capture homophily and structural equivalence. Our proposal and node2vec share some common characteristics, such as generating walks and using word2vec. However, there are two major differences. First, node2vec does not use class labels as input and is not designed for directed graphs. Second, the type of walk is limited to only a single type, and the parameters that determine the balance between breadth-first and depth-first search must be manually set by the user. In contrast, in order to capture class connectivity, we define four types of direction-aware class label walks, and their importance is learned automatically without manual intervention.

3 Preliminary

Graph We consider a directed graph $G = (V, E)$, which consists of a node set V with n nodes and a directed edge set E with m edges. The adjacency matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$ is defined such that $a_{ij} = 1$ if $(v_i, v_j) \in E$, and $a_{ij} = 0$ otherwise. Additionally, we define the feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, where each node is assigned a d-dimensional feature vector. Each node v has a unique class label $y_v \in \{1, \ldots, C\}$ (number of classes: C), and the class label vector is denoted as $\mathbf{y}$. For clarity in explanations, we refer to the starting node of a directed edge as the parent node and the ending node as the child node. Additionally, we use the terms, sibling nodes and guardian nodes, which are naturally defined based on the parent-child relationship between nodes.

We define heterophilous graphs after defining edge homophily [28]. The edge homophily [28] is calculated as follows.

$$H(G) = \frac{\sum_{0 \leq i,j < n} a_{ij} \delta(y_{v_i}, y_{v_j})}{m}, \tag{1}$$

where $\delta(y_{v_i}, y_{v_j})$ returns one if $y_{v_i} = y_{v_j}$ otherwise zero. We define graphs with low $H(G)$ as heterophilous graphs. A lower edge homophily indicates a stronger tendency toward heterophily.

Problem Definition (Node classification) We split a node set V into a training set V_{train}, validation set V_{val}, and test set V_{test}. Given adjacency matrix $\mathbf{A}$, feature matrix $\boldsymbol{X}$, and node class labels in V_{train} and V_{val}, we predict the labels of the nodes in V_{test}.

4 Label Context Classifier

GNNs fail to capture the higher-order class label connectivity, which often appears in real-world graphs, as we described in Section 1. This weakness is caused by the fact that GNNs transform the node classification problem into a simple classification problem: GNNs train the model from embedding space to class label space, so they ignore the class label connectivity among nodes, particularly appeared in the training sets.

To this end, we propose a novel classifier, Label Context Classifier (LCC), which trains the model using various types of label walks in order to capture higher-order class label connectivity. Our method consists of the following steps as illustrated in Figure 2: (1) extract four types of fundamental label walks: a simple walk (forward walk, backward walk) and a mixture of forward and backward walks (sibling walk, guardian walk). They are mutually exclusive and each walk type captures a different aspect of class label connectivity, (2) generate embeddings that capture the label context by training a model using the label walks, and (3) train LCC using the concatenation of node attributes and the label context embeddings obtained from different types of label walks. This enables the model to appropriately select suitable embeddings for node classification.

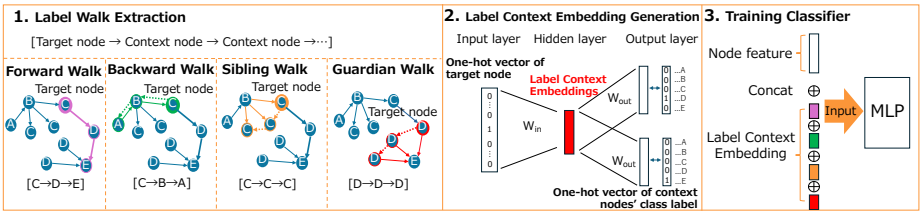


Fig. 2: The framework of Label Context Classifier (LCC). LCC consists of three steps: 1) extraction of four types of label walks, 2) generation of label context embeddings, and 3) training a classifier using the label context embeddings.

Table 1: Summary of label walk types and their characteristics.

Walk type	Description	Traverse edge direction	Class label connectivity order (#hops)
Forward	Depth-first search-based walk	forward only	walk length
Backward	Inverse of forward walk	backward only	walk length
Sibling	Walk on sibling nodes	backward + forward	2
Guardian	Inverse of sibling walk	forward + backward	2

4.1 Label Walk Extraction

A label walk is defined as a walk obtained from a graph where the nodes in the walk are replaced with their labels in the training set. We refer to the first label in the walk as the target label and the remaining labels as context labels. If a node in the walk is not in the training dataset, we use the null label in the label walk, and it is ignored in the label context embedding step.

Table 1 presents a summary of the characteristics of the four label walk types, indicating they are mutually exclusive and each walk type captures a different aspect of higher-order class label connectivity. Step 1 in Figure 2 shows examples of label walk types: forward walk (C, D, E), backward walk (C, B, A), sibling walk (C, C, C), and guardian walk (D, D, D). The details are described in the following sections.

Forward Walk The forward walk aims to capture the higher-order class label connectivity directed from a target label down to neighboring context labels. We extract forward walks using the depth-first search that follows the forward direction of directed edges. For the given target node v_0 and the length w of label walk, a forward walk (FW) is formulated as follows:

$$FW(v_0) = (y_{v_0}, y_{v_1}, \ldots, y_{v_w}), \qquad \text{s.t.}(v_i, v_{i+1}) \in E, i \in \{0, 1, \ldots, w-1\}. \quad (2)$$

The order of class label connectivity is walk length. For example, it is 1st-order if the walk length is 1 (i.e., 1 hop).

Figure 1 shows an example: the Student nodes are more likely to connect to the Department nodes (1st order connectivity), and those Department nodes are more likely to connect to the Staff nodes (2nd order connectivity).

Backward Walk The backward walk is the inverse notion of the forward walk. The reason why we introduce the backward walk in addition to the forward walk is that the edge direction is user-defined, so both directions are useful. Also, the backward walk is useful for the sink nodes (i.e., nodes without outgoing edges), because they cannot utilize forward walks. We extract backward walks using the depth-first search that follows the reverse direction of directed edges. Similar to the forward walk, a backward walk (BW) is defined as follows:

$$BW(v_0) = (y_{v_0}, y_{v_1}, \ldots, y_{v_w}), \qquad \text{s.t.}(v_{i+1}, v_i) \in E, i \in \{0, 1, \ldots, w-1\}. \quad (3)$$

Figure 1 shows an example: the Department nodes are more likely to be connected from the Student nodes (1st order connectivity), and those Student nodes are more likely to be connected from the Course nodes (2nd order connectivity).

Sibling Walk In addition to the simple forward/backward walks, we introduce the sibling walk, our new idea which is a mixture of forward and backward walks. The motivation for introducing sibling walks is that sibling nodes (the child nodes connected to the same parent node) often share the same class label for certain target nodes. We extract sibling walks by 1) traversing backward to a parent node of the target node v_0, and then 2) repeatedly traversing forward to its child nodes until reaching the desired label walk length w. A sibling walk (SW)[1] is formulated as follows:

$$\begin{aligned} SW(v_0) &= (y_{v_0}, y_{v_1}, \ldots, y_{v_w}) \\ &\text{s.t.} \quad \exists p \in \text{parents}(v_0), \quad v_1, \ldots, v_w \in \text{children}(p) \setminus \{v_0\}. \end{aligned} \quad (4)$$

where parents(v) and children(v) are parent nodes and child nodes of v, respectively. The order of class label connectivity is 2 regardless of the walk length, because the target node and the sibling nodes are 2-hops apart.

Figure 1 shows an example: the sibling walk captures the connectivity between the nodes labeled as Student whose parent nodes are labeled as Course.

Guardian Walk The guardian walk is the inverse notion of the sibling walk. Similarly to the sibling walk, the motivation for introducing guardian walks is that guardian nodes often share the same class label for certain target nodes. Compared to the general walks based on depth-first or breadth-first search, the sibling walk and guardian walk extract only siblings and guardians, which mitigates the noisy effect on the downstream classifier. Indeed, our experiments in Section 6 verify that the sibling walk and guardian walk significantly improve the accuracy. We extract guardian walks by 1) traversing forward to a child node of the target node v_0, and then 2) repeatedly traversing backward to its parent nodes until reaching the desired label walk length w. A guardian walk (GW) is defined as follows:

$$\begin{aligned} GW(v_0) &= (y_{v_0}, y_{v_1}, \ldots, y_{v_w}) \\ &\text{s.t.} \quad \exists c \in \text{children}(v_0), \quad v_1, \ldots, v_w \in \text{parents}(c) \setminus \{v_0\}. \end{aligned} \quad (5)$$

[1] Since a sibling walk traverses multiple sibling nodes without edges, it does not strictly follow the "walk" definition in the graph theory.

Algorithm 1 Label context embedding matrix generation

Input: adjacency matrix $\mathbf{A}$, class label $\mathbf{y}$, embedding dimension d', label walk length w, number of label walk k, epoch T
Output: label context embedding matrix $\mathbf{Z} \in \mathbb{R}^{n \times d'}$

Initialize
Initialize the label context embedding $\mathbf{z}_v \in \mathbb{R}^{d'}$ of node $v \in V$ to a random value
Create an one-hot vector ℓ_v of class label for node v from $\mathbf{y}$
Extract label walks
for $v \in V$ **do**
  if label walk is forward / backward walk **then**
    for $i = 1, \ldots, k$ **do**
      Extract label walks (forward, backward walks) for v
    end for
  else if label walk is sibling / guardian walk **then**
    Extract a label walk (sibling, guardian walk) for v
  end if
end for
Train the model for label context embedding
for $t = 1, \ldots, T$ **do**
  for each label walk **do**
    Calculate output embedding $\hat{\ell}_v = \mathbf{z}_v^\top \mathbf{W}_{\mathbf{out}}$
    Calculate cross entropy as loss $\mathcal{L} = \sum_{\ell_u \in contexts(v)} \mathcal{L}_{CE}(\hat{\ell}_v, \ell_u)$
    Update $\mathbf{z}_v$, $\hat{\mathbf{y}}_v$ and $\mathbf{W}_{\mathbf{out}}$ to minimize $\mathcal{L}$
  end for
end for
Output label context embedding matrix
return $\mathbf{Z} = [\mathbf{z}_v]_{v \in V}$

Figure 1 shows an example: the guardian walk captures the connectivity between the nodes labeled as Student whose child nodes are labeled as Department.

4.2 Label Context Embeddings

We generate embeddings that capture the label context using the label walks. Our purpose is to verify the effectiveness of leveraging label walks for capturing higher-order class label connectivity, so we use a simple two-layer MLP to train the model to predict the context labels using the target node as input. The idea is inspired by the Skip-gram model of word2vec [13]. Extending our framework to use more advanced techniques, such as transformers, is part of future work. Specifically, the target node is represented as a one-hot node vector in the input layer, and the context labels are represented as a one-hot label vector in the output layer. The model is trained to minimize the loss between the estimated output and the ground-truth context labels, ensuring that the internal layer represents the embedding for a given input node.

Step 2 in Figure 2 shows an overview of generating label context embeddings. Let $\mathbf{Z} \in \mathbb{R}^{n \times d'}$ be the label context embedding matrix obtained in the hidden layer for each node where d' represents the dimension of the embedding vector. The output layer

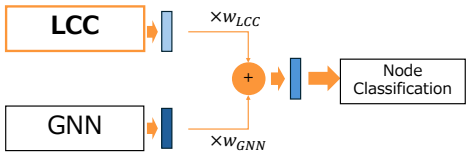


Fig. 3: Integration of LCC and any GNN. The final prediction is computed by average outputs from LCC and GNN weighted by their importance.

embedding $\hat{\ell}_v$ is obtained by multiplying the label context embedding $\mathbf{z}_v$ of the target node v by a weight matrix $\mathbf{W_{out}} \in \mathbb{R}^{d' \times C}$.

$$\hat{\ell}_v = \mathbf{z}_v^\top \mathbf{W_{out}} \tag{6}$$

Then, the cross-entropy loss $\mathcal{L}_{CE}$ is computed with respect to the one-hot vector ℓ_u of the ground-truth context label. The total loss $\mathcal{L}$ over all pairs of $\hat{\ell}_v$ of target node v and its context label ℓ_u is formulated as follows:

$$\mathcal{L} = \sum_{\ell_u \in contexts(v)} \mathcal{L}_{CE}(\hat{\ell}_v, \ell_u), \tag{7}$$

where $contexts(v)$ are context labels in the label walk starting from the target node v. Finally, the model is trained to update $\mathbf{Z}$ and $\mathbf{W_{out}}$ to minimize the loss $\mathcal{L}$. Remember that the context label is null if its corresponding node is not in the training set. We ignore the null label in the model training. In addition, when the context node is the same as the target node in a label walk, we exclude it from the loss computation to prevent information leaks. Algorithm 1 presents the details for generating label context embeddings.

4.3 Training Classifier using Label Context Embeddings

To perform node classification, we finally train a node classifier (LCC) that predicts the class label of the target node using its label context embeddings. Since different types of label walks capture different aspects of class label connectivity, we train the MLP classifier using the concatenation of node features and all label context embeddings obtained from different types of label walks.

5 Integration of LCC and GNN

LCC captures the higher-order class label connectivity that GNNs fail to capture. Therefore, LCC complements the capability of GNNs. Since there are several GNNs designed for heterophilic graphs, such as H2GCN, LINKX, and GloGNN, we integrate LCC and one of these models and adaptively learn the importance of both LCC and GNN to achieve high accuracy. An overview of integration is illustrated in Figure 3. Since our

Algorithm 2 Integration of LCC and any GNN

Input: Adjacency matrix $\mathbf{A}$, Feature matrix $\mathbf{X}$, Class label $\mathbf{y}$,
Trained node classification models (LCC, GNN), Temperature T
Output: Ensemble output $\mathbf{Y}^{GNN+LCC}$
1: **### Calculate model outputs ###**

$$\mathbf{Y}^{LCC} = \text{LCC}(\mathbf{A}, \mathbf{X}, \mathbf{y}), \mathbf{Y}^{GNN} = \text{GNN}(\mathbf{A}, \mathbf{X}, \mathbf{y}).$$

2: **### Calculate model weights ###**
3: Calculate the validation loss of each model using Equations 8 and 9.
4: Calculate temperature-adjusted model weights using Equations 10 and 11.
5: Calculate ensemble output using Equation 12.
6: **return** $\mathbf{Y}^{GNN+LCC}$

integration does not need additional training for both LCC and GNN, it does not require additional training costs.

Specifically, after independently training both LCC and GNN on the training set, we determine the importance weights of LCC and GNN using validation loss. The reason we use the validation loss is to prevent overfitting to the training data. The losses $\mathcal{L}_{LCC}$ and $\mathcal{L}_{GNN}$ of each model are calculated as the cross-entropy loss $\mathcal{L}_{CE}$ between the predictions $\hat{y}_v^{LCC}$, $\hat{y}_v^{GNN}$ and the ground-truth label y_v in the validation set.

$$\mathcal{L}_{LCC} = \sum_{v \in V_{val}} \mathcal{L}_{CE}(\hat{y}_v^{LCC}, y_v), \tag{8}$$

$$\mathcal{L}_{GNN} = \sum_{v \in V_{val}} \mathcal{L}_{CE}(\hat{y}_v^{GNN}, y_v), \tag{9}$$

Since models with lower validation loss are considered more reliable for node classification, we compute the importance weights w_{LCC} and w_{GNN} as the reciprocal of the validation loss as follows:

$$w_{LCC} = \frac{\exp\left(\frac{1}{\mathcal{L}_{LCC}} \cdot \frac{1}{T}\right)}{\exp\left(\frac{1}{\mathcal{L}_{LCC}} \cdot \frac{1}{T}\right) + \exp\left(\frac{1}{\mathcal{L}_{GNN}} \cdot \frac{1}{T}\right)}, \tag{10}$$

$$w_{GNN} = 1 - w_{LCC}, \tag{11}$$

where T is a temperature parameter to adjust the importance weights.

Finally, the prediction $\mathbf{Y}^{GNN+LCC}$ is computed by weighting the predictions $\mathbf{Y}^{LCC}$ and $\mathbf{Y}^{GNN}$ of the two models with their respective importance weights.

$$\mathbf{Y}^{GNN+LCC} = w_{LCC} \cdot \mathbf{Y}^{LCC} + w_{GNN} \cdot \mathbf{Y}^{GNN} \tag{12}$$

In this way, we can integrate LCC and any GNN without additional training and effectively complement their limitations. Algorithm 2 presents this integration algorithm.

Table 2: The statistics of datasets

dataset	#nodes	#edges	#attributes	#class	edge homophily
Cornell	183	298	1,703	5	0.131
Texas	183	325	1,703	5	0.108
Wisconsin	251	515	1,703	5	0.196
Chameleon	2,277	36,101	2,325	5	0.235
Squirrel	5,201	217,073	2,089	5	0.224
Roman-Empire	22,662	44,363	300	18	0.044
Amazon-Ratings	24,492	113,276	300	5	0.382

6 Evaluation Experiments

We evaluate our proposal, the integration of GNN and LCC (GNN+LCC), to answer the following four questions:

Q1: Does GNN+LCC contribute to improving the accuracy of existing GNN designed for heterophilous graphs?
Q2: Are label walks effective for node classification?
Q3: Which label walk types are important for node classification?
Q4: Does the length of the label walk affect performance?

Our code can be found at https://github.com/TakahashiTakutooo/GNN_LCC.

6.1 Experimental Setup

Datasets. We use seven heterophilous directed graph datasets: Cornell, Texas, Wisconsin, Chameleon, Squirrel, Roman-Empire, and Amazon-Ratings [14, 15]. Cornell, Texas, and Wisconsin represent category-based connections in university web pages. Chameleon and Squirrel are networks focused on specific topics from Wikipedia. Roman-Empire represents word connections in Wikipedia articles about the Roman Empire, while Amazon-Ratings represents product connections frequently purchased together on Amazon. Table 2 shows the statistics of the datasets.
Data Splitting. We split each dataset into training, validation, and test sets using five different split methods provided by PyG[2]. For each split method, we perform node classification and compute the average accuracy. The data split ratios are as follows: For Cornell, Texas, Wisconsin, Chameleon, and Squirrel, the train/validation/test split is $48\%/32\%/20\%$. For Roman-Empire and Amazon-Ratings, the split is $50\%/25\%/25\%$.
Hyperparameters. For our proposed method, we perform a grid search using the validation set to determine the hyperparameters: label walk length, number of label walks, label context embedding dimension, and temperature parameter. The search ranges for each parameter are as follows: label walk length: {1, 2, 3}, number of label walks: {3, 5, 7}, {8, 16, 32}, temperature parameter: {0.01, 0.02, . . . , 0.09, 0.1, 0.2, . . . , 1.0}.

[2] https://github.com/pyg-team/pytorch_geometric

Table 3: Node classification accuracy (%) on the test data. For GNN+LCC (H2GCN+LCC, LINKX+LCC, GloGNN+LCC), the accuracy differences (Δ) from the respective GNN model and LCC are also indicated. The highest accuracy for each dataset is highlighted in bold.

	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Roman-Empire	Amazon-Ratings
GCN	43.24 ±4.8	56.22 ±4.6	53.33 ±4.8	62.76 ±0.50	44.36 ±2.7	47.24 ±0.84	45.92 ±0.96
GAT	46.49 ±10	53.51 ±11	48.24 ±4.4	65.22 ±2.3	42.56 ±5.9	55.34 ±1.1	44.75 ±0.97
H2GCN	74.05 ±4.3	80.54 ±7.5	80.00 ±2.8	67.46 ±2.0	55.41 ±2.0	79.09 ±0.65	45.58 ±0.13
LINKX	66.49 ±9.1	64.32 ±5.5	79.22 ±6.1	63.99 ±0.7	58.94 ±2.6	52.35 ±0.71	53.70 ±1.8
GloGNN	75.14 ±3.1	77.84 ±4.6	**85.1 ±2.3**	68.2 ±1.5	58.48 ±0.78	58.24 ±0.87	49.94 ±0.44
LCC	73.51 ±5.7	77.84 ±3.1	78.43 ±5.4	56.01 ±1.6	44.36 ±1.4	81.05 ±0.44	52.17 ±0.48
H2GCN+LCC	**78.92 ±2.0**	**84.32 ±3.1**	83.53 ±4.7	68.25 ±1.2	55.14 ±1.0	**84.10 ±0.41**	52.09 ±0.60
ΔH2GCN	**+4.87**	**+3.78**	**+3.53**	**+0.79**	-0.27	**+5.01**	**+6.51**
ΔLCC	**+5.41**	**+6.48**	**+5.10**	**+12.24**	**+10.78**	**+3.05**	-0.08
LINKX+LCC	76.76 ±6.0	77.84 ±3.9	78.82 ±3.3	66.93 ±2.0	**61.56 ±1.0**	81.11 ±0.15	**57.03 ±0.28**
ΔLINKX	**+10.27**	**+13.52**	-0.40	**+2.94**	**+2.62**	**+28.76**	**+3.33**
ΔLCC	**+3.25**	0.00	**+0.39**	**+10.92**	**+17.20**	**+0.06**	**+4.86**
GloGNN+LCC	76.76 ±3.2	81.62 ±4.6	**85.1 ±2.9**	**69.96 ±2.1**	61.31 ±1.4	80.56 ±0.3	54.42 ±0.24
ΔGloGNN	**+1.62**	**+3.78**	0.00	**+1.76**	**+2.83**	**+22.32**	**+4.48**
ΔLCC	**+3.25**	**+3.78**	**+6.67**	**+13.95**	**+16.95**	-0.49	**+2.25**

6.2 Experimental Results

Q1: Does GNN+LCC contribute to improving the accuracy of existing GNN designed for heterophilous graphs? To evaluate whether LCC enhances existing GNNs for heterophilous graphs, we integrate LCC with H2GCN, LINKX, and GloGNN (denoted as H2GCN+LCC, LINKX+LCC, and GloGNN+LCC) and compare their performance. Additionally, we compare with standard GNN baselines, GCN and GAT.

Overall. Table 3 shows the experimental results. The most important observation is that the highest node classification accuracy is achieved by one of GNN+LCC for all datasets. In addition, integrating LCC and GNN actually improves accuracy compared to each model alone in most cases (see ΔH2GCN, ΔLINKX, ΔGloGNN, ΔLCC). This result confirms that 1) LCC and GNN capture different graph properties and complement each other, and 2) our integration scheme effectively learns the importance weights of both LCC and GNN.

Analysis for exceptional cases. We also observe that there are only four exceptional cases out of 21 cases. GNN alone achieves slightly higher accuracy than GNN+LCC in two cases, **case1**: H2GCN+LCC for the Squirrel dataset (H2GCN is 0.27 higher) and **case2**: LINKX+LCC for the Wisconsin dataset (LINKX is 0.40 higher). Also, LCC alone achieves slightly higher accuracy than GNN+LCC in two cases, **case3**: H2GCN+LCC for the Amazon-Ratings dataset (LCC is 0.08 higher) and **case4**: GloGNN+LCC for the Roman-Empire dataset (LCC is 0.49 higher). The three cases above can be explained by investigating the importance weights between GNN and LCC: the integration does not work well when one of the model weights is extremely high. Ta-

Table 4: The importance weights of each model and temperature parameter in H2GCN+LCC, optimized using grid search.

	LCC weight	H2GCN weight	temperature parameter
Cornell	0.587	0.413	0.20
Texas	0.482	0.518	0.70
Wisconsin	0.632	0.368	0.30
Chameleon	0.102	0.898	0.09
Squirrel	0.084	0.916	0.08
Roman-Empire	0.502	0.498	1.00
Amazon-Rationgs	0.925	0.075	0.02

Table 5: Node classification accuracy (%) on the test data for vanilla MLP and LCC. The higher accuracy for each dataset is highlighted in bold, and the accuracy difference between LCC and vanilla MLP is indicated as $gain$.

	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Roman-Empire	Amazon-Ratings
MLP	$72.97_{\pm 4.5}$	$76.22_{\pm 5.7}$	$77.25_{\pm 3.6}$	$44.04_{\pm 2.7}$	$31.24_{\pm 1.3}$	$64.98_{\pm 0.23}$	$41.00_{\pm 0.41}$
LCC	$\mathbf{73.51_{\pm 5.8}}$	$\mathbf{77.84_{\pm 3.2}}$	$\mathbf{78.43_{\pm 5.4}}$	$\mathbf{56.01_{\pm 1.6}}$	$\mathbf{44.36_{\pm 1.5}}$	$\mathbf{81.05_{\pm 0.45}}$	$\mathbf{52.17_{\pm 0.48}}$
$gain$	$\mathbf{+0.54}$	$\mathbf{+1.62}$	$\mathbf{+1.18}$	$\mathbf{+11.97}$	$\mathbf{+13.12}$	$\mathbf{+16.07}$	$\mathbf{+11.17}$

ble 4 shows the learned importance weights of LCC and H2GCN. The H2GCN weight (0.916) for the Squirrel dataset (**case1**) and the LCC weight (0.925) for the Amazon-Ratings dataset (**case3**) are extremely high. We observe the same phenomena for the LCC weight (0.881) for the Roman-Empire dataset (**case4**). This result implies that there is room for further revision of the weight control mechanism.

Importance weights of the two models in integration. Table 4 shows the importance weights and temperature parameters for H2GCN+LCC. We omit the results for LINKX+LCC and GloGNN+LCC due to space limitations. Overall, the importance weights are relatively balanced across datasets, except for the Chameleon, Squirrel, and Amazon-Ratings datasets. These results confirm the effectiveness of our weight control mechanism for the LCC and GNN integration. As examples of imbalanced weights, for the Chameleon and Squirrel datasets, H2GCN significantly outperforms LCC (see Table 3), resulting in very high H2GCN weights (0.898, 0.916, respectively). Similarly, for the Amazon-Ratings dataset, LCC significantly outperforms H2GCN, leading to a very high LCC weight (0.925).

Q2: Are label walks effective for node classification? To verify the effectiveness of using label context embeddings, we compare the accuracy between a vanilla MLP and LCC. The MLP takes only node features as input whereas LCC additionally takes label context embeddings obtained from four different label walks.

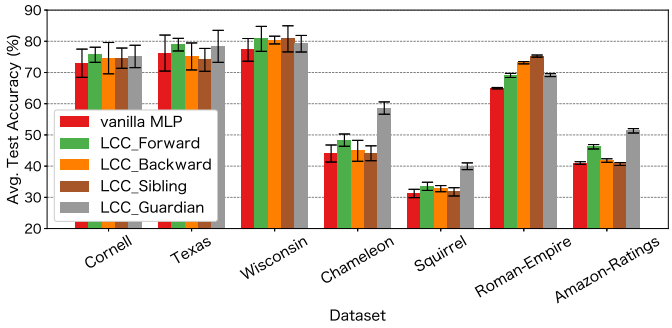


Fig. 4: Node classification accuracy of LCC variations using each label walk type.

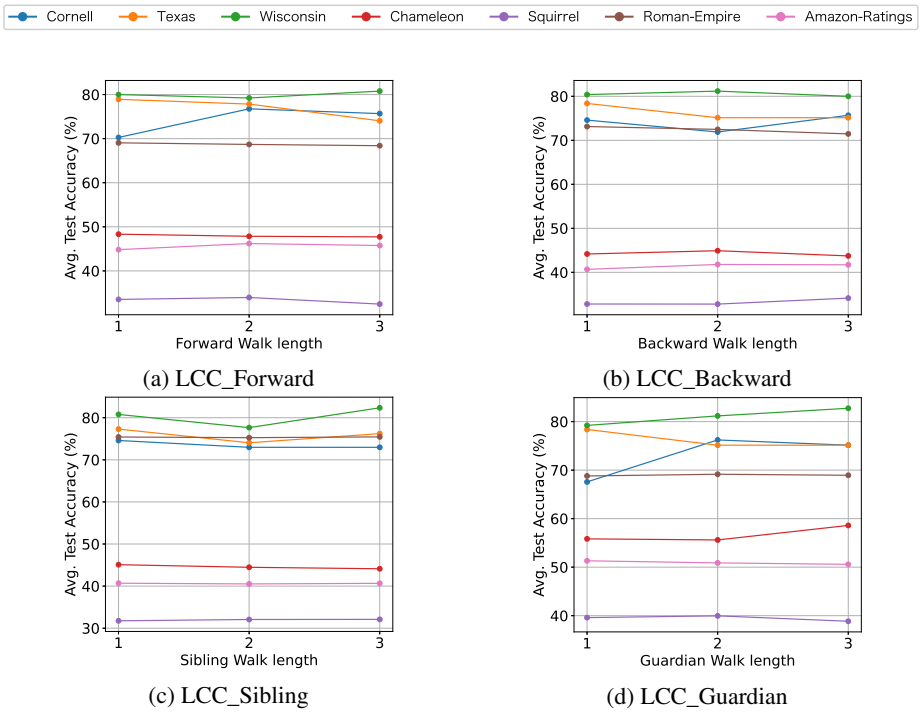
Table 5 shows that LCC outperforms the vanilla MLP in all datasets. This result confirms that the label context embeddings successfully capture class label connectivity and the label walks contribute to improving the node classification accuracy. In particular, the accuracy is improved by more than 10% for the Chameleon, Squirrel, Roman-Empire, and Amazon-Ratings datasets. In particular, the accuracy improvement is significant for the Roman-Empire dataset, more than 16%, which may be due to the large number of classes and the low homophily ratio of the dataset.

Q3: Which label walk types are important for node classification? To evaluate the importance of each of the four types of label walks, we compare LCC variations which concatenate node features with the label context embedding generated from a single type of label walk: LCC_Forward (using forward walks), LCC_Backward (using backward walks), LCC_Sibling (using sibling walks), and LCC_Guardian (using guardian walks). We also compare with the vanilla MLP as a baseline.

Figure 4 shows that the most important label walk varies across datasets. It is quite interesting to observe that, for the Chameleon, Squirrel, and Amazon-Ratings datasets, the guardian walks contribute the most to improving accuracy and the gain is quite significant. Since the guardian walk captures class label connectivity between guardian nodes, we conjecture that these datasets exhibit strong such connectivity. In contrast, for the Roman-Empire dataset, the sibling walks contribute the most to improving accuracy. The result confirms the significant effectiveness of using different label walks, particularly the guardian walk and sibling walk.

Q4: Does the length of the label walk affect performance? To investigate the impact of label walk lengths, we examine the performance of LCC_Forward, LCC_Backward, LCC_Sibling, and LCC_Guardian by varying the label walk length to 1, 2, and 3.

Overall, the result in Figure 5 indicates that the higher-order class label connectivity is crucial for improving the accuracy, as demonstrated by performance improvements when we increase the walk length for forward/backward walks (the class label connectivity order is walk length) or we use sibling/guardian walks (the class label connectivity order is 2). Also, the trends vary across datasets and the type of label walks, highlighting



(a) LCC_Forward (b) LCC_Backward

(c) LCC_Sibling (d) LCC_Guardian

Fig. 5: Node classification accuracy (y-axis) when we change label walk length (x-axis).

the importance of tuning the label walk length as a hyperparameter using a validation set.

For forward walks in the Cornell dataset, the accuracy is significantly improved when increasing the walk length from 1 to 2 and 3, while in the Texas dataset, the accuracy is decreased significantly when the walk length is increased to 3. For backward walks in the Squirrel and Amazon-Ratings datasets, the accuracy is improved with longer walk lengths, whereas the accuracy is decreased for the Texas and Roman-Empire datasets. For sibling walks, the accuracy of Wisconsin drops significantly when increasing the walk length from 1 to 2 but improves again when the walk length is increased to 3. For guardian walks, the accuracy improves with longer walk lengths in the Cornell, Wisconsin, and Chameleon datasets.

7 Conclusion

In this paper, we focused on the fact that conventional GNNs are designed in a way that does not effectively utilize structural information between class labels. The proposed method consists of two key components: (1) the development of a node classifier, LCC, which estimates the class label of a target node based on the neighboring class labels that constitute directed label walks, and (2) the integration of LCC with GNN, enabling

both models to complement each other by capturing different graph characteristics. The experiments demonstrated that the proposed method improves node classification accuracy across seven heterophilous graphs.

Acknowledgments. This work was supported by JSPS KAKENHI Grant Numbers JP20H00583 and JP25H01117 and JST ASPIRE Grant Number JPMJAP2328.

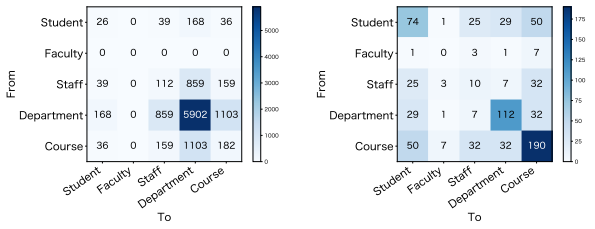
References

1. Chen, J., Chen, S., Gao, J., Huang, Z., Zhang, J., Pu, J.: Exploiting neighbor effect: Conv-agnostic gnn framework for graphs with heterophily. IEEE Transactions on Neural Networks and Learning Systems (2024)
2. Dong, Y., Chawla, N.V., Swami, A.: metapath2vec: Scalable representation learning for heterogeneous networks. In: KDD (2017)
3. Duvenaud, D.K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.P.: Convolutional networks on graphs for learning molecular fingerprints. In: NeurIPS (2015)
4. Gasteiger, J., Bojchevski, A., Günnemann, S.: Combining neural networks with personalized pagerank for classification on graphs. In: ICLR (2019)
5. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: KDD (2016)
6. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: NeurIPS (2017)
7. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017)
8. Li, X., Zhu, R., Cheng, Y., Shan, C., Luo, S., Li, D., Qian, W.: Finding global homophily in graph neural networks when meeting heterophily. In: ICML (2022)
9. Lim, D., Hohne, F.M., Li, X., Huang, S.L., Gupta, V., Bhalerao, O.P., Lim, S.N.: Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. In: NeurIPS (2021)
10. Luan, S., Hua, C., Lu, Q., Zhu, J., Zhao, M., Zhang, S., Chang, X.W., Precup, D.: Revisiting heterophily for graph neural networks. In: NeurIPS (2022)
11. Maekawa, S., Noda, K., Sasaki, Y., et al.: Beyond real-world benchmark datasets: An empirical study of node classification with gnns. NeurIPS (2022)
12. Maekawa, S., Sasaki, Y., Onizuka, M.: A simple and scalable graph neural network for large directed graphs. arXiv preprint arXiv:2306.08274 (2023)
13. Mikolov, T., Chen, K., Corrado, G.S., Dean, J.: Efficient estimation of word representations in vector space. In: ICLR (2013)
14. Pei, H., Wei, B., Chang, K.C.C., Lei, Y., Yang, B.: Geom-gcn: Geometric graph convolutional networks. In: ICLR (2020)
15. Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., Prokhorenkova, L.: A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In: ICLR (2023)
16. Reiser, P., Neubert, M., Eberhard, A., Torresi, L., Zhou, C., Shao, C., Metni, H., van Hoesel, C., Schopmans, H., Sommer, T., Friederich, P.: Graph neural networks for materials science and chemistry. Communications Materials **3** (2022)
17. Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., Battaglia, P.: Learning to simulate complex physics with graph networks. In: ICML (2020)
18. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph Attention Networks. In: ICLR (2018)

19. Wan, G., Du, B., Pan, S., Haffari, G.: Reinforcement learning based meta-path discovery in large-scale heterogeneous information networks. In: AAAI (2020)
20. Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., Yu, P.S.: Heterogeneous graph attention network. In: WWW (2019)
21. Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.i., Jegelka, S.: Representation learning on graphs with jumping knowledge networks. In: ICML (2018)
22. Yu, Z., Feng, B., He, D., Wang, Z., Huang, Y., Feng, Z.: Lg-gnn: Local-global adaptive graph neural network for modeling both homophily and heterophily. In: IJCAI (2024)
23. Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V.: Graphsaint: Graph sampling based inductive learning method. In: ICLR (2020)
24. Zhong, Z., Ivanov, S., Pang, J.: Simplifying node classification on heterophilous graphs with compatible label propagation. Trans. Mach. Learn. Res. (2022)
25. Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications. AI Open **1**, 57–81 (2020)
26. Zhu, J., Rossi, R.A., Rao, A., Mai, T., Lipka, N., Ahmed, N.K., Koutra, D.: Graph neural networks with heterophily. In: AAAI (2021)
27. Zhu, J., Yan, Y., Heimann, M., Zhao, L., Akoglu, L., Koutra, D.: Heterophily and graph neural networks: Past, present and future. IEEE Data Eng. Bull. (2023)
28. Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., Koutra, D.: Beyond homophily in graph neural networks: Current limitations and effective designs. In: NeurIPS (2020)

A Analysis of Sibling Walk and Guardian Walk

Figure 6 illustrates the sibling and guardian class label connectivity in the heterophilous directed graph of the Texas dataset. Figure 6 (a) shows the sibling class label connectivity. According to this, Department nodes have strong connectivity to Department nodes, while nodes of other class labels also show relatively strong connectivity to Department nodes. In contrast, Figure 6 (b) shows the guardian class label connectivity, revealing that nodes with Student nodes, Department nodes, and Course nodes have strong connectivity to the same class label nodes. This indicates that guardian walk is crucial for the Texas dataset. Furthermore, our experimental results (Q3) corroborate the effectiveness of guardian walk on this dataset.



(a) Sibling class label connectivity (b) Guardian class label connectivity

Fig. 6: The class label connectivity in the Texas dataset. (a) sibling class label connectivity from class label in y-axis to class label in x-axis. (b) guardian class label connectivity from class label in y-axis to class label in x-axis.

We also report the 1st-order connectivity and the sibling and guardian class label connectivity for other datasets, Cornell, Wisconsin, Chameleon, Squirrel, Roman-Empire, and Amazon-Ratings in the Supplementary Material.

B The Best Hyper Parameters of GNN+LCC

For our proposed method, we perform a grid search using the validation set to determine the hyperparameters: label walk length, number of label walks, label context embedding dimension, and temperature parameter. The search ranges for each parameter are as follows: label walk length: {1, 2, 3}, number of label walks: {3, 5, 7}, {8, 16, 32}, temperature parameter: {0.01, 0.02, . . . , 0.09, 0.1, 0.2, . . . , 1.0}. The best hyperparameters for each dataset are reported in Table 6.

Table 6: The best hyperarameters.

(a) Label Walk parameters (Forward, Backward, Sibling, Guardian) of LCC

	Forward			Backward			Sibling		Guardian	
	Len.	Num.	Dim.	Len.	Num.	Dim.	Len.	Dim.	Len.	Dim.
Cornell	3	5	8	1	3	8	1	8	3	8
Texas	1	7	8	2	7	8	2	16	1	8
Wisconsin	3	5	32	1	5	16	1	16	1	16
Chameleon	1	5	8	2	3	8	3	8	3	16
Squirrel	1	3	8	2	7	16	1	32	2	32
Roman-Empire	1	3	32	1	3	16	2	16	2	16
Amazon-Ratings	2	5	8	2	7	8	1	16	1	8

(b) Best temperature parameters for GNN + LCC models

	H2GCN + LCC	LINKX + LCC	GloGNN + LCC
Cornell	0.2	1.0	1.0
Texas	0.7	0.3	1.0
Wisconsin	0.3	1.0	1.0
Chameleon	0.09	0.05	0.1
Squirrel	0.08	0.1	0.2
Roman-Empire	1.0	0.09	0.2
Amazon-Ratings	0.02	0.2	0.1