DATA QUALITY-AWARE GRAPH MACHINE LEARNING

By

Yu Wang

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

August 9, 2024

Nashville, Tennessee

To my parents and family for their forever support and love.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Tyler Derr, for his unwavering guidance, encouragement, and support throughout my Ph.D. His mentorship has been invaluable, teaching me how to identify significant research problems, write compelling papers, and design effective presentations. Dr. Derr has also imparted essential skills in efficiently managing a research lab and mentoring students. His tireless efforts to provide me with numerous opportunities and learning experiences have been instrumental in my development as an independent researcher. I am honored to have been his student and will always cherish his advice and friendship, which profoundly influenced my professional and personal life.

I extend my deepest gratitude to my committee members—Dr. Mark Ellingham, Dr. Nitesh Chawla, Dr. Ryan Rossi, Dr. Soheil Kolouri, and Dr. Xenofon Koutsoukos—for their invaluable insights and suggestions. Dr. Ellingham introduced me to graph theory and nonlinear optimization in 2019/2020, sparking a profound interest that has significantly shaped my research in graph machine learning. At the WSDM'22 Doctoral Forum, Dr. Chawla offered essential advice on tackling technical challenges and emphasized the importance of deep engagement with coding frameworks when reviewing papers. His expertise in imbalanced learning has inspired my focus on imbalanced graph machine learning. Dr. Kolouri has been a pivotal mentor in geometric deep learning, enhancing my approach to complex problems with his rigorous logic and elegant mathematical derivations. The weekly machine-learning seminars he hosted with Dr. Derr have broadened my perspectives and inspired much of my research. While I have not had the chance to collaborate directly with Dr. Koutsoukos, I deeply appreciate his senior advice on intersecting graph machine learning and cybersecurity as well as his support as a department chair for my academic hunting. My internship with Dr. Rossi at Adobe was incredibly enriching; his keen insights into research trends have greatly influenced my research flavor, motivating me to drive my findings toward real-world applications. I am particularly grateful for the time he sacrificed during the Independence holiday, working late to help refine my project objectives, which was crucial to its ultimate success.

I joined the Network and Data Science (NDS) Lab at the beginning of the Spring 2021 semester as Dr. Derr's first Ph.D. student when he was establishing the lab. During my Ph.D. study, I have had the pleasure and fortune of having supportive and encouraging friends and colleagues. I am thankful to all my collaborators inside/outside the NDS Lab, especially: Dr. Charu Aggarwal, Dr. Neil Shah, Dr. Danai Koutra, Dr. Jundong Li, Dr. Jens Meiler, Dr. Rocco Moretti, Dr. Nesreen K. Ahmed, Dr. Franck Dernoncourt, Dr. Pamela Wisniewski, Dr. Namyong Park, Dr. Tong Zhao, Dr. Benedek Rozemberczki, Dr. Hiba Baroud, Dr. Wei Jin, Dr. Huiyuan Chen, Yushun Dong, Song Wang, Dr. Jinzhu Yu, Dr. Yue Hu, Dr. Yanbing Wang, Ao Qu, Yuying Zhao, Lance Liu, Xueqi Cheng, Bo Ni, Xin(Allen) Wang. I look forward to continued collaboration with you.

I was fortunate to work as an intern at Adobe Research and The Home Depot with amazing colleagues and mentors: Dr. Tong Sun, Dr. Ryan Rossi, Dr. Nedim Lipka, Dr. Ruiyi Zhang, Dr. Alexa Siu, Dr. Zichao Wang, Dr. Huiming Qu, Dr. Xiquan Cui, Dr. Amin Javari, Dr. Walid Shalaby, Dr. Gromit Chan, Dr. Chang Xiao, Yanzhe Zhang, Jianyi Zhang, Zhendong Chu, Yu Fu, Shengcao Cao, Jiaqi Wang, Wanrong Zhu, Raymond Fok, Yuchen Zhuang, Xu Ma, Yizhou Wang, Hang Hua, Ramraj Chandradevan. I enjoyed the wonderful and productive years with you.

I would also like to extend my heartfelt appreciation to my dear friends at Vanderbilt who provided unwavering support and encouragement during the moments when I felt frustrated during my Ph.D. study. Dr. Jinzhu Yu, Dr. Ruoyu Wang, Dr. Chunlei Su, Dr. Kan Li, Dr. Zezhou Yang, Yuzhi Lin, Dr. Chuchuan Hong, Dr. Damin Xia, Minya Na, Dr. Qi Yang, Dr. Lee Ho Hin, Dr. Riqiang Gao, Dr. Yuchen Tang, Xin Yu, Dr. Quan Liu, Dr. Zimu Su, Ruining Deng, Tianyuan Yao, Xiaohan Wang, Yuxiang Gao, Fangfei Liu, Dr. Siyuan Yu, Yayan Zhao.

Finally, I would like to thank my parents, Qin Yu and Dahong Wang, as well as my entire family for their forever love and support. This dissertation is dedicated to them!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Motivation and Contribution

Recent years have witnessed a significant shift from just model-centric Artificial Intelligence (AI), which focuses on developing top-performing models, to data-centric AI, which emphasizes the quality and quantity of the data in AI models. Concurrently, graph machine learning (GML), such as Graph Neural Networks (GNNs) has emerged as a de-facto approach for graph-structured data by fusing the topological information via message-passing and the feature information via neural transformation. Despite its unprecedented success, its strong dependence on node features and graph topology also makes it vulnerable to data quality issues, which would catastrophically impair GML performance. On the one hand, graphs, like many other data modalities, suffer from conventional data-quality issues, e.g., imbalance. On the other hand, the intrinsic complexity of graphs can exacerbate the aforementioned issues and bring up new ones. For example, the imbalance issue that happens at the quantitative level could also occur at the topological level. The inherent bias encoded in the sensitive feature might be amplified by social interactions. These issues severely impair downstream task performance and are challenging to diagnose due to the inherent complexity of graphs.

Given the criticality of the graph-data quality issues in compromising GML performance and also the ubiquity of GML in real-world applications, my research strives to **establish the Data Quality-Aware Graph Machine Learning framework, which identifies data-quality issues on graph-structured data, diagnoses the problems of existing GML methods when facing data-quality issues, proposes practical solutions from the model/data-centric perspective to mitigate their negative impacts and customizes the developed Data Quality-Aware GML framework into real-world social-good applications.** In Figure 1.1, I systematically study the graph data-quality issues and propose solutions in the following four perspectives:

- (1) **Topology** [1, 2, 3, 4, 5]: issues caused by incorporating the graph topology/structure in machine learning, e.g., heterophily topology, varying local topology, and missing topology issues.
- (2) **Imbalance** [6, 7, 8]: issues caused by quantitative imbalance of the training data on graphs, e.g., node-level and graph-level imbalance issues.
- (3) **Bias** [4, 6, 9]: issues caused by bias/unfairness of the training data on graphs, e.g., social interaction bias and degree-related bias.
- (4) **Limited Data** [10, 11, 12]: issues caused by limited data, e.g., limited training and prompting data.

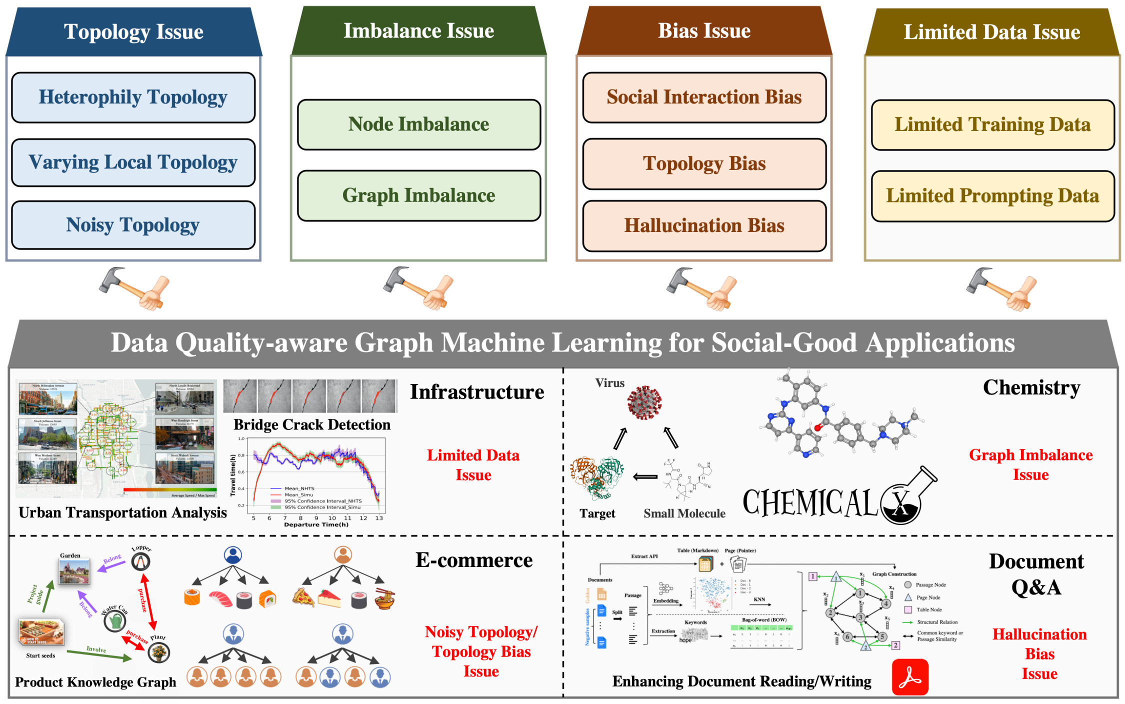


Figure 1.1: A comprehensive overview of my research contributions in Data Quality-aware Graph Machine Learning, including discovering and handling Topology, Imbalance, Bias, and Limited Data Issues. The developed framework has been used for real-world social-good applications, including infrastructure, chemistry, E-commerce, and Documental reading.

Moreover, I have tailored the well-developed Data Quality-Aware Graph Machine Learning for real-world social-good applications, including molecular classification in drug discovery, information retrieval for e-commerce platforms, interdependency analysis of infrastructure networks, and question-answering for document reading.

1.2 Dissertation Organization

The structure of the dissertation is as follows:

- **Chapters 2-5**: Graph topology acts as a double-edged sword in influencing GML performance. While the topology can provide additional gleaning patterns to benefit graph-based tasks, it may also compromise the quality of the learned node representations when misapplied in unsuitable scenarios. Specifically, these four chapters investigate 1) the heterophily topology issue in node classification - how graph machine learning models behave worse in networks where neighboring nodes are different and we propose a tree-decomposition to selectively aggregate neighboring information from different layers; 2) the varying local topology issue in link prediction - how different local subgraphs cause different nodes to have different link prediction/recommendation performance, and we propose a neighborhood-aware denoising mechanism to filter out the outlier neighbors in the aggregation; 3) the noisy topology issue in session recommendation - how the noisy graph connections misalign with the real-world context and we propose a context-aware neighborhood aggregation to filter out the outlier neighbors.

- **Chapters 6-7**: Imbalance in real-world data is widespread across various domains (e.g., chemistry/social), manifesting in diverse formats (e.g., quantity/topology), and exhibits different graph granularity (e.g., nodes/graphs/edges). Consequently, my research focuses on handling these three granularities of imbalance respectively, including 1) imbalance node classification - how the imbalance training labels would influence the node classification and we propose a prototypical training mechanism along with imbalanced label propagation to handle this issue) and 2) imbalance graph classification - how the imbalance labeled graphs would influence the graph classification and we propose a graph-of-graph propagation and structural augmentation to handle this issue).

- **Chapters 8-10**: Bias in real-world data presents significant social risks, such as discrimination in critical decision-making systems and misinformation from large language models. My research addresses three main categories of biases: 1) Social Interaction Bias - We explore how graph machine learning is affected by interaction biases and propose an adaptive masking strategy to mitigate these effects. 2) Degree-Related Bias - We analyze the inherent biases in current evaluation algorithms for link prediction and recommender systems, particularly how they favor users with varying degrees of connectivity. 3) Hallucination Bias - We address the challenge of ensuring the reliability of content generated by large language models and propose the integration of knowledge graphs to improve content quality.

- **Chapters 11**: Recent success has demonstrated the significance of data in powering the machine learning model. However, in real-world cases, collecting numerous well-curated data is extremely time-consuming and resource-extensive. This chapter focuses on handling these limited data issues in graph generation. More specifically, we collect numerous well-curated graphs from diverse domains and train a large graph generative model for graph generation. We demonstrate this large-scale training paradigm with sufficient high-quality data would enable our model to learn fundamental transferable patterns across different domains.

- **Chapter 12**: This chapter summarizes the dissertation by overviewing the four different categories of data quality issues as well as their corresponding solutions we have investigated. Following that, we preview the future directions to explore.

CHAPTER 2

Topology Issue: Overcoming the Heterophily Topology in Node Classification

Graph Neural Networks (GNNs) have achieved significant success in learning better representations by performing feature propagation and transformation iteratively to leverage neighborhood information. Nevertheless, iterative propagation restricts the information of higher-layer neighborhoods to be transported through and fused with the lower-layer neighborhoods', which unavoidably results in feature smoothing between neighborhoods in different layers and can thus compromise the performance on heterophily networks. In this work, we first theoretically analyze the feature smoothing between neighborhoods in different layers and empirically demonstrate the variance of the homophily level across neighborhoods at different layers. Motivated by these analyses, we further propose a tree decomposition method to disentangle neighborhoods in different layers to alleviate feature smoothing among these layers. Moreover, we characterize the multi-hop dependency via graph diffusion within our tree decomposition formulation to construct Tree Decomposed Graph Neural Network (TDGNN), which can flexibly incorporate information from large receptive fields and aggregate this information utilizing the multi-hop dependency. Comprehensive experiments demonstrate the superior performance of TDGNN on both homophily and heterophily networks under a variety of node classification settings.[1].

2.1 Introduction

Graph representation learning has recently emerged as a powerful strategy for node classification [13, 14, 15, 16], graph classification [14, 17, 18, 19] and link prediction [20, 21] on graph-structured data. As the generalization of deep learning to the graph domain, Graph Neural Networks (GNNs) have become one of the most promising paradigms [22], which adopts a message-passing scheme to learn node representations by utilizing both the node features and the graph topology [15, 16, 23]. A typical GNN architecture for node classification consists of two stages: message-passing and neural transformation. Firstly, messages are propagated from neighboring nodes to their corresponding center nodes and then aggregated together via pooling operations such as average pooling, min/max pooling, and learnable pooling. Afterward, the neural transformation layer transforms the aggregated messages to extract useful node representations. These two stages are packed together and termed as one layer of graph convolution. Deep GNNs iteratively perform multiple graph convolutions to obtain a larger receptive field and thus incorporate information of neighborhoods in higher layers [24, 25, 26, 27].

[1]https://dl.acm.org/doi/abs/10.1145/3459637.3482487

Although GNNs have gained significant achievements, most of existing works have been focused on homophily networks where neighboring nodes possess similar feature distributions or belong to the same class. In this way, propagating neighboring embeddings would benefit the prediction of the center nodes. Despite the success of homophily networks, a common challenge faced by GNNs is node classification on heterophily networks where neighboring nodes do not necessarily share similar features or possess the same class label. In this case, blindly aggregating neighborhood information to the center nodes as what most popular models do, such as GCN and GAT, would, unfortunately, mix information from different classes together and essentially hurt the performance of downstream node classification. Stepping further, several methods propose deep GNNs to incorporate higher-layer neighborhood information through iterative propagation [23, 27, 28, 29]. However, most of them are focused on handling over-smoothing issues and the incorporated higher-layer neighborhood information may still come from different feature/class distributions, leaving the heterophily issue unaddressed.

Given the challenge that features of higher-layer neighborhoods are over-smoothed with the lower-layer neighborhoods and noticing the potential negative impact of this over-smoothing on heterophile networks, we propose an effective framework, termed Tree Decomposed Graph Neural Network (TDGNN), to learn node representations from larger receptive fields without causing feature over-smoothing between different layers of neighborhoods and allow flexible layer configurations to avoid under-performance on heterophily networks. Our major contributions are listed as follows:

- Motivated by our theoretical analysis on feature smoothing and empirical demonstration of the variance of the homophily level across neighborhoods in different layers, we propose a tree decomposition method to disentangle features of neighborhoods in different layers, which can help alleviate the problem of feature smoothing and provides more flexible layer configurations for complex networks.
- We capture and maintain the importance of multi-hop dependency in learning better representations within our tree decomposition method by characterizing this multi-hop dependency by graph diffusion, which ultimately leads to the construction of the proposed Tree Decomposed Graph Neural Network (TDGNN).
- We conduct experiments in both semi-supervised and full-supervised settings and on both homophily and heterophily network datasets to comprehensively demonstrate the superiority of our proposed TDGNN framework over existing methods. Additionally, we perform a parameter analysis to better understand and contrast TDGNN to prior GNNs.

2.2 Related Work

Deep Graph Neural Networks. Deep GNNs are related to over-smoothing and primarily aim to incorporate higher-layer neighborhood information through iterative propagation. For example, SGC [23] and S^2GC [27] attempt to capture higher-layer neighborhood information by applying K^{th} power of the graph convolution in a single neural network layer. APPNP [29] replaces the power of the graph convolution with the Personalized PageRank [30] and GDC [28] further extends APPNP by generalizing Personalized PageRank to an arbitrary graph diffusion process. More recently, methods such as GCNII [24] and DAGNN [26], aggregate embeddings at different layers and outperform all previously mentioned deep GNNs.

GNNs on Heterophily Networks. Heterophily has recently been raised as an important issue since it breaks the traditional network homophily assumption that is widely adopted in many GNNs. More specifically, in a heterophily network, the concept that linked nodes are likely from different classes or have dissimilar features is initially recognized within the context of GNNs in [31]. Zhu et al. [32] proposes a set of effective designs that allow GNNs to generalize to challenging heterophily settings, and Chen et al. [24] leverages initial residual connection and identity mapping to enable GCN to express a K^{th} order polynomial filter with arbitrary coefficients. In comparison, our work demonstrates that the poor performance of GNNs on heterophily networks is caused by feature smoothing between neighborhoods in different layers. By decomposing the computational tree of center nodes and increasing the depth of the GNNs, we can selectively devise suitable layer configurations to boost the model performance on a heterophily network.

2.3 Methodology

In this section, we design a Tree Decomposed Graph Neural Network (TDGNN) by mainly solving the two challenges mentioned in Section 2.1, which are feature smoothing between neighborhoods in different layers and lack of consideration of the multi-hop dependency in GNNs. For the first challenge, we theoretically show the feature smoothing between different layers when applying iterative propagation and further propose a tree decomposition method to disentangle neighborhood information in different layers. For the second challenge, we formalize the definition of the multi-hop dependency and characterize it through a graph diffusion process. Combining the tree decomposition method to disentangle the neighborhood information on different layers and the graph diffusion to model the multi-hop dependency, we propose TDGNN. We also introduce two mechanisms to aggregate node representations of each layer: TDGNN-s, which directly sums the representations of all layers together, and TDGNN-w, which assigns learnable weights and adaptively combines the node representations of each layer. The framework is shown in Figure 2.3, which has three main components: tree decomposition to handle feature smoothing between different neighborhood layers, graph diffusion to model multi-hop dependency, and aggregation to combine representations of different layers.

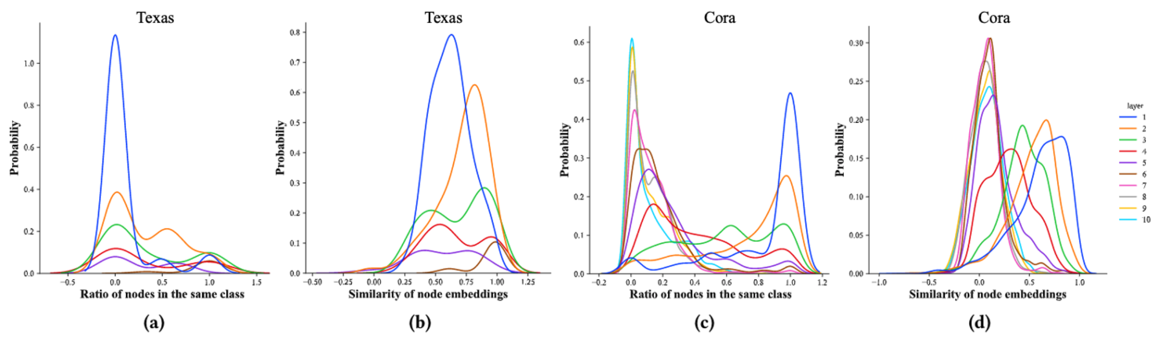


Figure 2.1: Visualizing the variance of homophily across neighborhoods at different levels according to the distribution of the ratio of different layer neighborhoods in the same class as their corresponding center nodes (i.e., (a) and (c)) and the cosine similarity of their embeddings (obtained from feeding node features through only the transformation layers of a pre-trained 2-layer GCN) to their center nodes (i.e., (b) and (d)) for the Texas and Cora datasets.

2.3.1 Tree Decomposition

The basic assumption in GNNs is that the neighborhood information of the center node leveraged by applying feature propagation and aggregation can enhance the prediction performance of the center node itself [33]. Such an assumption is justified by the core network property, homophily, where linked nodes tend to share similar features and typically belong to the same class [31, 34]. However, the level of homophily might be completely different among different networks or even among different subgraphs within the same network. One extreme situation would be the heterophily network where linked nodes are likely from different classes or have dissimilar features [32, 35].

In Figure 2.1, we show the level of homophily across different neighborhood layers in the Cora and Texas datasets. More specifically, the level of homophily is measured by distributions of the ratio of neighborhoods $\mathcal{N}_i^k$ in different neighborhood layers k that share the same class as (in Figure 2.1(a) and Figure 2.1(c)) and have similar embeddings to (in Figure 2.1(b) and Figure 2.1(d)) their corresponding center node v_i, which are obtained from feeding node features only through the transformation layers in a pre-trained 2-layer GCN [15] without introducing any bias from feature propagation. In the Cora dataset, it can be observed that the majority of neighbors among their 1st-layer neighborhoods have the same class as their corresponding center nodes, but the number of center nodes that have most of their neighborhoods sharing the same class as themselves decreases as the layer increases. Furthermore, the embeddings of these neighborhoods on the 1st-layer, on average, have high similarity to their corresponding center nodes. This demonstrates the high homophily of the Cora dataset on low layers and propagating features of nodes in these low layers fuse embeddings of nodes in the same class and thus make embeddings of different classes more separable. However, even for this extreme homophily layer in this strong homophily Cora dataset [36], not all of the nodes in the 1st-layer have their neighborhoods sharing the same label with themselves, and this strong homophily becomes progressively weaker as we reach further out to higher layers. For example, only around 10% of the nodes in the 3rd-layer have all their neighborhoods sharing the same class and over half of the nodes in the 10th-layer have nearly all neighborhoods different from themselves. Even worse, in the Texas dataset, even for low layers, most of the neighborhoods have different classes from their center nodes, especially for nodes on the 1st-layer, almost all nodes belong to different classes from their center nodes, which demonstrates the strong heterophily of the Texas dataset. Propagating features of nodes in such low layers to their center nodes fuses embeddings of nodes in different classes and makes those nodes indistinguishable, which results in learning worse node representations. Such feature smoothing among different layers is unavoidable as long as the procedure of iterative propagation is taken, that is: *during iterative feature propagation, the information of neighborhoods in higher layers has to be transported through and fused with the information of neighborhoods in lower layers and then propagated to their corresponding center nodes.*

If we take the most popular GNN-variant, a 2-layer GCN, as an example (for simplicity), then after 2-layer graph convolutions, the representation of the node v_i is:

$$\begin{aligned}\mathbf{h}_i^2 = & \underbrace{\sigma(\mathbf{h}_i^0\mathbf{W}^0)\mathbf{W}^1\Big(\frac{1}{(d_i+1)^2} + \sum_{j\in\mathcal{N}_i^1}\frac{1}{(d_i+1)(d_j+1)}\Big)}_{0^{\text{th}}\text{-layer features (self)}} + \underbrace{\sum_{j\in\mathcal{N}_i^1}\sigma(\mathbf{h}_j^0\mathbf{W}^0)\mathbf{W}^1\Big(\frac{d_i+d_j+2}{(d_i+1)^{1.5}(d_j+1)^{1.5}}\Big)}_{1^{\text{st}}\text{-layer neighborhood features}} \\ & + \underbrace{\sum_{j\in\mathcal{N}_i^1}\sum_{k\in\{\mathcal{N}_j^1\cap\mathcal{N}_i^1\}}\sigma(\mathbf{h}_k^0\mathbf{W}^0)\mathbf{W}^1\frac{1}{\sqrt{d_i+1}\sqrt{d_k+1}(d_j+1)}}_{1^{\text{st}}\text{-layer neighborhood features}} \\ & + \underbrace{\sum_{j\in\mathcal{N}_i^1}\sum_{k\in\{\mathcal{N}_j^1\cap\mathcal{N}_i^2\}}\sigma(\mathbf{h}_k^0\mathbf{W}^0)\mathbf{W}^1\frac{1}{\sqrt{d_i+1}\sqrt{d_k+1}(d_j+1)}}_{2^{\text{nd}}\text{-layer neighborhood features}}, \end{aligned} \tag{2.1}$$

which consists of three components corresponding to the feature information of neighborhoods in the 0th, 1st, and 2nd layers. Eq. (2.1) intuitively shows that the embedding of node v_i after two iterative graph convolutions contains the information of both 1st and 2nd- layer neighborhood information, which will compromise the performance on Texas dataset since the feature and class information of the 1st-layer is of great difference from their corresponding center nodes according to Figure 2.1(a) and Figure 2.1(b). Even for the Cora dataset, where the feature and class information of the 1st-layer neighborhoods is similar to their corresponding center nodes, still some center nodes have neighborhoods in the 1st-layer different from themselves and for these nodes, incorporating their neighborhood information might compromise their predictions.

Thus, based on our analysis, to advance the frontier of GNNs to be able to selectively leverage neighborhood information in different layers, we propose a tree decomposition method. More specifically, our proposed method disentangles neighborhoods in different layers and connects them directly with their corresponding center nodes. These direct connections allow the propagation of higher-layer neighborhoods' information to their corresponding center nodes without any interference from lower-layer neighborhoods along the way. Furthermore, this tree decomposition procedure enables more flexible layer configurations of neighborhoods. For example in Figure 2.2, we decompose the computational tree in GNNs of the center node v_1. Then, in the training process, we selectively propagate information of nodes in different layers: propagating along the 1st-layer subgraph, the 2nd-layer subgraph, and both of these two subgraphs. The choice depends on the network homophily of different layers and is determined by hyperparameter-tuning. The adjacency matrix of the kth-layer subgraph $\mathbf{T}^k$ obtained from tree decomposition can be computed by the difference between corresponding powers of the normalized adjacency matrices with added self-loops and formalized as follows:

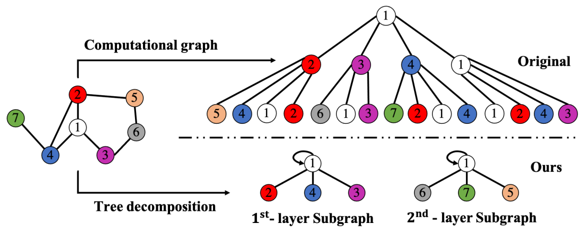


Figure 2.2: Tree decomposition of the center node v_1 in the given graph to two layers compared to the computational graph in the original GNNs (e.g., GCN).

$$\mathbf{T}^k = \text{sign}(\widehat{\mathbf{A}}^k) - \text{sign}(\widehat{\mathbf{A}}^{k-1}) + \mathbf{I}, \tag{2.2}$$

$$\text{sign}(\widehat{\mathbf{A}}^k)_{ij} = \begin{cases} 1, & \text{if } \widehat{\mathbf{A}}^k_{ij} > 0 \\ 0, & \text{if } \widehat{\mathbf{A}}^k_{ij} = 0, \end{cases} \tag{2.3}$$

where $\widehat{\mathbf{A}}^0 = \mathbf{I}$ is the identity matrix and $\widehat{\mathbf{A}} = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}$ is the renormalized adjacency matrix as previously defined. The equivalence between $\mathbf{T}^k$ and the kth-layer subgraph, including the self-loop, can be easily proven, so we omit the details for brevity.

2.3.2 Multi-hop Dependency

Although the tree decomposition could avoid the issue of feature smoothing between different layers, we also lose the multi-hop dependency captured by the original iterative propagation which might cause over-smoothing [25]. Two nodes have multi-hop dependency if they are connected by a path in the network, and specifically, k-hop dependency is defined as two nodes connected by at least one simple path with length k. For example in Figure 2.2, features of node v_2 can not only be propagated along the edge $v_2 \to v_1$ to v_1 but also along the longer path $v_2 \to v_5 \to v_6 \to v_3 \to v_1$ to v_1. However, after tree decomposition, the edge

$v_2 \rightarrow v_1$ is the only way for propagating features of v_2 to v_1. Instead of inserting multiple edges between the higher-layer neighborhood nodes and their corresponding center nodes, we model this multi-hop dependency via graph diffusion [28]. Specifically, $\widehat{\mathbf{A}}^i$ represents the i^{th}-hop dependency and its entry $\widehat{\mathbf{A}}^i_{pq}$ measures the strength of paths of length i in propagating features from node v_p to v_q. Assuming the maximum hop of the dependency we consider is K, since k^{th}-layer ($k \leq K$) neighborhood nodes can only propagate their features along paths of length from k to K, thus the total multi-hop dependencies from node v_p to v_q along these paths is calculated as $\sum_{i=k}^{K} \widehat{\mathbf{A}}^i_{pq}$. Such multi-hop dependency across the spectrum from k to K between a single pair of nodes can be further generalized to all pairs of nodes in the graph via diffusion and defined as:

$$\mathbf{E}^{k,K} = \sum_{i=k}^{K} \widehat{\mathbf{A}}^i, \tag{2.4}$$

where $\mathbf{E}^{k,K}$ considers dependencies from paths of length k to K, which could be used as the edge weights for propagating node features in the k^{th}-layer subgraph obtained from tree decomposition.

2.3.3 Tree Decomposed Graph Neural Network

Now, having motivated and introduced the two major components of our proposed framework, namely the tree decomposition and multi-hop dependency formulations, we collect them together and present our Tree Decomposed Graph Neural Network (TDGNN). As previously noted, an illustration of our proposed TDGNN is shown in Figure 2.3, and its corresponding mathematical formulation is defined as:

$$\mathbf{H}^0 = \text{MLP}(\mathbf{X}), \tag{2.5}$$

$$\mathbf{H}^k = (\mathbf{T}^k \odot \mathbf{E}^{k,K})\mathbf{H}^0, \ k = 1, 2, ..., L, \tag{2.6}$$

$$\mathbf{Z} = \begin{cases} \sum_{k=0}^{L} \mathbf{H}^k, & \text{TDGNN-s} \\ \sum_{k=0}^{L} \theta_k \mathbf{H}^k, & \text{TDGNN-w.} \end{cases} \tag{2.7}$$

We first apply a Multilayer Perceptron (MLP) network to the original feature matrix $\mathbf{X}$ to get the initial representations of nodes $\mathbf{H}^0$ [26, 29]. Then, we decompose the whole network by calculating the adjacency matrix $\mathbf{T}^k$ of k^{th}-layer tree based on Eq. (2.2) and Eq. (2.3). This k^{th}-layer subgraph contains only edges between center nodes and their corresponding k^{th}-layer neighborhood nodes, including a self-loop. Since we consider the neighborhood nodes up to L^{th}-layer, the k is from 1 to L. Next, we utilize graph diffusion to calculate the multi-hop dependency $\mathbf{E}^{k,K}$ based on Eq. (2.4) and K is the predefined maximum hop of dependency we consider. Afterward, we propagate the initial node representations $\mathbf{H}^0$ along edges in each subgraph

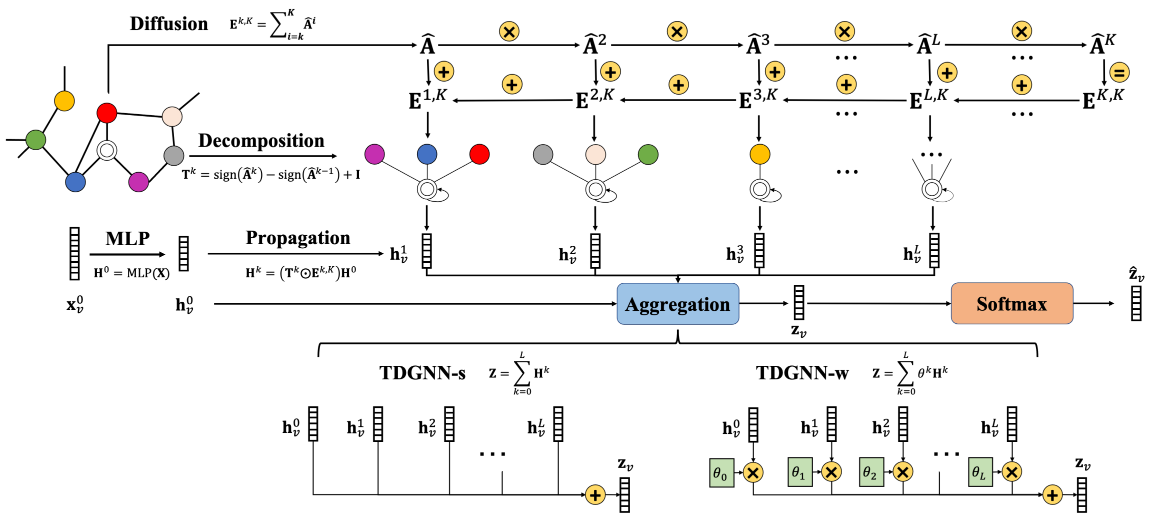

Diffusion
$E^{k,K} = \sum_{i=k}^{K} \hat{A}^i$
Decomposition
$T^k = \mathrm{sign}(\hat{A}^k) - \mathrm{sign}(\hat{A}^{k-1}) + I$
MLP
$H^0 = \mathrm{MLP}(X)$
Propagation
$H^k = (T^k \odot E^{k,K}) H^0$
Aggregation
Softmax
TDGNN-s $Z = \sum_{k=0}^{L} H^k$
TDGNN-w $Z = \sum_{k=0}^{L} \theta^k H^k$

Figure 2.3: An illustration of the proposed Tree Decomposed Graph Neural Network (TDGNN). For brevity, we only present the pipeline of predicting the label of one node.

following each corresponding adjacency matrix $\mathbf{T}^k$ with the corresponding edge weight from multi-hop dependency $\mathbf{E}^{k,K}$ to get representations $\mathbf{H}^k$ for each layer k based on Eq. (2.6). We collect representations from each layer and aggregate them together using two aggregation mechanisms to get the final representations $\mathbf{Z}$ based on Eq. (2.7). The first aggregation mechanism is to directly sum up the representations of all layers together. The second aggregation mechanism is to assign learnable weights and adaptively combine the representations of each layer. The corresponding two versions of our model are termed as TDGNN-s and TDGNN-w, respectively. Ultimately, $\mathbf{Z}$ is employed to compute the cross-entropy loss for labeled nodes as:

$$\mathcal{L} = -\sum_{v_i \in \mathcal{V}_l} \sum_{j=1}^{C} \mathbf{Y}_{ij} \log \widehat{\mathbf{Z}}_{ij}, \tag{2.8}$$

where $\widehat{\mathbf{Z}}$ is the probability distribution of each node belonging to each class and is obtained by applying softmax on the final representation $\mathbf{Z}$. Note that $\mathcal{V}_l \subset \mathcal{V}$ is the set of training nodes with known label information as previously defined and C is the total number of classes to be predicted.

In summary, our model decouples transformation from propagation [26], which enlarges the receptive fields without introducing more trainable parameters. Obtaining low dimensional representations before propagation [29] makes the training process of TDGNN computationally efficient. Additionally, tree decomposition preprocessing allows a more flexible choice of utilizing/combining different layers to propagate features. The multi-hop dependency enables feature propagation along paths of various lengths, which conforms to other recent work [37]. Furthermore, applying learnable weight coefficients equips TDGNN with the ability to flexibly select an effective receptive field based on a specific network.

2.3.4 Complexity Analysis

Compared to vanilla GCN, the additional computational load mostly comes from the tree decomposition and the graph diffusion. Since tracking down the corresponding L-layer subgraphs is equivalent to calculating the difference between corresponding powers of the normalized adjacency matrices with added self-loops by Eq. (2.2), which is exactly given by graph diffusion, the time for the tree decomposition could be saved. The time complexity for performing graph diffusion process is $O(Kn^3) = O(n^3)$ due to K times matrix multiplication and can be reduced approximately to $O(n^{2.81})$ if using the Strassen algorithm [38] or even further to $O(n^{2.38})$ [39]. Moreover, in practice, real-world graphs are extremely sparse, and thus, sparse matrix multiplication methods [40] could also be used to further improve computational efficiency. Notably, both the tree decomposition and the graph diffusion are preprocessing outside of the training, which significantly reduces the computational load of the whole framework.

For the space complexity, the bottleneck would be saving diffusion matrices $\widehat{\mathbf{A}}^k, k = 1, 2, ..., K$, and the adjacency matrices of each subgraph $\mathbf{T}^k, k = 1, 2, ..., K$, which leads to $O(Kn^2)$ and constitutes a severe threat for networks of large scale. However, as we highlighted before and demonstrate in Figure 2.4, since higher-layer neighborhoods may have completely different features from their corresponding center nodes and incorporating their information gains little benefits in learning better embeddings, we could only consider lower-layer neighborhoods [41] and thus keep only the first few adjacency matrices. On the other hand, most of the real-world networks have the small-world property that most nodes can be reached from every other node by a small number of hops [42], which confirms and helps justify why we can remove the higher-layer adjacency matrices. Moreover, we could apply the same strategy as GraphSAGE [43] where we sample nodes from the center node's local neighborhood via random walk and propagate features among these sampled nodes [41]. Since this work focuses on disentangling neighborhoods and characterizing the multi-hop dependency, the aforementioned is left as one future direction.

2.4 Experiment

In this section, we conduct extensive node classification experiments to evaluate the superiority of our proposed TDGNN model. We begin by introducing the datasets and experimental setup we employed. Then, we compare TDGNN with prior baselines and some state-of-the-art (SOTA) deep GNNs.

Table 2.1: Statistics of network datasets for node classification.

Networks		**Nodes**	**Edges**	**Features**	**Classes**	**Train/Val/Test**	**Type**
Homophily	Cora	2708	5429	1433	7	140/500/1000	Citation network
	Citeseer	3327	4732	3703	6	120/500/1000	Citation network
	Pubmed	19717	44338	500	3	60/500/1000	Citation network
Non-homophily	Cornell	183	295	1703	5	48%/32%/20%	Webpage network
	Texas	183	309	1703	5	48%/32%/20%	Webpage network
	Wisconsin	251	499	1703	5	48%/32%/20%	Webpage network
	Actor	7600	33544	931	5	48%/32%/20%	Actor co-occurrence network

2.4.1 Experimental Settings

Datasets. We evaluate the performance of our TDGNN model and baseline models with node classification on multiple real-world datasets. More specifically, we use the three standard citation network datasets Cora, Citeseer, and Pubmed [44] for semi-supervised node classification [45], where nodes correspond to documents associated with the bag-of-words as the features and edges correspond to citations. For full-supervised node classification, in addition to the three citation networks we include three extra web network datasets, Cornell, Texas, and Wisconsion [31], where nodes and edges represent web pages and hyperlinks, and one actor co-occurrence network dataset, Actor [31], where nodes and edges represent actors and their co-occurrence in the same movie. Table 2.1 contains the basic network statistics for each of these datasets.

Baselines. To evaluate the effectiveness of TDGNN, we choose the following representative supervised node classification baselines including SOTA GNN models. **MLP** [46]: 2-layer multilayer perceptron with dropout and ReLU non-linearity, which is empirically shown in other works to perform well on non-homophily network datasets [32]. **GCN** [15]: GCN is one of the most popular graph convolutional models. **GAT** [16]: Graph attention network employs an attention mechanism to pay different levels of attention to nodes within the neighborhood set and is widely used as a GNN baseline. **SGC** [23]: Simple graph convolution network removes nonlinearities and collapsing weight matrices between consecutive layers, which obtains comparable accuracy and yields orders of magnitude speedup over GCN. We note that SGC collapses the traditional GNN aggregation tree such that the center node receives the features directly from the flattened neighborhood while being weighted according to the higher-order neighborhood information. **APPNP** [29]: APPNP links GCN and PageRank to derive an improved propagation scheme based on personalized PageRank, which incorporates higher-order neighborhood information and meanwhile keeps the local information. **Geom-GCN** [31]: Geom-GCN explores to capture long-range dependencies in non-homophily networks. It uses the geometric relationships defined in the latent space to build structural neighborhoods for aggregation. Since Geom-GCN is mainly designed for non-homophily networks, we only report its performance in full-supervised node classification where three non-homophily networks are included. **DAGNN** [26]: Deep adaptive graph neural network first decouples the representation transformation from propagation so that large receptive fields can be applied without suffering from performance degradation. Then, it utilizes an adaptive adjustment mechanism, which adaptively balances the information from local and global neighborhoods for each node. **GCNII** [24]: GCNII employs residual connection to retain part of the information from the previous layer and adds an identity mapping to ensure the non-decreasing performance as the GNN model goes deeper (i.e., successfully adds more layers). For baselines with multiple variants (Geom-GCN, GCNII), we only choose the best for each dataset and denote it as model*.

Parameter Settings. We implement our proposed TDGNN and some necessary baselines using Pytorch [47] and Pytorch Geometric [48], a library for deep learning on graph-structured data built upon Pytorch. For DAGNN[2], and GCNII[3], we use the original code from the authors' GitHub repository. We aim to provide a rigorous and fair comparison between different models on each dataset by tuning hyperparameters for all models individually. The number of hidden units is searched from $\{16, 32, 64, 128\}$, the dropout rate is searched from $\{0, 0.5, 0.8\}$, the weight decay is searched from $[1e^{-4}, 2e^{-2}]$, the training epochs is searched from $\{300, 500, 1000, 1500, 3000, 4000\}$ and the learning rate is set to be 0.01. We find that some baselines even achieve better results than their original reports. Note that in this work, we do not treat the random seed as a hyperparameter and therefore, the random seed fixed in previous models for reproducing results, if any, is reset to be totally random to remove any potential bias and thus allow for more generalized comparison. For reproducibility, codes of all of our models and corresponding hyperparameter configurations for results in Table 2.2-2.3 are publicly available [4].

2.4.2 Semi-supervised Node Classification

For the semi-supervised node classification, we apply the fixed split following [45] and random training/-validation/testing split on Cora, Citeseer, and Pubmed, with 20 nodes per class for training, 500 nodes for validation, and 1000 nodes for testing. For each model, we conduct 100 runs and report the mean classification accuracy with the standard deviation in both the fixed and random splitting cases. Table 2.2 reports the best mean accuracy with the standard deviation over different data splits where the best model per benchmark is highlighted in bold and the number in parentheses corresponds to the neighborhood layers used at which the best performance is achieved. For example, (0-4) means the corresponding performance is achieved when we use the neighborhood of layers up to 4, and 0-layer neighborhoods correspond to using the features of the nodes themselves for prediction. The random split is to remove the positional bias from the training nodes as nodes in the class centers tend to impose more influence and inductivity [49].

[2]https://github.com/vthost/DAGNN
[3]https://github.com/chennnM/GCNII
[4]https://github.com/YuWVandy/TDGNN

Table 2.2: Semi-supervised classification accuracy (%) ± stdev over Cora, Citeseer, and Pubmed datasets. Best and runner-up performances are reported in **bold** and underlined.

Method	Cora		Citeseer		Pubmed		Avg. Rank
	Fixed	Random	Fixed	Random	Fixed	Random	
GCN	81.50±0.79 (0-2)	79.91±1.64 (0-2)	71.42±0.48 (0-2)	68.78±2.01 (0-2)	79.12±0.46 (0-2)	77.84±2.36 (0-2)	7.17
GAT	83.10±0.40 (0-2)	80.80±1.60 (0-2)	70.80±0.50 (0-2)	68.90±1.70 (0-2)	79.10±0.40 (0-2)	77.80±2.10 (0-2)	7.00
SGC	82.63±0.01 (0-2)	80.18±1.57 (0-2)	72.10±0.14 (0-2)	69.33±1.90 (0-2)	79.12±0.10 (0-2)	76.74±2.84 (0-2)	6.83
APPNP	83.34±0.56 (0-10)	82.26±1.39 (0-10)	72.22±0.50 (0-10)	70.53±1.57 (0-10)	80.14±0.24 (0-10)	79.54±2.23 (0-10)	3.83
DAGNN	84.88±0.49 (0-10)	83.47±1.18 (0-10)	73.39±0.57 (0-9)	70.87±1.44 (0-10)	**80.51±0.42 (0-20)**	79.52±2.19 (0-20)	2.33
GCNII*	**85.57±0.45 (0-64)**	82.58±1.68 (0-64)	73.24±0.61 (0-32)	70.04±1.72 (0-10)	80.00±0.48 (0-16)	79.03±1.68 (0-16)	3.83
TDGNN-s	85.35±0.49 (0-4)	**83.84±1.45 (0-6)**	**73.78±0.60 (0-8)**	**71.27±1.71 (0-8)**	80.20±0.33 (0-5)	**80.01±1.96 (0-5)**	**1.33**
TDGNN-w	84.42±0.59 (0-4)	83.43±1.35 (0-6)	72.14±0.49 (0-6)	70.32±1.57 (0-6)	80.12±0.44 (0-5)	79.77±2.04 (0-5)	3.67

Table 2.3: Summary of full-supervised classification accuracy (%) ± standard deviation over 8 datasets. Best and runner-up performances are reported in **bold** and underlined.

Method	Cora	Cite.	Pub.	Corn.	Tex.	Wisc.	Act.	Avg. Rank
MLP	75.78±1.84 (0)	73.81± 1.74 (0)	86.90±0.37 (0)	80.97±6.33 (0)	81.32± 4.19 (0)	85.38±3.95 (0)	36.60±1.25 (0)	5.57
GCN	86.97±1.32 (0-2)	76.37±1.47 (0-2)	88.19±0.48 (0-2)	58.57±3.57 (0-2)	58.68±4.64 (0-2)	53.14±6.25 (0-2)	28.65±1.38 (0-2)	8.14
GAT	87.30±1.01 (0-2)	75.55±1.32 (0-2)	85.33±0.48 (0-2)	61.89±5.05 (0-2)	58.38±6.63 (0-2)	55.29±4.09 (0-2)	28.45±0.89 (0-2)	8.00
SGC	87.07±1.20 (0-2)	76.01±1.78 (0-2)	85.11±0.52 (0-2)	58.68±3.75 (0-2)	60.43±5.11 (0-2)	53.49±5.13 (0-2)	27.46±1.46 (0-2)	8.57
Geom-GCN*	85.35±1.57 (0-2)	**78.02±1.15 (0-2)**	89.95±0.47 (N/A)	60.54±3.67 (0-2)	66.76±2.72 (N/A)	64.51±3.66 (N/A)	31.63±1.15 (N/A)	5.86
APPNP	86.76±1.74 (0-10)	77.08±1.56 (0-10)	88.45±0.42 (0-10)	74.59±5.11 (0-10)	74.30±4.74 (0-10)	81.10±2.93 (0-10)	34.36±1.09 (0-10)	5.43
DAGNN	87.26±1.42 (0-10)	76.47±1.54 (0-10)	87.49±0.63 (0-20)	80.97±6.33 (0)	81.32±4.19 (0)	85.38±3.95 (0)	36.60±1.25 (0)	4.71
GCNII*	**88.27±1.31 (0-64)**	77.06±1.67 (0-64)	**90.26±0.41 (0-64)**	76.70±5.40 (0-16)	77.08±5.84 (0-32)	80.94±4.94 (0-16)	35.18±1.30 (0-64)	3.71
TDGNN-s	88.26±1.32 (0-4)	76.64±1.54 (0-8)	89.13±0.39 (0-1)	80.97±6.33 (0)	82.95±4.59 (0, 4-5)	85.47±3.88 (0, 4-5)	36.70±1.28 (0, 3-4)	2.86
TDGNN-w	88.01±1.32 (0-5)	76.58±1.40 (0-2)	89.22±0.41 (0-1)	**82.92±6.61 (0, 2-6)**	**83.00±4.50 (0, 2)**	**85.57±3.78 (0, 3-5)**	**37.11±0.96 (0, 3-4)**	**2.14**

* We reuse the results in [36] for Geom-GCN. 'N/A' indicates the corresponding layers are not reported.

We observe that TDGNN-s performs the best in terms of the average rank through all datasets and across both random and fixed splits, which suggests the comprehensive superiority of TDGNN-s to other baselines. Specifically, our TDGNN-s model outperforms the representative baselines, including GCN, GAT, SGC, and APPNP, across all datasets by significant margins. Compared with two recent deep GNN models, DAGNN and GCNII*, TDGNN-s can still achieve comparable or even better performance. Especially when the data split is random, TDGNN-s outperforms all other models, demonstrating the strong robustness of TDGNN-s (in terms of dataset splits). It is also worth noting that our TDGNN model achieves the SOTA performance with relatively shallow layers compared with DAGNN and GCNII*. On the Cora dataset, the best performance is achieved when layers are used up to 4 and 6 for our TDGNN-s model, respectively, in fixed and random data splitting. At the same time, DAGNN and GCNII require up to 10 and 64 layers to achieve the best, which demands heavy computation and thus is time inefficient. On the Citeseer dataset, our model also utilizes up to the most shallow layers compared with DAGNN and GCNII* to achieve the SOTA performance. Surprisingly, the weighted version of our model, TDGNN-w, performs poorly than TDGNN-s while still outperforming most of the baselines. This is because the weight coefficients $\{\theta_i\}_{i=0}^{L}$ are only decided by training nodes and the suitable weights for combining aggregated features $\{\mathbf{H}_i\}_{i=0}^{L}$ and getting good predictions on training and validation nodes might not be suitable for testing nodes, which inspires future work for a layer aggregation mechanism that enables node-adaptive layer combination.

2.4.3 Full-supervised Node Classification

For the full-supervised node classification task, we evaluate our TDGNN model and existing GNNs using 7 datasets: Cora, Citeseer, Pubmed, Cornell, Texas, Wisconsin, and Actor. For each dataset, we use 10 random splits (48%/32%/20% of nodes per class for training/validation/testing) from [31]. Note that although [31] reports that the ratios are 60%/20%/20%, this differs from the actual data splits shared on their GitHub [36]. We conduct 100 runs with each split evaluated 10 times and report the mean accuracy with the standard deviation in Table 2.3. Here, the numbers in the parentheses again correspond to layers of neighborhoods utilized (e.g., (0, 3-5) means the corresponding performance is achieved when we use neighborhoods of layers 3 to 5 and 0-layer neighborhoods corresponding to the center nodes themselves.)

First, we observe from Table 2.3 that TDGNN-w has the best average rank across the two types of networks (i.e., homophily and heterophily) with TDGNN-s ranking second. Next, we observe that TDGNN-w significantly outperforms the baselines across the heterophily networks. However, both variants of TDGNN are slightly outperformed on the homophily networks in this full-supervised setting (whereas in most homophily networks under the semi-supervised setting, TDGNN-s performs the best). Thus, to better understand the inner workings of TDGNN, we next perform a detailed parameter analysis.

2.4.4 Sensitivity Analysis

Here, we compare the performance of TDGNN with other baselines when utilizing neighborhoods in different layers. Furthermore, we perform a parameter analysis of TDGNN by varying the neighborhood layers (L) and the multi-hop dependencies (K).

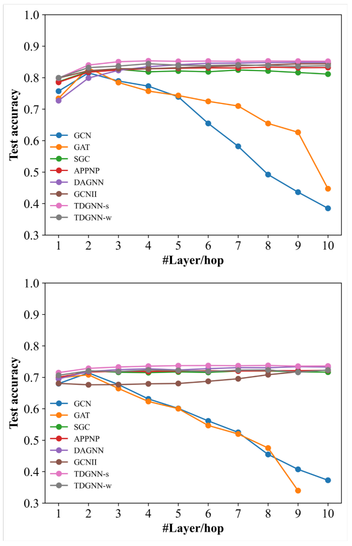
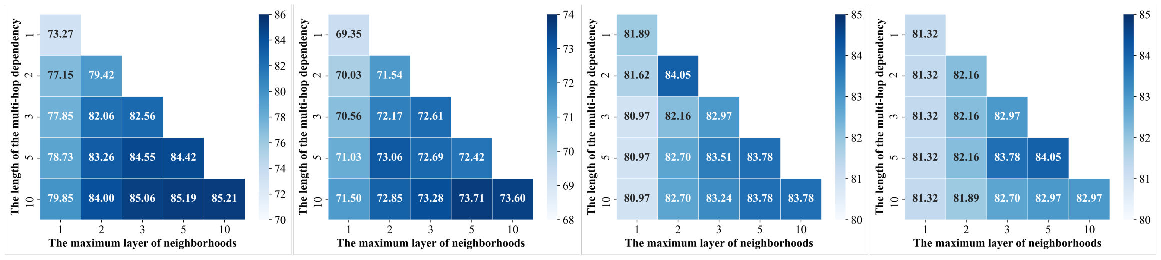


Figure 2.4: Results of models with different layers on **Cora (Top)** and **Citeseer (Bottom)**

First, to demonstrate the strength of the TDGNN-s model in shallow layers, we visualize the performance of each model using layers from up to 1 to up to 10 in Figure 2.4. For the Cora and Citeseer datasets, our model achieves around 84% and 73% using only the first 2-layer neighborhoods and the first three-layer neighborhoods, respectively. Compared to two SOTA deep GNNs where DAGNN achieves the same level of performance using 5 layers and 7 layers, and GCNII* achieves using 8 layers and at least 32 layers, our model can leverage less neighborhood information to achieve comparable performance, which clearly validates the importance of considering multi-hop dependency. To some extent, this raises the concern over whether we need deep GNNs to incorporate higher-layer neighborhood information in homophily networks or if shallow feature information aggregated according to higher-order multi-hop dependencies provides sufficient infor-



(a) Cora (TDGNN-s) **(b) Citeseer (TDGNN-s)** **(c) Cornell (TDGNN-w)** **(d) Texas (TDGNN-w)**

Figure 2.5: Visualizing the effect of varying the maximum layer neighborhoods and the length of multi-hop dependency on the performance of TDGNN.

mation. Besides, the continued high-level performance as model depth increases demonstrates the higher resilience of TDGNN-s against over-smoothing.

Second, we vary the maximum layer of neighborhoods and the multi-hop dependency to study their effect on the performance of the proposed two models: TDGNN-s on two representative homophily networks and TDGNN-w on two representative heterophily networks. Both of the maximum layer of the neighborhoods and the length of the multi-hop dependency are selected from $\{1, 2, 3, 5, 10\}$ due to the small-world theory that two nodes will be connected through few series of intermediaries [42]. Figure 2.5 visualizes the averaged accuracy across 10 runs for various layers and dependency configurations. For two homophily networks, including extra neighborhood layers significantly increase the model performance for lower-layers and such boosting effect becomes progressively weaker as more and more higher-layer neighborhood layers are included, e.g., the performance increases from 79.85 to 84.00 and from 71.50 to 72.85 for Cora and Citeseer when including the 2^{nd}-layer neighborhood while only from 84.00 to 85.06 and from 72.85 to 73.28 when including the 3^{rd}-layer. This weaker boost as the layer number increases is also in line with the decreasing homophily level as observed in Figure 2.1. In comparison, for heterophily networks, in Figures 2.5(c) and 2.5(d) we can observe a more significant need for the decoupling of neighborhood layers since increasing the receptive field (i.e., increasing the maximum layer of neighborhoods) is not always advantageous. Similarly including deeper multi-hop dependencies is not always a clear advantage as seen in the homophily networks because lower-layer neighborhoods that have different labels or representations from their corresponding center nodes may contribute more to their center nodes' prediction through longer dependency. We note that these findings also align with our empirical analysis in Figure 2.1. Therefore, we believe that the increased performance obtained by TDGNN over prior work is partially credited to its ability to separate the concept of graph convolutions in deeper GNNs with higher-layer neighborhoods into both multi-hop dependencies and decoupled neighborhood layers, which can allow any deep GNN model to be more flexibly customized via hyperparameter tuning on a wider variety of complex networks.

2.5 Conclusion

In this chapter, we theoretically analyze the feature smoothing of neighborhoods in different layers and propose a tree decomposition method that disentangles neighborhoods of different layers, thus allowing a more flexible layer configuration. Moreover, our work provides the first theoretical and empirical analysis that unveils the importance of multi-hop dependency in learning better node representations and discloses its connection with graph diffusion. Based on these insights, we design our Tree Decomposed Graph Neural Network (TDGNN) model with two variants, TDGNN-s and TDGNN-w, which simultaneously address the problem of feature smoothing between different layers and incorporate the multi-hop dependency. Extensive experiments demonstrate that TDGNN outperforms representative baselines on a wide range of real-world datasets across network types (including homophily and heterophily) and various node classification task settings.

CHAPTER 3

Topology Issue: Analyzing the Varying Local Topology Issue in Link Prediction

While Graph Neural Networks (GNNs) have shown great power in learning node embeddings for link prediction (LP), no previous works have explored its varying performance across different nodes and its underlying reasons. To this end, we aim to demystify which nodes perform better by analyzing their local topology. Despite the widespread belief that low-degree nodes exhibit poorer LP performance, our empirical findings provide nuances to this viewpoint and prompt us to propose a better metric, Topological Concentration (TC), based on the intersection of the local subgraph of each node with the ones of its neighbors. We empirically demonstrate that TC has a higher correlation with LP performance than other node-level topological metrics, offering a better way to identify low-performing nodes than using degree. With TC, we discover a novel topological distribution shift issue in which newly joined neighbors of a node tend to become less interactive with that node's existing neighbors, compromising the generalizability of node embeddings for LP at testing time. To make the computation of TC scalable, we further propose Approximated Topological Concentration (ATC) and justify its efficacy in approximating TC and reducing the computation complexity.[1].

3.1 Introduction

Recent years have witnessed unprecedented success in applying link prediction (LP) in real-world applications [50, 51, 52]. Compared with heuristic-based [53, 54] and shallow embedding-based LP approaches [55, 56], GNN-based ones [21, 57] have achieved state-of-the-art performance; these methods first learn node/-subgraph embeddings by linear transformations with message-passing and a decoder/pooling layer to predict link scores/subgraph class. While existing works boost overall LP performance [58, 59] by more expressive message-passing or data augmentation, it is heavily under-explored whether different nodes within a graph would obtain embeddings of different quality and have varying LP performance.

Previous works have explored GNNs' varying performance on nodes within a graph, considering factors like local topology [60, 61], feature quality [62], and class quantity [63]. While these studies have provided significant insights, their focus has primarily remained on node/graph-level tasks, leaving the realm of LP unexplored. A more profound examination of the node-varying LP performance can enhance our comprehension of network dynamics [54], facilitate the timely detection of nodes with ill-topology [64], and pave the way for customized data-driven strategies to elevate specific nodes' LP performance [65]. Given the criticality of studying the varying LP performance and the apparent gap in the existing literature, we ask:

[1] https://openreview.net/forum?id=apA6SSXx2e

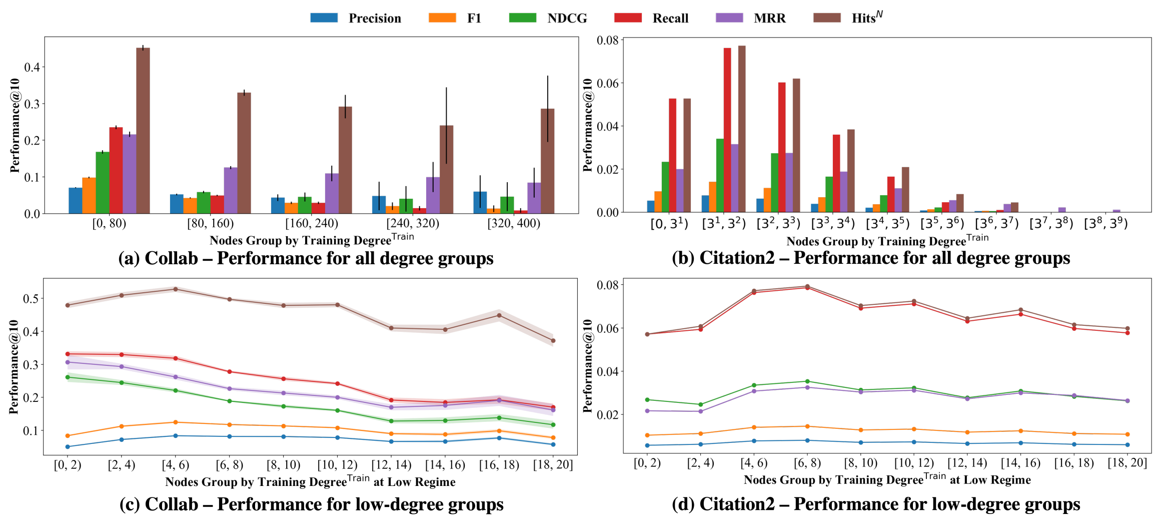


Figure 3.1: Average LP performance of nodes across different degree groups based on Degree$^{\text{Train}}$(i.e., node degree by training edges) on Collab/Citation2. In **(a)-(b)**, Performance@10 does not increase as the node degree increases. In **(c)-(d)**, few/lower-degree nodes do not perform worse than higher-degree counterparts.

Can we propose a metric that measures GNNs' varying LP performance across different nodes?

To answer a related question in the node classification task, prior works observed that GNNs perform better on high-degree nodes than low-degree nodes [60, 66]. Similarly, the persistent sparse topology issue in the general LP domain and recommender systems [67, 68] indicates that nodes with zero-to-low degrees lag behind their high-degree counterparts. However, as surprisingly shown in Figure 3.1(a)-(b), GNN-based LP on these two large-scale social networks does not exhibit a consistent performance trend as the node degree increases. For example, the performance@10 on Collab under all evaluation metrics decreases as the node degree increases, while on Citation2, performance@10 first increases and then decreases. This counter-intuitive observation indicates the weak correlation between the node degree and LP performance, which motivates us to design a more correlated metric to answer the above question.

Following [21] that the link formation between each pair of nodes depends on the interaction between their local subgraphs, we probe the relation between the local subgraphs around each node (i.e., its computation tree) and its GNN-based LP performance. Specifically, we propose Topological Concentration (TC) to measure the topological interaction between the local subgraph of each node and the local subgraphs of the neighbors of that node. Our empirical observations show that TC offers a superior characterization of node LP performance in GNNs, leading to 82.10% more correlation with LP performance and roughly 200% increase in the performance gap between the identified under-performed nodes and their counterparts than degree. Moreover, we discover a novel topological distribution shift (TDS) in which newly joined neighbors of a node tend to become less interactive with that node's existing neighbors. Our contributions are as follows:

- We propose Topological Concentration (TC) and demonstrate it leads to 82.10% more correlation with LP performance and roughly 200% increase in the performance gap between the identified under-performed nodes and their counterparts than node degree, shedding new insights on degree-related issues in LP. We further propose Approximated Topological Concentration (ATC) and demonstrate it maintains high correlations to the LP performance similar to TC while significantly reducing the computation complexity.
- We uncover a novel Topological Distribution Shift (TDS) issue according to TC and demonstrate its negative impact at the node/graph level for link prediction at the testing time. Moreover, we discover that different nodes within the same graph can have varying amounts of TDS.

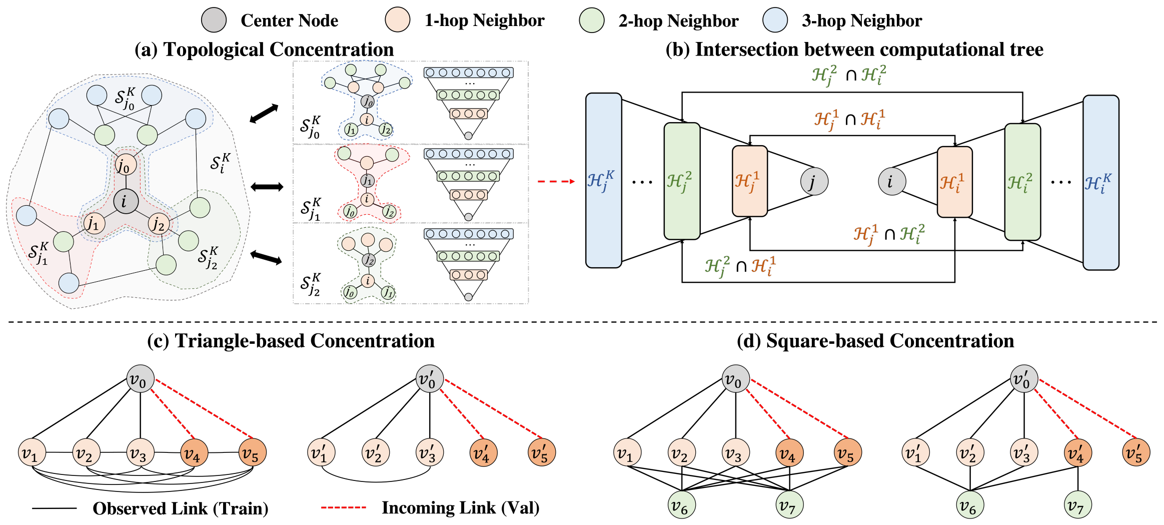
3.2 Related Work

Varying Performance of GNNs on Node/Graph Classification. GNNs' efficacy in classification differs across nodes/graphs with varying label quantity [7, 63] and varying topology quality [32, 60, 61, 66]. To enhance GNNs' performance for the disadvantaged nodes/graphs in these two varying conditions, previous works either apply data augmentations to derive additional supervision [69, 70] or design expressive graph convolutions to mitigate structural bias [71]. However, none tackle the varying performance of nodes in LP. We fill this gap by studying the relationship between node LP performance and its local topology.

GNN-based LP. GNN-based LP works by first learning node embeddings/subgraph embeddings through linear transformation and message-passing, and then applying the scoring function to predict link probability/subgraph class [21, 72]. It has achieved new SOTA performance owing to using the neural network to extract task-related information and the message-passing to encode the topological properties (e.g., common neighbors) [57, 73].

Varying Performance of GNNs on LP. As LP nowadays has been heavily used to enhance user experience in social/e-commerce recommendations [74, 75], studying its varying performance across different users has real-world applications such as identify users with ill-topology and take augmentation strategies. Although no efforts have been investigated into the node-varying performance in GNN-based LP, prior work [68, 76] have investigated the relation of node-varying LP performance with its degree, and both claimed that users/nodes with higher activity levels/degrees tend to possess better recommendation performance than their less active counterparts, which also aligns with observations in GNN-based node classification [60, 66]. However, Figure 3.1(c)-(d) has already raised concern over the validity of this claim in LP. we follow [4] and theoretically discover that some node-centric evaluation metrics have degree-related bias in Appendix 9.4, implying that the GNNs' varying LP performance could be partially attributed to the choice of evaluation metrics. To mitigate this bias, we employ a full spectrum of metrics and find that degree is not so correlated with the node LP performance. This motivates us to devise a better topological metric than the degree.

3.3 Topological Concentration



Center Node
1-hop Neighbor
2-hop Neighbor
3-hop Neighbor
(a) Topological Concentration
(b) Intersection between computational tree
(c) Triangle-based Concentration
(d) Square-based Concentration
Observed Link (Train)
Incoming Link (Val)

Figure 3.2: **(a)-(b)**: v_i's Topological Concentration: we calculate the average intersection between v_i's computation tree and each of v_i's neighbor's computation tree. The intersection between two computation trees is the ratio of the observed intersections to all possible intersections. **(c)-(d)**: two specifications of TC, corresponding to social and e-commerce networks. A higher triangle/square-based concentration indicates more triangles/squares are formed among v_0's local subgraph.

3.3.1 Topological Concentration: Intuition and Formalization

As the link formation between a node pair heavily depends on the intersection between their local subgraphs [21, 57], we similarly hypothesize the predictability of a node's neighbors relates to the intersection between this node's subgraph and the subgraphs of that node's neighbors, e.g., the prediction of the links $\{(i, j_k)\}_{k=0}^{2}$ in Figure 3.2(a) depends on the intersection between K-hop computational tree centered on $v_i, \mathcal{S}_i^K$ and K-hop computational trees centered on the neighbors of v_i, $\{\mathcal{S}_{j_k}^K\}_{k=0}^{2}$. A higher intersection leads to higher LP performance. For example, in Figure 3.2(c)-(d), v_0 neighbors closely interact with themselves while v_0' neighbors do not, posing different topological conditions for the LP on v_0 and v_0'. From graph heuristics perspective, v_0 shares common neighbors v_1, v_2, v_3 with its incoming validation neighbors v_4, v_5 while v_0' shares no neighbors with v_4', v_5'. From the message-passing perspective, the propagated embeddings of v_0 and v_4, v_5 share common components since they all aggregate $\{v_k\}_{k=1}^{3}$ embeddings while v_0' and v_4', v_5' do not share any common embeddings among $\{v_k'\}_{k=1}^{3}$. When the subgraph (i.e., computation tree) surrounding a node increasingly overlaps with the subgraphs of its neighbors, more paths originating from that node are likely to loop nearby and eventually return to it, resulting in a more dense/concentrated local topology for that node. Inspired by this observation, we introduce *Topological Concentration* to measure the average level of intersection among these local subgraphs as follows:

Definition 1. ***Topological Concentration (TC):*** *The Topological Concentration $C_i^{K,t}$ for node $v_i \in \mathcal{V}$ is defined as the average intersection between v_i's K-hop computation tree ($\mathcal{S}_i^K$) and the computation trees of each of v_i's type t neighbors:*

$$C_i^{K,t} = \mathbb{E}_{v_j \sim \mathcal{N}_i^t} I(\mathcal{S}_i^K, \mathcal{S}_j^K) = \mathbb{E}_{v_j \sim \mathcal{N}_i^t} \frac{\sum_{k_1=1}^{K} \sum_{k_2=1}^{K} \beta^{k_1+k_2-2} |\mathcal{H}_i^{k_1} \cap \mathcal{H}_j^{k_2}|}{\sum_{k_1=1}^{K} \sum_{k_2=1}^{K} \beta^{k_1+k_2-2} g(|\mathcal{H}_i^{k_1}|, |\mathcal{H}_j^{k_2}|)} \tag{3.1}$$

$\forall v_i \in \mathcal{V}, \forall t \in \mathcal{T}$, where $I(\mathcal{S}_i^K, \mathcal{S}_j^K)$ quantifies the intersection between the K-hop computation trees around v_i and v_j, and is decomposed into the ratio of the observed intersections $|\mathcal{H}_i^{k_1} \cap \mathcal{H}_j^{k_2}|$ to the total possible intersections $g(\mathcal{H}_i^{k_1}, \mathcal{H}_j^{k_2})$ between neighbors that are k_1 and k_2 hops away as shown in Figure 3.2(b). $\beta^{k_1+k_2-2}$ accounts for the exponential discounting effect as the hop increases. The normalization term g is a function of the size of the computation trees of node v_i, v_j [77]. Although computation trees only consist of edges from the training set, v_i's neighbors $\mathcal{N}_i^t$ in Eq. (3.1) could come from training/validation/testing sets, and we term the corresponding TC as TC$^{\text{Train}}$, TC$^{\text{Val}}$, TC$^{\text{Test}}$ and their values as $C_i^{K,\text{Train}}, C_i^{K,\text{Val}}, C_i^{K,\text{Test}}$. We verify the correlation between TC and the node LP performance in Section 3.3.2.

3.3.2 Topological Concentration: Observation and Analysis

In this section, we draw three empirical observations to delve into the role of TC in GNN-based LP. For all experiments, we evaluate datasets with only the topology information using LightGCN and those also having node features using GCN/SAGE [15, 78, 79][2].

Obs. 1. TC correlates to LP performance more than other node topological properties. In Figure 3.3 (a)/(d), we group nodes in Collab/Citation2 based on their TC$^{\text{Train}}$ and visualize the average performance of each group. Unlike Figure 3.1(a)/(b), where there is no apparent relationship between the performance and the node degree, the performance almost monotonically increases as the node TC$^{\text{Train}}$ increases regardless of the evaluation metrics. This demonstrates the capability of TC$^{\text{Train}}$ in characterizing the quality of nodes' local topology for their LP performance. Moreover, we quantitatively compare the Pearson Correlation of the node LP performance with TC$^{\text{Train}}$ and other commonly used node local topological properties, Degree$^{\text{Train}}$ (i.e., the number of training edges incident to a node) and SubGraph Density (i.e., the density of the 1-hop training subgraph centering around a node). As shown in Figure 3.3(b)/(c), TC$^{\text{Train}}$ almost achieves the highest Pearson Correlation with the node LP performance across every evaluation metric than the other two topological properties except for the precision metric. This is due to the degree-related evaluation bias implicitly encoded in the precision metric, i.e., even for the untrained link predictor, the precision of a node still increases linearly as its degree increases, as proved in Theorem 8. Note that the node's 1-hop Subgraph Density equals its local clustering coefficient (LCC), and one previous work [80] has observed its correlation

[2]Due to GPU memory limitation, we choose SAGE for Citation2.

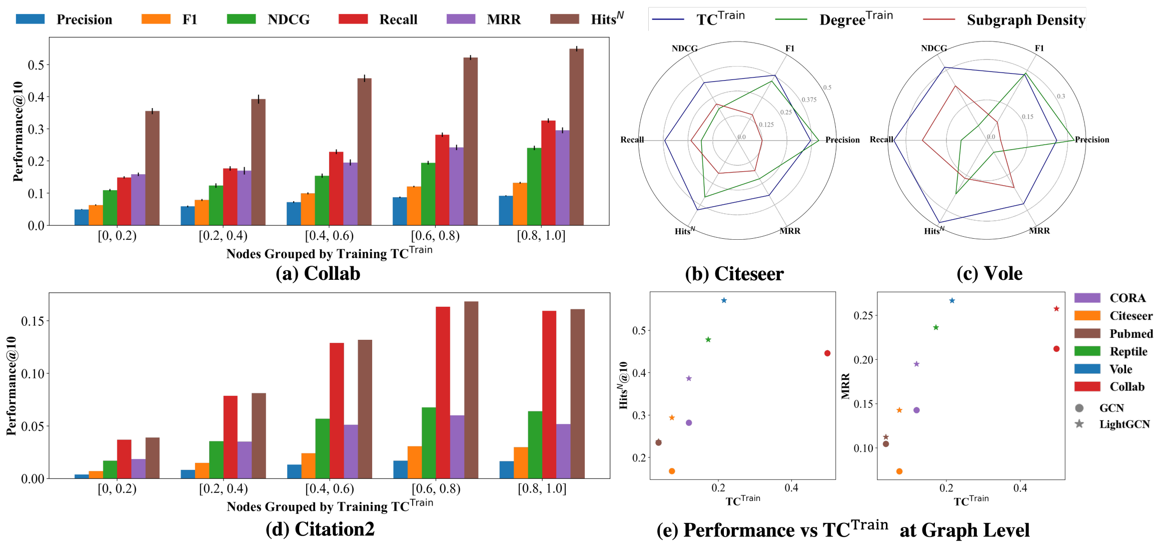


Figure 3.3: **(a)/(d)**: The average LP Performance of nodes on Collab/Citation2 monotonically increases as the TC$^{\text{Train}}$ increases. **(b)/(c)**: TC$^{\text{Train}}$ mostly achieves the highest Pearson Correlation with LP performance on Citeseer/Vole than Degree$^{\text{Train}}$ and Subgraph Density metrics. **(e)**: LP performance is positively correlated to TC$^{\text{Train}}$ across different network datasets.

with node LP performance. Additionally, Figure 3.3(e) shows that TC$^{\text{Train}}$ also positively correlated with LP performance across various networks, depicting a preliminary benchmark for GNNs' LP performance at the graph level [81]. The LightGCN architecture exhibits a steeper slope than GCN, as it relies exclusively on network topology without leveraging node features and thus is more sensitive to changes in the purely topological metric, TC. The deviation of Collab under both GCN and LightGCN baselines from the primary linear trend might be attributed to the duplicated edges in the network creating the illusion of a higher TC$^{\text{Train}}$ [82].

Obs. 2. TC better identifies low-performing nodes than degree, and lower-degree nodes may not necessarily have lower LP performance. As previously shown in Figure 3.1(c)/(d), when the node degree is at the very low regime, we do not observe a strict positive relationship between node Degree$^{\text{Train}}$ and its LP performance. For example, the node Recall/MRR/NDCG@10 in Collab decreases as Degree$^{\text{Train}}$ increases and Hits^N/F1/Precision@10 first increases and then decreases. These contradicting observations facilitate our hypothesis that the degree might not fully capture the local topology in characterizing the underperforming nodes. Conversely, in Figure 3.4(a)/(d) on Collab/Citation2, nodes with lower TC$^{\text{Train}}$ almost always have worse LP performance under all evaluation metrics except when TC$^{\text{Train}}$ is between $[0, 0.02)$. For this extreme case, we ascribe it to the distribution shift as nodes with extremely low TC$^{\text{Train}}$ generally have a decent TC$^{\text{Test}}$ and sustain a reasonable LP performance. We thoroughly investigate this distribution shift issue in **Obs. 3**. Furthermore, we adjust the Degree$^{\text{Train}}$ from 1 to 10 to group nodes into 'Lower-degree/Higher-degree' and adjust TC$^{\text{Train}}$ from 0.01 to 0.1 to group nodes into 'Concentrated/Non-Concentrated'. We compare their

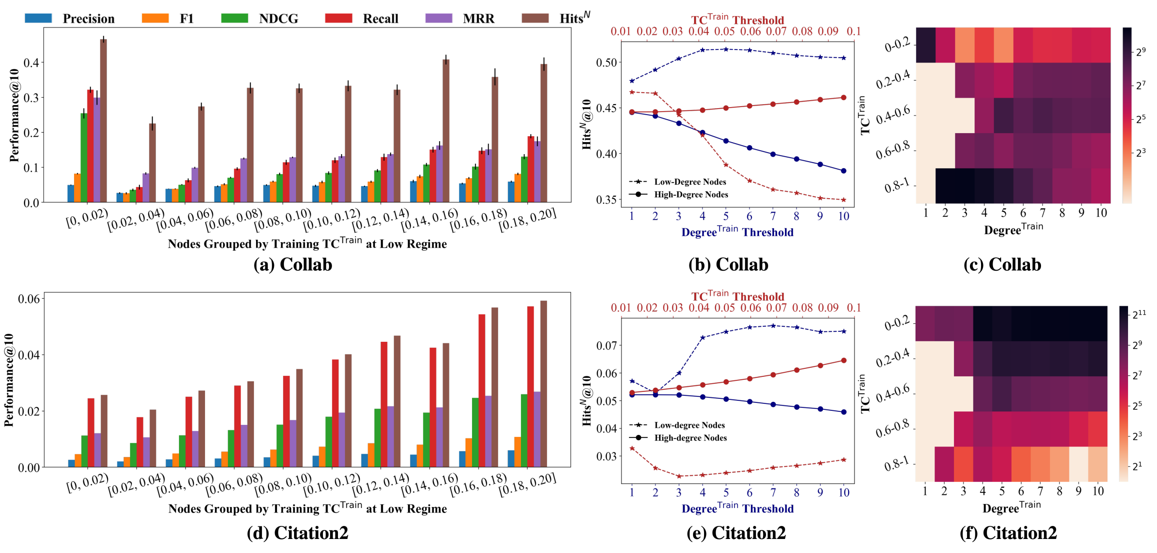


Figure 3.4: **(a)/(d)**: The average LP performance of nodes with extremely low TC$^{\text{Train}}$ on Collab/Citation2 almost monotonically increases as TC$^{\text{Train}}$ increases. **(b)/(e)**: Nodes with lower Degree$^{\text{Train}}$ surprisingly perform better than their higher degree counterparts (**Blue** curves). In contrast, Non-concentrated nodes identified by owning lower TC$^{\text{Train}}$ in most cases perform worse than their concentrated counterparts (**Red** curves). **(c)/(f)**: As node Degree$^{\text{Train}}$ increases, the ratio of nodes owning higher TC$^{\text{Train}}$ increases first and then decreases, corresponding to the observed first-increase-and-then-decrease performance trend in Figure 3.1(c)/(d).

average LP performance on Collab/Citation2 in Figure 3.4(b)/(e). Intriguingly, Lower-degree nodes always perform better than their Higher-degree counterparts across all Degree$^{\text{Train}}$ thresholds. This brings nuances into the conventional understanding that nodes with a weaker topology (lower degree) would yield inferior performance [60, 66, 68]. In contrast, with TC$^{\text{Train}}$ metric, Non-concentrated nodes generally underperform by a noticeable margin than their concentrated counterparts.

We further visualize the relation between Degree$^{\text{Train}}$ and TC$^{\text{Train}}$ in Figure 3.4(c)/(f). When node Degree$^{\text{Train}}$ increases from 1 to 4, the ratio of nodes owning higher TC$^{\text{Train}}$ also increases because these newly landed nodes start interactions and create their initial topological context. Since we have already observed the positive correlation of TC$^{\text{Train}}$ to nodes' LP performance previously, the LP performance for some evaluation metrics also increases as the Degree$^{\text{Train}}$ initially increases from 0 to 4 observed in Figure 3.1(c)/(d). When Degree$^{\text{Train}}$ increases further beyond 5, the ratio of nodes owning higher TC$^{\text{Train}}$ gradually decreases, leading to the decreasing performance observed in the later stage of Figure 3.1(c)/(d). This decreasing TC$^{\text{Train}}$ is because, for high Degree$^{\text{Train}}$ nodes, their neighbors are likely to lie in different communities and share fewer connections among themselves. For example, in social networks, high-activity users usually possess diverse relations in different online communities, and their interacted people are likely from significantly different domains and hence share less common social relations themselves [83].

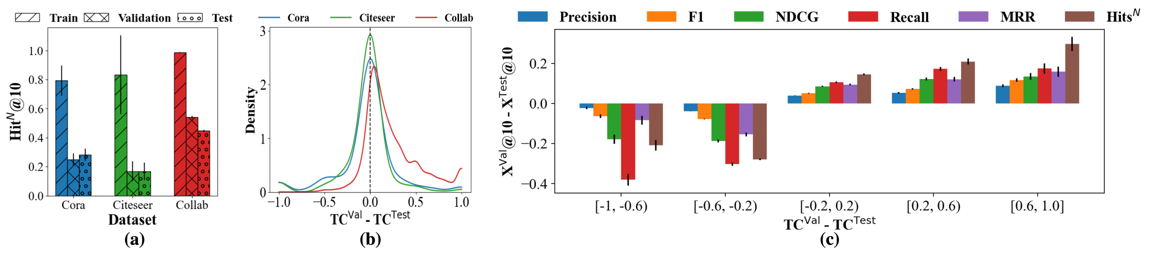


Figure 3.5: **(a)** Hits^N@10 of predicting training/validation/testing edges on Cora/Citeseer/Collab. The gap between validation and testing performance is much bigger on Collab than on Cora/Citeseer. **(b)** Compared with Cora/Citeseer where edges are randomly split, the distribution of the difference between TC$^{\text{Val}}$ and TC$^{\text{Test}}$ shifts slightly right on Collab where edges are split according to time, indicating the interaction between training and testing neighbors become less than the one between training and validation neighbors. **(c)** As the gap between TC$^{\text{Val}}$ and TC$^{\text{Test}}$ increases for different nodes, their corresponding performance gap also increases, demonstrating TDS varies among different nodes even within the same graph.

Obs. 3. Topological Distribution Shift compromises the LP performance at testing time, and TC can measure its negative impact at both graph and node level. In real-world LP scenarios, new nodes continuously join the network and form new links with existing nodes, making the whole network evolve dynamically [84, 85]. *Here, we discover a new Topological Distribution Shift (TDS) issue, i.e., as time goes on, the newly joined neighbors of a node become less interactive with that node's old neighbors.* Since the edges serving message-passing and providing supervision only come from the training set, TDS would compromise the capability of the learned node embeddings for predicting links in the testing set. As verified in Figure 3.5(a), the performance gap between validation and testing sets on Collab where edges are split according to time is much more significant than the one on Cora/Citeseer where edges are split randomly. Note that the significantly higher performance on predicting training edges among all these three datasets is because they have already been used in the training phase [86], and this distribution shift is different from TDS. As TC essentially measures the interaction level among neighbors of a particular node, we further visualize the distribution of the difference between TC$^{\text{Val}}$ and TC$^{\text{Test}}$ in Figure 3.5(b). We observe a slight shift towards the right on Collab rather than on Cora/Citeseer, demonstrating nodes' testing neighbors become less interactive with their training neighbors than their validation neighbors. Figure 3.4(c) further demonstrates the influence of this shift at the node level by visualizing the relationship between TDS and the performance gap. We can see that as the strength of such shift increases (evidenced by the larger difference between TC$^{\text{Val}}$ and TC$^{\text{Test}}$), the performance gap also increases. This suggests that nodes within the same graph display varying levels of TDS. As one potential application, we can devise adaptive data valuation techniques to selectively use neighborhood information (i.e., emphasize less on stale edges in LP as [57] did).

3.3.3 Topological Concentration: Computational Complexity and Optimization

Calculating TC following Eq. (3.1) involves counting the intersection between two neighboring sets that are different hops away from the centering nodes in two computation trees. Assuming the average degree of the network is $\widehat{d}$, the time complexity of computing $C_i^{K,t}$ for all nodes in the network is $\mathcal{O}(|\mathcal{E}|\sum_{k=1}^{K}\sum_{k=1}^{K}\min(\widehat{d}^{k_1}, \widehat{d}^{k_2})) = \mathcal{O}(K^2|\mathcal{E}||\mathcal{V}|) \approx \mathcal{O}(K^2|\mathcal{V}|^2)$ for sparse networks, which increases quadratically as the size of the network increases and is hence challenging for large-scale networks. To handle this issue, we propagate the randomly initialized Gaussian embeddings in the latent space to approximate TC in the topological space and propose *Approximated Topological Concentration* as follows:

Definition 2. ***Approximated Topological Concentration (ATC):*** *Approximated topological concentration $\widetilde{C}_i^{K,t}$ for $v_i \in \mathcal{V}$ is the average similarity between v_i and its neighbors' embeddings initialized from Gaussian Random Projection [87] followed by row-normalized graph diffusion $\widetilde{\mathbf{A}}^k$ [88], with ϕ as the similarity metric function:*

$$\widetilde{C}_i^{K,t} = \mathbb{E}_{v_j\sim\mathcal{N}_i^t}\phi(\mathbf{N}_i, \mathbf{N}_j), \quad \mathbf{N} = \sum_{k=1}^{K}\alpha_k\widetilde{\mathbf{A}}^k\mathbf{R}, \quad \mathbf{R} \sim \mathcal{N}(\mathbf{0}^d, \mathbf{\Sigma}^d) \tag{3.2}$$

Theorem 1. *Assuming $g(|\mathcal{H}_i^{k_1}|, |\mathcal{H}_j^{k_2}|) = |\mathcal{H}_i^{k_1}||\mathcal{H}_j^{k_2}|$ in Eq.* (3.1) *and let ϕ be the dot-product based similarity metric [79], then node v_i's 1-layer Topological Concentration $C_i^{1,t}$ is linear correlated with the mean value of the 1-layer Approximated Topological Concentration $\mu_{\widetilde{C}_i^{K,t}}$ as:*

$$C_i^{1,t} \approx d^{-1}\mu_{\mathbb{E}_{v_j\sim\mathcal{N}_i^t}(\mathbf{E}_j^1)^\top\mathbf{E}_i^1} = d^{-1}\mu_{\widetilde{C}_i^{1,t}}, \tag{3.3}$$

where $\mathbf{E}^1 \in \mathbb{R}^{n\times d}$ denotes the node embeddings after 1-layer SAGE-style message-passing and d is the embedding dimension. The full proof is in Appendix 3.5.2. *This theorem bridges the gap between TC defined in the topological space and ATC defined in the latent space, which theoretically justifies the effectiveness of this approximation.* Computationally, obtaining node embeddings $\mathbf{N}$ in Eq. (3.2) is free from optimization, and the graph diffusion can be efficiently executed via power iteration, which reduces the complexity to $\mathcal{O}(Kd(|\mathcal{E}| + |\mathcal{V}|))$. Note that although we only demonstrate the approximation power for the case of 1-layer message-passing, we empirically verify the efficacy for higher-layer message-passing in the following.

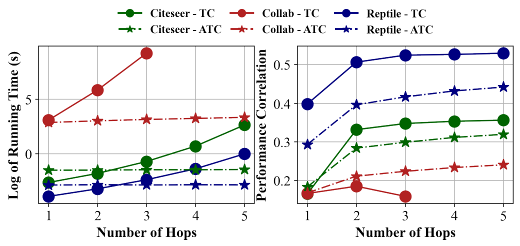


Figure 3.6: ATC$^{\text{Train}}$ maintains a similar level of correlation to TC$^{\text{Train}}$ while significantly reducing the computational time.

Here, we compare TC$^{\text{Train}}$ and ATC$^{\text{Train}}$ under various number of hops in terms of their computational time and their correlation with LP performance in Figure 3.6. As the number of hops increases, the running time for computing TC increases exponentially (especially for large-scale datasets like Collab, we are only affordable to compute its TC^{Train} up to 3 hops) while ATC stays roughly the same. This aligns with the quadratic/linear time complexity $\mathcal{O}(K^2|\mathcal{V}|^2)$/$\mathcal{O}(Kd(|\mathcal{E}| + |\mathcal{V}|))$ we derived earlier for TC/ATC. Moreover, ATC achieves a similar level of correlation to TC at all different hops. For both TC and ATC, their correlations to LP performance increase as the number of hops K used in Eq. (3.1)-Eq. (3.2) increases.

3.4 Conclusion

Although many recent works have achieved unprecedented success in enhancing link prediction (LP) performance with GNNs, demystifying the varying levels of embedding quality and LP performance across different nodes within the graph is heavily under-explored yet fundamental. In this work, we take the lead in understanding the nodes' varying performance from the perspective of their local topology. In view of the connection between link formation and the subgraph interaction, we propose Topological Concentration (TC) to characterize the node LP performance and demonstrate its superiority in leading higher correlation and identifying more low-performing nodes than other common node topological properties. Moreover, we discover a novel topological distribution shift (TDS) issue by observing the changing LP performance over time and demonstrate the capability of using TC to measure this distribution shift. Our work offers the community strong insights into which local topology enables nodes to have better LP performance with GNNs.

3.5 Appendix

3.5.1 Link-centric and Node-centric Evaluation Metrics

In addition to the conventional link-centric evaluation metrics used in this work, node-centric evaluation metrics are also used to mitigate the positional bias caused by the tiny portion of the sampled negative links. We introduce their mathematical definition respectively as follows:

Link-Centric Evaluation. Following [82], we rank the prediction score of each link among a set of randomly sampled negative node pairs and calculate the link-centric evaluation metric Hits@K as the ratio of positive edges that are ranked at K^{th}-place or above. Note that this evaluation may cause bias as the sampled negative links only count a tiny portion of the quadratic node pairs [89]. Hereafter, we introduce the node-centric evaluation metrics and specifically denote the node-level Hit ratio as Hits^N@K to differentiate it from the link-centric evaluation metric Hits@K.

Node-Centric Evaluation. For each node $v_i \in \mathcal{V}$, the model predicts the link formation score between v_i and *every other node*, and selects the top-K nodes to form the potential candidates $\widetilde{\mathcal{E}}_i$. Since the ground-truth candidates for node v_i is $\mathcal{N}_i^{\text{Test}}$ (hereafter, we notate as $\widehat{\mathcal{E}}_i$), we can compute the Recall(R), Precision(P), F1, NDCG(N), MRR and Hits^N of v_i as follows:

$$\text{R@}K_i = \frac{|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|}{|\widehat{\mathcal{E}}_i|}, \qquad \text{P@}K_i = \frac{|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|}{K} \tag{3.4}$$

$$\text{F1@}K_i = \frac{2|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|}{K + |\widehat{\mathcal{E}}_i|}, \qquad \text{N@}K_i = \frac{\sum_{k=1}^{K} \frac{\mathbb{1}[v_{\phi_i^k} \in (\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i)]}{\log_2(k+1)}}{\sum_{k=1}^{K} \frac{1}{\log_2(k+1)}} \tag{3.5}$$

$$\text{MRR@}K_i = \frac{1}{\min_{v \in (\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i)} \text{Rank}_v}, \qquad \text{Hits}^N\text{@}K_i = \mathbb{1}[|\widehat{\mathcal{E}}_i \cap \widetilde{\mathcal{E}}_i| > 0], \tag{3.6}$$

where ϕ_i^k denotes v_i's k^{th} preferred node according to the ranking of the link prediction score, Rank_v is the ranking of the node v and $\mathbb{1}$ is the indicator function equating 0 if the intersection between $\widehat{\mathcal{E}}^i \cap \widetilde{\mathcal{E}}_i$ is empty otherwise 1. The final performance of each dataset is averaged across each node:

$$\text{X@}K = \mathbb{E}_{v_i \in \mathcal{V}} X\text{@}K_i, \text{X} \in \{\text{R}, \text{P}, \text{F1}, \text{N}, \text{MRR}, \text{Hits}^N\} \tag{3.7}$$

Because for each node, the predicted neighbors will be compared against all the other nodes, there is no evaluation bias compared with the link-centric evaluation, where only a set of randomly selected negative node pairs are used.

3.5.2 Proof of Theorems

Approximation power of ATC for TC

Theorem 1. *Assuming $g(|\mathcal{H}_i^{k_1}|, |\mathcal{H}_j^{k_2}|) = |\mathcal{H}_i^{k_1}||\mathcal{H}_j^{k_2}|$ in Eq.* (3.1) *and let ϕ be the dot-product based similarity metric [79], then node v_i's 1-layer Topological Concentration $C_i^{1,t}$ is linear correlated with the mean value of the 1-layer Approximated Topological Concentration $\mu_{\widetilde{C}_i^{K,t}}$ as:*

$$C_i^{1,t} \approx d^{-1}\mu_{\mathbb{E}_{v_j \sim \mathcal{N}_i^t}(\mathbf{E}_j^1)^\top \mathbf{E}_i^1} = d^{-1}\mu_{\widetilde{C}_i^{1,t}}, \tag{3.8}$$

where $\mathbf{E}^1 \in \mathbb{R}^{n\times d}$ denotes the node embeddings after 1-layer SAGE-style message-passing over the node embeddings $\mathbf{R} \sim \mathcal{N}(\mathbf{0}^d, \mathbf{\Sigma}^d)$ and χ is the approximation error.

Proof. Assuming without loss of generalizability that the row-normalized adjacency matrix $\widetilde{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$ is used in aggregating neighborhood embeddings. We focus on a randomly selected node $\mathbf{E}_i \in \mathbb{R}^d, \forall v_i \in \mathcal{V}$ and its 1-layer ATC given by Eq. (3.2) is:

$$\begin{aligned}
\widetilde{C}_i^{1,t} = \mathbb{E}_{v_j\sim\mathcal{N}_i^t}(\mathbf{E}_j^1)^\top\mathbf{E}_i^1 &= \mathbb{E}_{v_j\sim\mathcal{N}_i^t}(\widetilde{\mathbf{A}}\mathbf{R})_j^\top(\widetilde{\mathbf{A}}\mathbf{R})_i \\
&= \mathbb{E}_{v_j\sim\mathcal{N}_i^t}\frac{1}{|\mathcal{N}_j^{\text{Train}}||\mathcal{N}_i^{\text{Train}}|}\Big(\sum_{v_m\in\mathcal{N}_j^{\text{Train}}}\mathbf{R}_m\Big)^\top\Big(\sum_{v_n\in\mathcal{N}_i^{\text{Train}}}\mathbf{R}_n\Big) \\
&= \mathbb{E}_{v_j\sim\mathcal{N}_i^t}\frac{1}{|\mathcal{N}_j^{\text{Train}}||\mathcal{N}_i^{\text{Train}}|}\sum_{(v_m,v_n)\in\mathcal{N}_j^{\text{Train}}\times\mathcal{N}_i^{\text{Train}}}(\mathbf{R}_m)^\top\mathbf{R}_n \\
&= \mathbb{E}_{v_j\sim\mathcal{N}_i^t}\frac{1}{|\mathcal{H}_i^1||\mathcal{H}_j^1|}\Big(\underbrace{\sum_{\substack{(v_m,v_n)\in\mathcal{N}_j^{\text{Train}}\times\mathcal{N}_i^{\text{Train}},\\ v_m\neq v_n}}(\mathbf{R}_m)^\top\mathbf{R}_n}_{\text{Non-common neighbor embedding pairs}} + \underbrace{\sum_{v_k\in\mathcal{N}_j^{\text{Train}}\cap\mathcal{N}_i^{\text{Train}}}(\mathbf{R}_k)^\top\mathbf{R}_k}_{\text{Common neighbor embedding pairs}}\Big),
\end{aligned} \tag{3.9}$$

Note that the first term is the dot product between any pair of two non-common neighbor embeddings, which is essentially the dot product between two independent samples from the same multivariate Gaussian distribution (*note that here we do not perform any training optimization, so the embeddings of different nodes are completely independent*) and by central limit theorem [90] approaches the standard Gaussian distribution with 0 as the mean, $i.e.$, $\mu_{(\mathbf{R}_m)^\top\mathbf{R}_n} = 0$. In contrast, the second term is the dot product between any Gaussian-distributed sample and itself, which can be essentially characterized as the sum of squares of d independent standard normal random variables and hence follows the chi-squared distribution with d degrees of freedom, i.e., $(\mathbf{R}_k)^\top\mathbf{R}_k \sim \chi_d^2$ [91]. By Central Limit Theorem, $\lim_{d\to\infty} P(\frac{\chi_d^2 - d}{\sqrt{2d}} \leq z) = P_{\mathcal{N}(0,1)}(z)$ and hence

$\lim_{d\to\infty}\chi_d^2 = \mathcal{N}(d, 2d), i.e., \mu_{(\mathbf{R}_k)^\top\mathbf{R}_k} = d$. Then we obtain the mean value of $\mathbb{E}_{v_j\sim\mathcal{N}_i^t}(\mathbf{E}_j^1)^\top\mathbf{E}_i^1$:

$$\begin{aligned}\mu_{\widetilde{C}_i^{1,t}} = \mu_{\mathbb{E}_{v_j\sim\mathcal{N}_i^t}(\mathbf{E}_j^1)^\top\mathbf{E}_i^1} &\approx \mathbb{E}_{v_j\sim\mathcal{N}_i^t}\frac{1}{|\mathcal{H}_i^1||\mathcal{H}_j^1|}\Big(\mu_{\sum_{\substack{(v_m,v_n)\in\mathcal{N}_j^{\text{Train}}\times\mathcal{N}_i^{\text{Train}},\\ v_m\neq v_n}}(\mathbf{R}_m)^\top\mathbf{R}_n} + \mu_{\sum_{v_k\in\mathcal{N}_j^{\text{Train}}\cap\mathcal{N}_i^{\text{Train}}}(\mathbf{R}_k)^\top\mathbf{R}_k}\Big) \\ &\approx \mathbb{E}_{v_j\in\mathcal{N}_i^t}\frac{d|\mathcal{N}_i^{\text{Train}}\cap\mathcal{N}_j^{\text{Train}}|}{|\mathcal{H}_i^1||\mathcal{H}_j^1|} = \mathbb{E}_{v_j\in\mathcal{N}_i^t}\frac{d|\mathcal{H}_i^1\cap\mathcal{H}_j^1|}{|\mathcal{H}_i^1||\mathcal{H}_j^1|} = dC_i^{1,t}.\end{aligned} \tag{3.10}$$

The second approximation holds since we set d to be at least 64 for all experiments in this chapter. We next perform Monte-Carlo Simulation to verify that by setting $d = 64$, the obtained distribution is very similar to the Gaussian distribution. Assuming without loss of generality that the embedding dimension is 64 with the mean vector $\boldsymbol{\mu} = \mathbf{0}^{64} \in \mathbb{R}^{64}$ and the identity covariance matrix $\mathbf{\Sigma}^{64} = \mathbf{I} \in \mathbb{R}^{64\times 64}$, we randomly sample 1000 embeddings from $\mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$.

We visualize the distributions of the inner product between the pair of non-common neighbor embeddings, i.e., the first term in Eq. (3.9) $(\mathbf{R}_m)^\top\mathbf{R}_n, v_m \neq v_n$, and the pair of common neighbor embeddings, i.e., the second term in Eq. (3.9) $(\mathbf{R}_k)^\top\mathbf{R}_k, v_k \in \mathcal{N}_j^{\text{Train}} \cap \mathcal{N}_i^{\text{Train}}$ in Figure 3.7. We can see that the distribution of the dot product between the pair of non-common neighbor embeddings behaves like a Gaussian distribution centering around 0. In contrast, the distribution of the dot product between the pair of common neighbor embeddings behaves like a chi-square distribution of degree 64, which also centers around 64, and this in turn verifies the Gaussian approximation. Note that the correctness of the first approximation in Eq. (3.10) relies on the assumption that the average of the inverse of the node's neighbors should be the same across all nodes. Although it cannot be theoretically satisfied, we still empirically verify the positive correlation between TC and the link prediction performance shown in Figure 3.3.

The above derivation bridges the gap between the Topological Concentration (TC) defined in the topological space and the Approximated Topological Concentration (ATC) defined in the latent space, which theoretically justifies the approximation efficacy of ATC. □

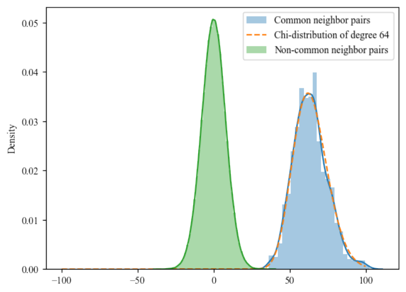


Figure 3.7: The distribution of the inner product between common neighbor pairs is statistically higher than that between non-common neighbor pairs.

CHAPTER 4

Topology Issue: Overcoming the Varying Local Topology Issue in Link Prediction

The previous chapter has demonstrated the impact of varying local topology among different nodes on their link prediction, this chapter focuses on overcoming this issue in the context of recommender systems. Most existing message-passing mechanisms for recommendation are directly inherited from GNNs without scrutinizing whether the varying local topology among different users would compromise the learning of user preferences. In this chapter, we first analyze how message-passing captures the collaborative effect and propose a recommendation-oriented topological metric, Common Interacted Ratio (CIR), which measures the level of interaction between a specific neighbor of a node and the rest of its neighbors. After demonstrating the benefits of leveraging collaborations from neighbors with higher CIR, we propose a recommendation-tailored GNN, Collaboration-Aware Graph Convolutional Network (CAGCN), that goes beyond the 1-Weisfeiler-Lehman(1-WL) test in distinguishing non-bipartite-subgraph-isomorphic graphs. Experiments on six benchmark datasets show that the best CAGCN variant outperforms the most representative GNN-based recommendation model, LightGCN, by nearly 10% in Recall@20 and also achieves around 80% speedup[1].

4.1 Introduction

Recommender systems aim to alleviate information overload by helping users discover items of interest [92, 50] and have been widely deployed in real-world applications [93]. Given historical user-item interactions (e.g., click, purchase, review, and rate), the key is to leverage the collaborative effect [94, 95, 96] to predict how likely users will interact with items. A standard paradigm for modeling collaborative effect is first to learn embeddings of users/items capable of recovering historical user-item interactions and then perform top-K recommendation based on the pairwise similarity between the learned user/item embeddings.

Since historical user-item interactions can be naturally represented as a bipartite graph with users/items being nodes and interactions being edges [96, 97, 79] and given the unprecedented success of GNNs in learning node representations [15, 98, 99, 100], recent research has started to leverage GNNs to learn user/item embeddings for the recommendation. Two pioneering works, NGCF [96] and LightGCN [79], leverage graph convolutions to aggregate messages from local neighborhoods, directly injecting the collaborative signal into user/item embeddings. More recently, [59, 101] explored the robustness and self-supervised learning [102] of graph convolution for recommendation. However, the message-passing mechanisms in all previous recommendation models are directly inherited from GNNs without carefully justifying how collaborative signals

[1]https://dl.acm.org/doi/abs/10.1145/3543507.3583229

are captured and whether the captured collaborative signals would benefit the prediction of user preference. Such an ambiguous understanding of how the message-passing captures collaborative signals would pose the risk of learning uninformative or even harmful user/item representations when adopting GNNs in the recommendation. For example, [103] shows that a large portion of user interactions cannot reflect their actual purchasing behaviors. In this case, blindly passing messages following existing styles of GNNs could capture harmful collaborative signals from these unreliable interactions

To avoid collecting noisy or even harmful collaborative signals in message-passing of traditional GNNs, existing work GTN [103] proposes to adaptively propagate user/item embeddings by adjusting the weight of edges based on items' similarity to users' main preferences (i.e., the trend). However, such similarity is computed based on the learned embeddings that still implicitly encode noisy collaborative signals from unreliable user-item interactions. Worse still, calculating edge weights based on user/item embeddings along the training on the fly is computationally prohibitive and prevents the model from being deployed in industrial-level recommendations. SGCN [59] attaches the message-passing with a trainable stochastic binary mask to prune noisy edges. However, the unbiased gradient estimator increases the computational load.

Despite the fundamental importance of capturing beneficial collaborative signals, the related studies are still in their infancy. To fill this crucial gap, we aim to demystify the collaborative effect captured by message-passing and develop new insights towards customizing message-passing for recommendations. Furthermore, these insights motivate us to design a recommendation-tailored GNN, Collaboration-Aware Graph Convolutional Network(CAGCN), that passes neighborhood information based on their Common Interacted Ratio (CIR) via the Collaboration-Aware Graph Convolution (CAGC). Our major contributions are as follows:

- **Novel Perspective on Collaborative Effect:** We demystify the collaborative effect by analyzing how message-passing helps capture collaborative signals and when the captured collaborative signals are beneficial in computing users' ranking over items.
- **Novel Recommendation-tailored Topological Metric:** We then propose a recommendation-tailored topological metric, Common Interacted Ratio (CIR), and demonstrate the capability of CIR to quantify the benefits of the messages from neighborhoods.
- **Novel Convolution beyond 1-WL for Recommendation:** We integrate CIR into message-passing and propose a novel Collaboration-Aware Graph Convolutional Network (CAGCN). Then we prove that it can go beyond 1-WL test in distinguishing non-bipartite-subgraph-isomorphic graphs, show its superiority on real-world datasets, and provide an in-depth interpretation of its advantages.

Next, we comprehensively analyze the collaborative effect captured by message-passing and propose CIR to measure whether the captured collaborative effect benefits the prediction of user preferences.

4.2 Related Work

Collaborative Filtering & Recommendation. Collaborative filtering (CF) predicts users' interests by utilizing the preferences of other users with similar interests [104]. Early CF methods used Matrix Factorization techniques [105, 106, 107, 108] to capture CF effect via optimizing users/items' embeddings over historical interactions. Stepping further, Graph-based methods either leverage topological constraints or message-passing to inject the CF effect into user/item embeddings [96, 79]. ItemRank and BiRank [109, 110] perform label propagation and compute users' ranking based on structural proximity between the observed and the target items. To make user preferences learnable, HOP-Rec [111] combines the graph-based method and the embedding-based method. Yet, interactions captured by random walks there do not fully explore the high-layer neighbors and multi-hop dependencies [112]. By contrast, GNN-based methods are superior at encoding higher-order structural proximity in user/item embeddings [96, 79]. Recent work [59, 103, 113] has demonstrated that not all captured collaborations improve users' ranking. [59] proposes to learn binary mask and impose low-rank regularization while ours proposes novel topological metric CIR to weigh neighbors' importance. [103] smooths nodes' embeddings based on degree-normalized embedding similarity, while ours adaptively smooth based on topological proximity(CIR). [113] denoises interactions based on 1-layer propagated embeddings and hence cannot go beyond 1-WL test, while ours keeps neighbors and does not focus on diversity issues.

Link Prediction. As a generalized version of recommendation, link prediction finds applications in predicting drug interactions and completing knowledge graphs [51, 114]. Early studies adopt topological heuristics to score node pairs [115, 116, 117]. Furthermore, latent-based/deep-learning methods [55, 118] are proposed to characterize underline topological patterns in node embeddings via random walks [56] or regularizing [55]. To fully leverage node features, GNN-based methods are proposed and achieve unprecedented success owing to the use of the neural network to extract task-related information and the message-passing capture the topological pattern [21, 80, 58]. Recently, efforts have been invested in developing expressive GNNs that can go beyond the 1-WL test [119, 120, 121] for node/graph classification. Following this line, our work develops a recommendation-tailored graph convolution with provably expressive power in predicting user-item links.

4.3 Analysis of Collaborative Effect

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the user-item bipartite graph, where the node set $\mathcal{V} = \mathcal{U} \cup \mathcal{I}$ includes the user set $\mathcal{U}$ and the item set $\mathcal{I}$. Following previous work [96, 79, 122], we only consider the implicit user-item interactions and denote them as edges $\mathcal{E}$ where e_{pq} represents the edge between node p and q. The network topology is described by its adjacency matrix $\mathbf{A} \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{V}|}$, where $\mathbf{A}_{pq} = 1$ when $e_{pq} \in \mathcal{E}$, and $\mathbf{A}_{pq} = 0$ otherwise. Let $\mathcal{N}_p^l$ denote the set of observed neighbors that are exactly l-hops away from p and $\mathcal{S}_p = (\mathcal{V}_{\mathcal{S}_p}, \mathcal{E}_{\mathcal{S}_p})$ be the neighborhood subgraph [120] induced in $\mathcal{G}$ by $\widetilde{\mathcal{N}}_p^1 = \mathcal{N}_p^1 \cup \{p\}$. We use $\mathscr{P}_{pq}^l$ to denote the set of shortest paths of length l between node p and q and denote one of such paths as P_{pq}^l. Note that $\mathscr{P}_{pq}^l = \emptyset$ if it is impossible to have a path between p and q of length l, e.g., $\mathscr{P}_{11}^1 = \emptyset$ in an acyclic graph. We denote users/items' initial embeddings as $\mathbf{E}^0 \in \mathbb{R}^{(n+m)\times d^0}$ where $\mathbf{e}_p^0 = \mathbf{E}_p^0$ and d_p are the node p's embedding and node p's degree.

Following [79, 96], each node has no semantic features but purely learnable embeddings. Therefore, we remove the nonlinear transformation by leveraging LightGCN [79] as the canonical architecture and exclusively explore the collaborative effect captured by message-passing. LightGCN passes messages from user u/item i's neighbors within L-hops to u/i:

$$\mathbf{e}_u^{l+1} = d_u^{-0.5} \sum_{j \in \mathcal{N}_u^1} d_j^{-0.5} \mathbf{e}_j^l, \mathbf{e}_i^{l+1} = d_i^{-0.5} \sum_{v \in \mathcal{N}_i^1} d_v^{-0.5} \mathbf{e}_v^l, \tag{4.1}$$

$\forall l \in \{0, ..., L\}$. The propagated embeddings at all layers, including the original embedding, are aggregated together via mean-pooling:

$$\mathbf{e}_u = \frac{1}{(L+1)} \sum_{l=0}^{L} \mathbf{e}_u^l, \quad \mathbf{e}_i = \frac{1}{(L+1)} \sum_{l=0}^{L} \mathbf{e}_i^l, \forall u \in \mathcal{U}, \forall i \in \mathcal{I} \tag{4.2}$$

In the training stage, for each observed user-item interaction (u, i), LightGCN randomly samples a negative item i^- that u has never interacted with before and forms the triple (u, i, i^-), which collectively forms the set of observed training triples $\mathcal{O}$. After that, the ranking scores of the user over these two items are computed as $y_{ui} = \mathbf{e}_u^\top \mathbf{e}_i$ and $y_{ui^-} = \mathbf{e}_u^\top \mathbf{e}_{i^-}$, which are finally used in optimizing the pairwise Bayesian Personalized Ranking (BPR) loss [107]:

$$\mathcal{L}_{\text{BPR}} = \sum_{(u,i,i^-) \in \mathcal{O}} -\ln \sigma(y_{ui} - y_{ui^-}), \tag{4.3}$$

where $\sigma(\cdot)$ is the Sigmoid function, and we omit the L_2 regularization here since it is mainly for alleviating overfitting and has no influence on the collaborative effect captured by message passing. Under the above LightGCN framework, we expect to answer the following two questions:

- $\boldsymbol{Q_1}$: How does message-passing capture and use the collaborative effect in computing users' ranking?
- $\boldsymbol{Q_2}$: When do collaborations captured by message-passing benefit users' ranking over items?

Next, We address $\boldsymbol{Q_1}$ by theoretically deriving users' ranking over items under the message-passing framework of LightGCN and address $\boldsymbol{Q_2}$ by proposing the Common Interacted Ratio (CIR) to measure the benefits of leveraging collaborations from each neighbor in computing users' ranking. The answers to the above two questions further motivate our design of Collaboration-Aware Graph Convolutional Network in Section 4.4.

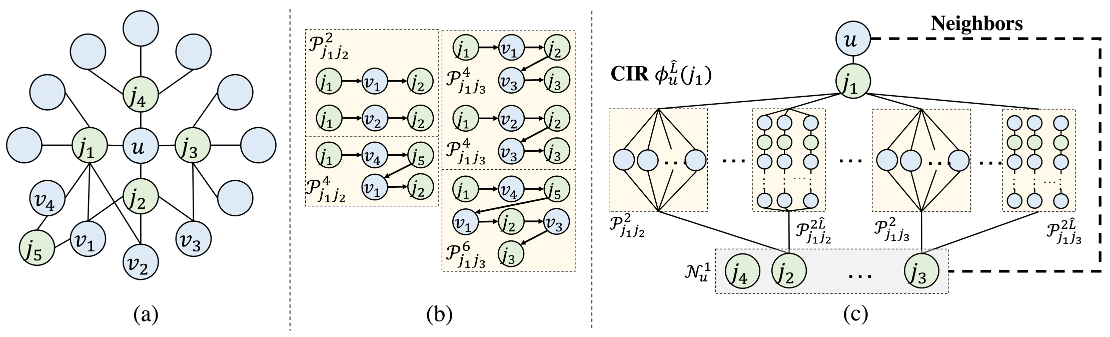

(a)
(b)
(c)
Neighbors
CIR $\phi_u^{\hat{L}}(j_1)$
$\mathcal{P}^2_{j_1 j_2}$
$\mathcal{P}^4_{j_1 j_2}$
$\mathcal{P}^4_{j_1 j_3}$
$\mathcal{P}^6_{j_1 j_3}$
$\mathcal{P}^{2\hat{L}}_{j_1 j_2}$
$\mathcal{P}^2_{j_1 j_3}$
$\mathcal{P}^{2\hat{L}}_{j_1 j_3}$
$\mathcal{N}_u^1$

Figure 4.1: In (a)-(b), j_1 has more interactions (paths) with (to) u's neighborhood than j_4 and hence is more representative of u's purchasing behaviors than j_4. In (c), we quantify CIR between j_1 and u via the paths (and associated nodes) between j_1 and $\mathcal{N}_u^1$.

4.3.1 How does message-passing capture collaborative effect?

The collaborative effect occurs when the prediction of a user's preference relies on other users' preferences or items' properties [123]. Therefore, to answer $\boldsymbol{Q_1}$, we need to seek whether we leverage other nodes' embeddings in computing a specific user's ranking over items. In the inference stage of LightGCN, we take the inner product between user u's embedding and item i's embedding after L-layers' message-passing to compute the ranking as

$$y_{ui}^L = (\sum_{l_1=0}^{L}\sum_{j\in\mathcal{N}_u^{l_1}}\sum_{l_2=l_1}^{L}\beta_{l_2}\alpha_{ju}^{l_2}\mathbf{e}_j^0)^\top(\sum_{l_1=0}^{L}\sum_{v\in\mathcal{N}_i^{l_1}}\sum_{l_2=l_1}^{L}\beta_{l_2}\alpha_{vi}^{l_2}\mathbf{e}_v^0), \tag{4.4}$$

where $\alpha_{ju}^{l_2} = \sum_{P_{ju}^{l_2}\in\mathscr{P}_{ju}^{l_2}}\prod_{e_{pq}\in P_{ju}^{l_2}} d_p^{-0.5}d_q^{-0.5}(\alpha_{ju}^{l_2} = 0 \text{ if } \mathscr{P}_{ju}^{l_2} = \emptyset)$ denotes the total weight of all paths of length l_2 from j to u, $\mathcal{N}_u^0 = \{u\}$ and specifically, $\alpha_{uu}^0 = 1$. β_{l_2} is the weight measuring contributions of propagated embeddings at layer l_2. Thus, based on Eq. (4.4), we present the answer to $\boldsymbol{Q_1}$ as $\boldsymbol{A_1}$: *L-layer LightGCN-based message-passing captures collaborations between pairs of nodes* $\{(j,v)|j\in\bigcup_{l=0}^L\mathcal{N}_u^l, v\in\bigcup_{l=0}^L\mathcal{N}_i^l\}$*, and the collaborative strength of each pair is determined by 1)* $\mathbf{e}_j^{0\top}\mathbf{e}_v^0$*: embedding similarity between j and v, 2)* $\{\alpha_{ju}^l\}_{l=0}^L(\{\alpha_{vi}^l\}_{l=0}^L)$*: weight of all paths of length l to L from j to u (v to i), and 3)* $\{\beta_l\}_{l=0}^L$*: the weight of each layer.*

4.3.2 When is the captured collaborative effect beneficial to users' ranking?

Although users could leverage collaborations from other users/items as demonstrated above, we cannot guarantee all of these collaborations benefit the prediction of their preferences. For example, in Figure 4.1(a)-(b), u's interacted item j_1 has more interactions (paths) to u's neighborhoods than j_4 and hence is more representative of u's purchasing behaviors [59, 103]. For each user u, we propose the Common Interacted Ratio to quantify the level of interaction between each specific neighbor of u and u's whole item neighborhood:

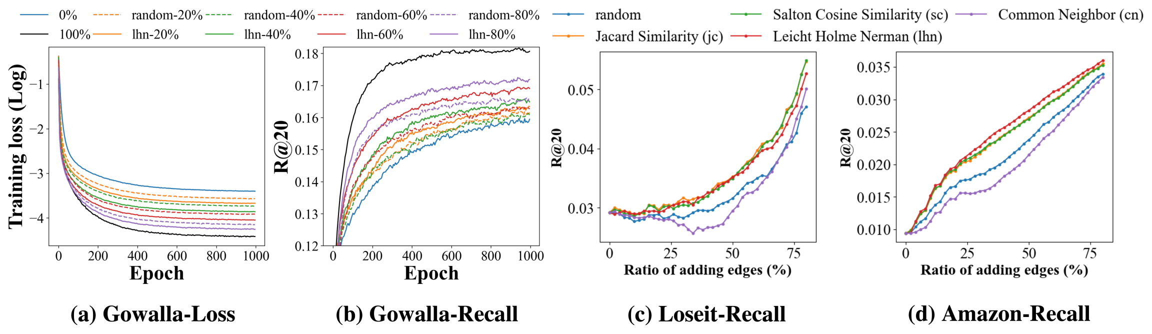


Figure 4.2: **(a)**-**(b)**: The training loss **(a)** is lower, and the performance **(b)** is higher when adding edges according to the variant CIR-lhn (Leicht Holme Nerman) than adding randomly under the same addition budget. **(c)**-**(d)**: The performance of adding edges according to CIR variants generally increases faster than adding randomly after pre-training on Loseit **(c)** and Amazon **(d)**.

Definition 3. ***Common Interacted Ratio (CIR):*** *For any item $j \in \mathcal{N}_u^1$ of user u, the CIR of j around u considering nodes up to $(\widehat{L}+1)$-hops away from u, i.e., $\phi_u^{\widehat{L}}(j)$, is defined as the average interacted ratio of j with all neighboring items of u in $\mathcal{N}_u^1$ through paths of length $\leq 2\widehat{L}$:*

$$\phi_u^{\widehat{L}}(j) = \frac{1}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \sum_{l=1}^{\widehat{L}} \alpha^{2l} \sum_{P_{ji}^{2l} \in \mathscr{P}_{ji}^{2l}} \frac{1}{f(\{\mathcal{N}_k^1 | k \in P_{ji}^{2l}\})}, \tag{4.5}$$

$\forall j \in \mathcal{N}_u^1, \forall u \in \mathcal{U}$, where $\{\mathcal{N}_k^1 | k \in P_{ji}^{2l}\}$ represents the set of the 1-hop neighborhood of node k along the path P_{ji}^{2l} from node j to i of length $2l$ including i, j. f is a normalization function to differentiate the importance of different paths in $\mathscr{P}_{ji}^{2l}$ and its value depends on the neighborhood of each node along the path P_{ji}^{2l}. α^{2l} is the importance of paths of length $2l$.

As shown in Figure 4.1(c), $\phi_u^{\widehat{L}}(j_1)$ is decided by paths of length 2 to $2\widehat{L}$. By configuring different $\widehat{L}$ and f, $\sum_{P_{ji}^{2l} \in \mathscr{P}_{ji}^{2l}} \frac{1}{f(\{\mathcal{N}_k^1 | k \in P_{ji}^{2l}\})}$ could express many graph similarity metrics [115, 116, 117, 124, 125] and we discuss them in Appendix 4.7.1. For simplicity, henceforth we denote $\phi_u^{\widehat{L}}(j)$ as $\phi_u(j)$. We next empirically verify the importance of leveraging collaborations from neighbors with higher CIR by incrementally adding edges into an initially edge-less graph according to their CIR and visualizing the performance change. Specifically, we consider the performance change in two settings, retraining and pretraining, which are visualized in Figure 4.2. In both settings, we iteratively cycle each node and add its corresponding neighbor according to the CIR until we hit the budget. Here, we consider variants of CIR that we later define in Section 4.5.1 with further details in Appendix 4.7.1.

For the re-training setting, we first remove all observed edges in the training set to create the edgeless bipartite graph and then incrementally add edges according to their CIR and retrain user/item embeddings. In Figure 4.2(a)-(b), we evaluate the performance on the newly constructed bipartite graph under different edge budgets. Clearly, the training loss/performance becomes lower/higher when adding more edges because message-passing captures more collaborative effects. Furthermore, since edges with higher CIR connect neighbors with more connections to the whole neighborhood, optimizing embeddings of nodes incident to these edges pull the whole neighborhood closer and hence leads to the lower training loss over neighborhoods' connections, which causes the overall lower training loss in Figure 4.2(a). In Figure 4.2(b), we observe that under the same adding budget, adding according to CIRs achieves higher performance than adding randomly. It is because neighbors with higher interactions with the whole neighborhood are more likely to have higher interactions with neighbors to be predicted (We empirically verify this in Table 4.4.). Then, for each user, maximizing its embedding similarity to its training neighbors with higher CIR will indirectly improve its similarity to its to-be-predicted neighbors, which leads to lower population risk and higher generalization/performance.

For the pre-training setting, we first pre-train user/item embeddings on the original bipartite graph and then propagate the pre-trained embeddings on the newly constructed bipartite graph under different edge budgets. This setting is more realistic since in the real world, with the exponential interactions streamingly coming in [126] while the storage space is limited, we are forced to keep only partial interactions and the pre-trained user/item embeddings. Figure 4.2(c) demonstrates that under the same adding budget, keeping edges according to CIR leads to higher performance than keeping randomly, which further verifies the effectiveness of CIR in quantifying the edge importance. An interesting observation is that adding more edges cannot always bring performance gain, as shown in Figure 4.2(d) when the ratio of added edges is between 0%-20%. We hypothesize there are two reasons. From network topology, only when edges are beyond a certain level can the network form a giant component so that users can receive enough neighborhood information. Secondly, from representation learning, more nodes would have inconsistent neighborhood contexts between the training and the inference when only a few edges are added. Such inconsistent neighborhood context would compromise the performance and will be alleviated when more edges are added, as shown later in Figure 4.2(c). Furthermore, different CIR variants cause different increasing speeds of performance. For example, sc is faster on Loseit in Figure 4.2(c) while lhn is faster on Amazon in Figure 4.2(d). Except for the cn, jc/sc/lhn leads to faster improvement than the random one, which highlights the potential of CIR in devising cost-effective strategies for pruning edges in the continual learning [127].

From the above analysis, we summarize the answer $\boldsymbol{A_2}$ to $\boldsymbol{Q_2}$ as: *Leveraging collaborations from u's neighboring node j with higher CIR $\phi_u(j)$ would cause more benefits to u's ranking.*

4.4 Collaboration-aware Graph Convolutional Networks

The former section demonstrates that passing messages according to neighbors' CIR is crucial in improving users' ranking. This motivates us to propose a new graph convolution operation, Collaboration-Aware Graph

Convolution(CAGC), which passes node messages based on the benefits of their provided collaborations. Furthermore, we wrap the proposed CAGC within LightGCN and develop two CAGC-based models.

4.4.1 Collaboration-Aware Graph Convolution

The core idea of CAGC is to strengthen/weaken the messages passed from neighbors with higher/lower CIR to center nodes. To achieve this, we compute the edge weight as:

$$\mathbf{\Phi}_{ij} = \begin{cases} \phi_i(j), & \text{if } \mathbf{A}_{ij} > 0 \\ 0, & \text{if } \mathbf{A}_{ij} = 0 \end{cases}, \forall i, j \in \mathcal{V} \tag{4.6}$$

where $\phi_i(j)$ is the CIR of neighboring node j centering around i. Note that unlike the symmetric graph convolution $\mathbf{D}^{-0.5}\mathbf{A}\mathbf{D}^{-0.5}$ used in LightGCN, here $\mathbf{\Phi}$ is unsymmetric. This is rather interpretable: the interacting level of node j with i's neighborhood is likely to be different from the interacting level of node i with j's neighborhood. We further normalize $\mathbf{\Phi}$ and combine it with the LightGCN convolution:

$$\mathbf{e}_i^{l+1} = \sum_{j \in \mathcal{N}_i^1} g(\gamma_i \frac{\mathbf{\Phi}_{ij}}{\sum_{k \in \mathcal{N}_i^1} \mathbf{\Phi}_{ik}}, d_i^{-0.5} d_j^{-0.5}) \mathbf{e}_j^l, \forall i \in \mathcal{V} \tag{4.7}$$

where γ_i is a coefficient that varies the total amount of messages flowing to node i and controls its embedding magnitude [128]. g is a function combining the edge weights computed based on CIR and LightGCN. We could either simply set g as the weighted summation of these two propagated embeddings or learn g by parametrization. Next, we prove that for certain choices of g, CAGC can go beyond 1-WL in distinguishing non-bipartite-subgraph-isomorphic graphs. First, we prove the equivalence between the subtree-isomorphism and the subgraph-isomorphism in bipartite graphs:

Theorem 2. *In bipartite graphs, two subgraphs are subtree-isomorphic iff they are subgraph-isomorphic*[2].

Proof. We prove this theorem in two directions. Firstly ($\Longrightarrow$), we prove that in a bipartite graph, two subgraphs that are subtree-isomorphic are also subgraph-isomorphic by contradiction. Assuming that there exists two subgraphs $\mathcal{S}_u$ and $\mathcal{S}_i$ that are subtree-isomorphic yet not subgraph-isomorphic in a bipartite graph, i.e., $\mathcal{S}_u \cong_{subtree} \mathcal{S}_i$ and $\mathcal{S}_u \ncong_{subgraph} \mathcal{S}_i$. By definition of subtree-isomorphism, we trivially have $\mathbf{e}_v^l = \mathbf{e}_{h(v)}^l, \forall v \in \mathcal{V}_{\mathcal{S}_u}$. Then to guarantee $\mathcal{S}_u \ncong_{subgraph} \mathcal{S}_i$ and also since edges are only allowed to connect u and its neighbors $\mathcal{N}_u^1$ in the bipartite graph, there must exist at least an edge e_{uv} between u and one of its neighbors $v \in \mathcal{N}_u^1$ such that $e_{uv} \in \mathcal{E}_{\mathcal{S}_u}, e_{h(u)h(v)} \notin \mathcal{E}_{\mathcal{S}_i}$, which contradicts the assumption that $\mathcal{S}_u \cong_{subtree} \mathcal{S}_i$. Secondly ($\Longleftarrow$), we can prove that in a bipartite graph, two subgraphs that are subgraph-

[2]Definitions of subtree-/subgraph-isomorphism are in Appendix 4.7.3 [120].

isomorphic are also subtree-isomorphic, which trivially holds since in any graph, subgraph-isomorphism leads to subtree-isomorphism [120]. □

Since the 1-WL test can distinguish subtree-isomorphic graphs [120], the equivalence between these two isomorphisms indicates that in bipartite graphs, both of the subtree-isomorphic graphs and subgraph-isomorphic graphs can be distinguished by 1-WL test. Therefore, to go beyond 1-WL in bipartite graphs, we need to propose a novel graph isomorphism, bipartite-subgraph-isomorphism in Definition 4, which is even harder to distinguish than the subgraph-isomorphism by 1-WL test.

Definition 4. ***Bipartite-subgraph-isomorphism:*** *$\mathcal{S}_u$ and $\mathcal{S}_i$ are bipartite-subgraph-isomorphic, denoted as $\mathcal{S}_u \cong_{bi-subgraph} \mathcal{S}_i$, if there exists a bijective mapping $h : \widetilde{\mathcal{N}}_u^1 \cup \mathcal{N}_u^2 \to \widetilde{\mathcal{N}}_i^1 \cup \mathcal{N}_i^2$ such that $h(u) = i$ and $\forall v, v' \in \widetilde{\mathcal{N}}_u^1 \cup \mathcal{N}_u^2$, $e_{vv'} \in \mathcal{E} \iff e_{h(v)h(v')} \in \mathcal{E}$ and $\mathbf{e}_v^l = \mathbf{e}_{h(v)}^l, \mathbf{e}_{v'}^l = \mathbf{e}_{h(v')}^l$.*

Lemma 1. *If g is a multilayer perceptron (MLP), then we have that $g(\{(\gamma_i\widetilde{\mathbf{\Phi}}_{ij}, \mathbf{e}_j^l)|j \in \mathcal{N}_i^1\}, \{(d_i^{-0.5}d_j^{-0.5}, \mathbf{e}_j^l)$ $|j \in \mathcal{N}_i^1\})$ is injective.*

Proof. If we assume that all node embeddings share the same discretization precision, then embeddings of all nodes in a graph can form a countable set $\mathcal{H}$. Similarly, for each edge in a graph, its CIR-based weight $\widetilde{\mathbf{\Phi}}_{ij}$ and degree-based weight $d_i^{-0.5}d_j^{-0.5}$ can also form two different countable sets $\mathcal{W}_1, \mathcal{W}_2$ with $|\mathcal{W}_1| = |\mathcal{W}_2|$. Then $\mathcal{P}_1 = \{\widetilde{\mathbf{\Phi}}_{ij}\mathbf{e}_i|\widetilde{\mathbf{\Phi}}_{ij} \in \mathcal{W}_1, \mathbf{e}_i \in \mathcal{H}\}, \mathcal{P}_2 = \{d_i^{-0.5}d_j^{-0.5}\mathbf{e}_i|d_i^{-0.5}d_j^{-0.5} \in \mathcal{W}_2, \mathbf{e}_i \in \mathcal{H}\}$ are also two countable sets. Let P_1, P_2 be two multisets containing elements from $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively, and $|P_1| = |P_2|$. Then by Lemma 1 in [120], there exists a function s such that $\pi(P_1, P_2) = \sum_{p_1 \in P_1, p_2 \in P_2} s(p_1, p_2)$ is unique for any distinct pair of multisets (P_1, P_2). Since the MLP-based g is a universal approximator [18] and hence can learn s, we know that g is injective. □

Theorem 3. *Let M be a GNN with a sufficient number of CAGC-based convolution layers defined by Eq.* (4.7). *If g is MLP, then M is strictly more expressive than 1-WL in distinguishing subtree-isomorphic yet non-bipartite-subgraph-isomorphic graphs.*

Proof. We prove this theorem in two directions. Firstly ($\Longrightarrow$), following [120], we prove that the designed CAGC here can distinguish any two graphs that are distinguishable by 1-WL by contradiction. Assume that there exist two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ which can be distinguished by 1-WL but cannot be distinguished by CAGC. Further, suppose that 1-WL cannot distinguish these two graphs in the iterations from 0 to $L-1$, but can distinguish them in the L^{th} iteration. Then, there must exist two neighborhood subgraphs $\mathcal{S}_u$ and $\mathcal{S}_i$ whose neighboring nodes correspond to two different sets of node labels at the L^{th} iteration, i.e., $\{\mathbf{e}_v^l|v \in \mathcal{N}_u^1\} \neq \{\mathbf{e}_j^l|j \in \mathcal{N}_i^1\}$. Since g is injective by Lemma 1, for $\mathcal{S}_u$ and $\mathcal{S}_i$, g would yield two different feature vectors at the L^{th} iteration. This means that CAGC can also distinguish $\mathcal{G}_1$ and $\mathcal{G}_2$, which contradicts the assumption.

Secondly ($\Longleftarrow$), we prove that there exist at least two graphs that can be distinguished by CAGC but cannot be distinguished by 1-WL. Figure 4.7 in Appendix 4.7.3 presents two of such graphs $\mathcal{S}_u, \mathcal{S}'_u$, which are subgraph isomorphic but non-bipartite-subgraph-isomorphic. Assuming u and u' have the same neighborhood feature vectors $\mathbf{e}$, then directly propagating according to 1-WL or even considering node degree as the edge weight as GCN [15] can still end up with the same propagated feature for u and u'. However, suppose we leverage JC to calculate CIR as introduced in Appendix 4.7.1. In that case, we end up with:

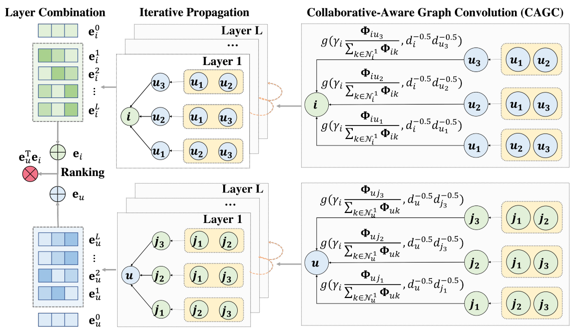
$$\{(d_u d_{j_1})^{-0.5}\mathbf{e}, (d_u d_{j_2})^{-0.5}\mathbf{e}, (d_u d_{j_3})^{-0.5}\mathbf{e}\} \neq \{(d_{u'}^{-0.5} d_{j'_1}^{-0.5} + \widetilde{\mathbf{\Phi}}_{u'j'_1})\mathbf{e}, (d_{u'}^{-0.5} d_{j'_2}^{-0.5} + \widetilde{\mathbf{\Phi}}_{u'j'_2})\mathbf{e}, (d_{u'}^{-0.5} d_{j'_3}^{-0.5} + \widetilde{\mathbf{\Phi}}_{u'j'_3})\mathbf{e}\} \quad (4.8)$$

Since g is injective by Lemma 1, CAGC would yield two different embeddings for u and u'. $\square$

Theorem 3 indicates that GNNs whose aggregation scheme is CAGC can distinguish non-bipartite-subgraph-isomorphic graphs that are indistinguishable by 1-WL.

4.4.2 Model Architecture and Complexity

Following the principle of LightGCN that the designed graph convolution should be light and easy to train, except for the message-passing component, all other components of our proposed CAGC-based models are exactly the same as LightGCN including the average pooling and the model training, which have already been covered in Section 4.3. We visualize the architecture of CAGC-based models in Figure 4.3. Based on the choice of g, we have two specific model variants. For the first variant CAGCN, we calculate the edge weight solely based on CIR in message-passing by setting $g(A, B) = A$ in Eq.(4.7) and set $\gamma_i = \sum_{r \in \mathcal{N}_i^1} d_i^{-0.5} d_r^{-0.5}$ to ensure that the total edge weights for messages received by each node are the same as the one in LightGCN. For CAGCN*, we set g as the weighted summation and set $\gamma_i = \gamma$ as a constant controlling the trade-off between contributions from message-passing by LightGCN and by CAGC. We term the model variant as CAGCN(*)-jc if we use Jaccard Similarity (JC) [125] to compute $\mathbf{\Phi}$. The same rule applies to other topological metrics listed in Appendix 4.7.1. Concrete equations of CAGCN and CAGCN* are provided in Appendix 4.7.2.



Layer Combination
Iterative Propagation
Collaborative-Aware Graph Convolution (CAGC)
$\mathbf{e}_i^0$
$\mathbf{e}_i^1$
$\mathbf{e}_i^2$
$\mathbf{e}_i^L$
$\mathbf{e}_u^\top \mathbf{e}_i$
$\mathbf{e}_i$
Ranking
$\mathbf{e}_u$
$\mathbf{e}_u^L$
$\mathbf{e}_u^2$
$\mathbf{e}_u^1$
$\mathbf{e}_u^0$
Layer L
Layer 1
$g(\gamma_i \frac{\mathbf{\Phi}_{iu_3}}{\sum_{k \in \mathcal{N}_i^1} \mathbf{\Phi}_{ik}}, d_i^{-0.5} d_{u_3}^{-0.5})$
$g(\gamma_i \frac{\mathbf{\Phi}_{iu_2}}{\sum_{k \in \mathcal{N}_i^1} \mathbf{\Phi}_{ik}}, d_i^{-0.5} d_{u_2}^{-0.5})$
$g(\gamma_i \frac{\mathbf{\Phi}_{iu_1}}{\sum_{k \in \mathcal{N}_i^1} \mathbf{\Phi}_{ik}}, d_i^{-0.5} d_{u_1}^{-0.5})$
$g(\gamma_i \frac{\mathbf{\Phi}_{uj_3}}{\sum_{k \in \mathcal{N}_u^1} \mathbf{\Phi}_{uk}}, d_u^{-0.5} d_{j_3}^{-0.5})$
$g(\gamma_i \frac{\mathbf{\Phi}_{uj_2}}{\sum_{k \in \mathcal{N}_u^1} \mathbf{\Phi}_{uk}}, d_u^{-0.5} d_{j_3}^{-0.5})$
$g(\gamma_i \frac{\mathbf{\Phi}_{uj_1}}{\sum_{k \in \mathcal{N}_u^1} \mathbf{\Phi}_{uk}}, d_u^{-0.5} d_{j_1}^{-0.5})$

Figure 4.3: The architecture of CAGCN(*).

4.5 Experiment

4.5.1 Experimental Settings

Datasets. Following [96, 79], we validate the proposed approach on **Gowalla**, **Yelp**, **Amazon**, and **Ml-1M**, the details of which are provided in [96, 79]. Moreover, we collect two extra datasets to further demonstrate the superiority of our proposed model in even broader user-item interaction domains: **(1) Loseit**: This dataset is collected from subreddit *loseit - Lose the Fat*[3] from March 2020 to March 2022 where users discuss healthy and sustainable methods of losing weight via posts. To ensure the quality of this dataset, we use the 10-core setting [129], i.e., retaining users and posts with at least ten interactions. **(2) News**: This dataset includes the interactions from subreddit *World News*[4] where users share major news around the world via posts. Similarly, we use 10-core setting. We summarize statistics of all six datasets in Table 4.1.

Table 4.1: Basic dataset statistics for recommender system.

Dataset	**# Users**	**# Items**	**# Interactions**	**Density**
Gowalla	$29,858$	$40,981$	$1,027,370$	0.084%
Yelp	$31,668$	$38,048$	$1,561,406$	0.130%
Amazon	$52,643$	$91,599$	$2,984,108$	0.062%
Ml-1M	$6,022$	$3,043$	$895,699$	4.888%
Loseit	$5,334$	$54,595$	$230,866$	0.08%
News	$29,785$	$21,549$	$766,874$	0.119%

Yelp**: Yelp2018; ***Amazon**: Amazon-Books;Ml-1M**: Movielens-1M.

Baseline methods. We compare our model with MF, NGCF, LightGCN, UltraGCN, GTN [107, 96, 79, 122, 103]. Details of them are clarified in Appendix 4.7.2. Since here the purpose is to evaluate the effectiveness of CAGC-based message-passing, we only compare with baselines that focus on graph convolution (besides the classic MF) including the state-of-the-art GNN-based recommendation models (i.e., UltraGCN and GTN). Note that our work could be further enhanced if incorporating other techniques such as contrastive learning to derive self-supervision but stacking these would sidetrack the main topic of this chapter, graph convolution, so we leave them as one future direction.

CAGCN-variants. For CAGCN, we calculate the edge weight solely based on our proposed CIR in message-passing by setting $g(A, B) = A$ in Eq. (4.7) and set $\gamma_i = \sum_{r \in \mathcal{N}_i^1} d_i^{-0.5} d_r^{-0.5}$ to ensure that the total edge weights for messages received by each node are the same as the one in LightGCN. For CAGCN*, we set g as the weighted summation and set $\gamma_i = \gamma$ as a constant controlling the trade-off between contributions from message-passing by LightGCN and by CAGC. We term the model variant as CAGCN(*)-jc if we use Jaccard Similarity (JC) [125] to compute $\mathbf{\Phi}$. The same rule applies to other topological metrics listed in Appendix 4.7.1. Concrete equations of CAGCN and CAGCN* are provided in Appendix 4.7.2.

[3]https://www.reddit.com/r/loseit/

[4]https://www.reddit.com/r/worldnews/

Evaluation Metrics. Two popular metrics: Recall and Normalized Discounted Cumulative Gain(NDCG) [96] are adopted for evaluation. We set the default value of K as 20 and report the average of Recall@20 and NDCG@20 over all users in the test set. During inference, we treat items that the user has never interacted with in the training set as candidate items. All models predict users' preference scores over these candidate items and rank them based on the computed scores to further calculate Recall@20 and NDCG@20.

Table 4.2: Comparing CAGCN(*) with baselines. The best and runner-up results are in **bold** and underlined.

Model	Metric	MF	NGCF	LightGCN	UltraGCN	CAGCN				CAGCN*		
						-jc	-sc	-cn	-lhn	-jc	-sc	-lhn
Gowalla	Recall@20	0.1554	0.1563	0.1817	0.1867	0.1825	0.1826	0.1632	0.1821	**0.1878**	**0.1878**	0.1857
	NDCG@20	0.1301	0.1300	0.1570	0.1580	0.1575	0.1577	0.1381	0.1577	**0.1591**	0.1588	0.1563
Yelp2018	Recall@20	0.0539	0.0596	0.0659	0.0675	0.0674	0.0671	0.0661	0.0661	0.0708	**0.0711**	0.0676
	NDCG@20	0.0460	0.0489	0.0554	0.0553	0.0564	0.0560	0.0546	0.0555	0.0586	**0.0590**	0.0554
Amazon	Recall@20	0.0337	0.0336	0.0420	**0.0682**	0.0435	0.0435	0.0403	0.0422	0.0510	0.0506	0.0457
	NDCG@20	0.0265	0.0262	0.0331	**0.0553**	0.0343	0.0342	0.0321	0.0333	0.0403	0.0400	0.0361
Ml-1M	Recall@20	0.2604	0.2619	0.2752	0.2783	0.2780	0.2786	0.2730	0.2760	0.2822	**0.2827**	0.2799
	NDCG@20	0.2697	0.2729	0.2820	0.2638	0.2871	**0.2881**	0.2818	0.2871	0.2775	0.2776	0.2745
Loseit	Recall@20	0.0539	0.0574	0.0588	0.0621	0.0622	0.0625	0.0502	0.0592	0.0654	**0.0658**	**0.0658**
	NDCG@20	0.0420	0.0442	0.0465	0.0446	0.0474	0.0470	0.0379	0.0461	0.0486	0.0484	**0.0489**
News	Recall@20	0.1942	0.1994	0.2035	0.2034	0.2135	0.2132	0.1726	0.2084	**0.2182**	0.2172	0.2053
	NDCG@20	0.1235	0.1291	0.1311	0.1301	0.1385	0.1384	0.1064	0.1327	0.1405	**0.1414**	0.1311
Avg. Rank	Recall@20	9.83	9.17	7.33	4.17	4.67	4.33	8.83	6.17	1.67	**1.50**	3.33
	NDCG@20	9.50	9.17	5.83	6.00	3.67	4.00	8.33	5.00	**2.50**	**2.50**	5.17

jc-Jacard Similarity, **sc**-Salton Cosine Similarity, **cn**-Common Neighbors, **lhn**-Leicht-Holme-Nerman

4.5.2 Performance Comparison

Table 4.3: Comparison of CAGCN* with GTN.

Model	Metric	GTN	CAGCN*		
			-jc	-sc	-lhn
Gowalla	Recall@20	0.1870	**0.1901**	0.1899	0.1885
	NDCG@20	0.1588	**0.1604**	0.1603	0.1576
Yelp2018	Recall@20	0.0679	**0.0731**	0.0729	0.0689
	NDCG@20	0.0554	**0.0605**	0.0601	0.0565
Amazon	Recall@20	0.0450	0.0573	**0.0575**	0.0520
	NDCG@20	0.0346	0.0456	**0.0458**	0.0409

We first compare our proposed CAGCN-variants with LightGCN. In Table 4.2, CAGCN-jc/sc/lhn achieves higher performance than LightGCN because we aggregate more information from nodes with higher CIR(jc, sc, lhn) that bring more beneficial collaborations as justified in Section 4.3.2. CAGCN-cn generally performs worse than LightGCN because nodes having more common neighbors with other nodes tend to have higher degrees, and blindly aggregating more information from these nodes would cause false-positive link prediction. Since different datasets exhibit different patterns of 2$^{\text{nd}}$-order connectivity, no fixed topological metric performs the best among all datasets. For example, CAGCN-jc performs better than CAGCN-sc on Yelp and News but worse on Gowalla and Ml-1M. Then, we compare CAGCN*-variants with other baselines. We omit CAGCN*-cn here due to the worse performance of CAGCN-cn than LightGCN. We can see that CAGCN*-jc/sc almost consistently achieves higher performance than other baselines except for UltraGCN on Amazon. This is because UltraGCN allows multiple negative samples for each positive interaction, e.g., 500 negative samples here on Amazon[5], which lowers the efficiency as we need to spend more time preparing a large

[5]UltraGCN negative samples: 1500/800/500/200 on Gowalla/Yelp2018/Amazon/Ml-1M.

number of negative samples per epoch. Among the baselines, UltraGCN exhibits the strongest performance because it approximates the infinite layers of message passing and constructs the user-user graphs to capture 2$^{\text{nd}}$-order connectivity. LightGCN and NGCF perform better than MF since they inject the collaborative effect directly through message-passing. To align the setting with GTN, we increase the embedding size d^0 to 256 following [103][6] and observe the consistent superiority of our model over GTN in Table 4.3. This is because in GTN [103], the edge weights for message-passing are still computed based on node embeddings that implicitly encode noisy collaborative signals from unreliable interactions. Conversely, our CAGCN* directly alleviates the propagation on unreliable interactions based on its CIR value, which removes noisy interactions from the source.

4.5.3 Efficiency Comparison

As recommendation models will eventually be deployed in user-item data of real-world scale, it is crucial to compare the efficiency of the proposed CAGCN(*) with other baselines. To guarantee a fair comparison, we use a uniform code framework implemented ourselves for all models and run them on the same machine with Ubuntu 20.04 system, AMD Ryzen 9 5900 12-Core Processor (3.0 GHz), 128 GB RAM and GPU NVIDIA GeForce RTX 3090. We report the Recall@20 on Yelp and NDCG@20 on Loseit achieved by the best CAGCN(*) variant based on Table 4.2. We track the performance and the training time per 5 epochs.

In Figure 4.4(a)-(b), CAGCN achieves higher performance than LightGCN in less time. We hypothesize that for each user, its neighbors with higher interactions with its whole neighborhood would also have higher interactions with its interacted but unobserved neighbors. Then as CAGCN aggregates more information from these observed neighbors that have higher interactions with the whole neighborhood, it indirectly enables the user to aggregate more information from its to-be-predicted neighbors.

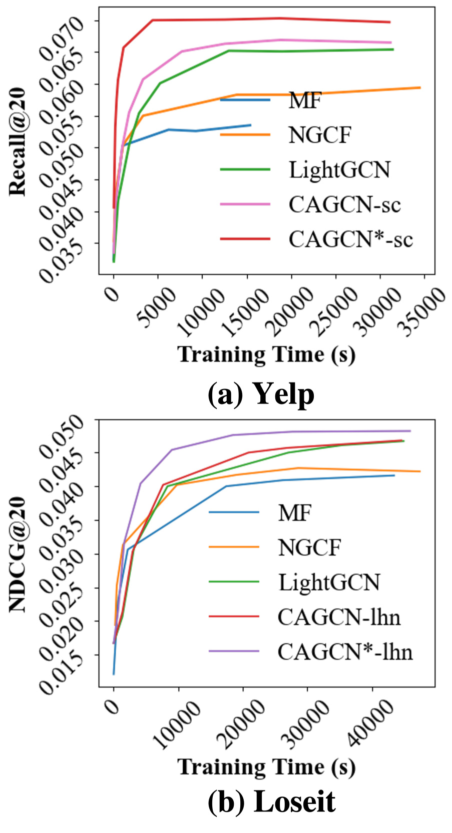


Figure 4.4: Training time (s) of different models.

[6]As the user/item embedding is a significant hyperparameter, it is crucial to ensure the same embedding size when comparing models; thus, we separately compare against GTN using their larger embedding size.

Table 4.4: Average Rank-Biased Overlap (RBO) of the ranked neighbor lists between training (i.e., $\mathcal{N}_u^1$) and testing/full (i.e., $\widehat{\mathcal{N}}_u^1$ and $\mathcal{N}_u^1 \cup \widehat{\mathcal{N}}_u^1$, respectively) dataset over all nodes $u \in \mathcal{U}$.

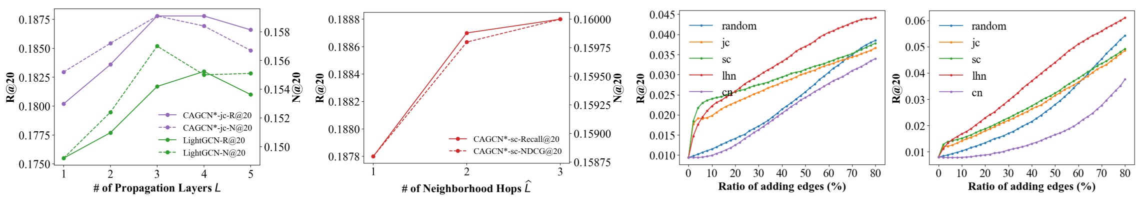
Metric	**Gowalla**		**Yelp**		**Ml-1M**	
	Train-Test	Train-Full	Train-Test	Train-Full	Train-Test	Train-Full
JC	0.604±0.129	0.902±0.084	0.636±0.124	0.897±0.081	0.848±0.092	0.978±0.019
SC	0.611±0.127	0.896±0.084	0.657±0.124	0.900±0.077	0.876±0.077	0.983±0.015
LHN	0.598±0.121	0.974±0.036	0.578±0.100	0.976±0.029	0.845±0.082	0.987±0.009
CN	0.784±0.120	0.979±0.029	0.836±0.100	0.983±0.023	0.957±0.039	0.995±0.006

To verify the above hypothesis, we define the to-be-predicted neighborhood set of user u in the testing set as $\widehat{\mathcal{N}}_u^1$ and for each neighbor $j \in \mathcal{N}_u^1$, calculate its CIR $\widehat{\phi}_u^{\widehat{L}}(j)$ with nodes in $\widehat{\mathcal{N}}_u^1$. Then we compare the ranking consistency among CIRs calculated from training neighborhoods (i.e., $\phi_u(j)$), from testing neighborhoods (i.e., $\widehat{\phi}_u(j)$) and from full neighborhoods (we replace $\widehat{\mathcal{N}}_u^1$ with $\mathcal{N}_u^1 \cup \widehat{\mathcal{N}}_u^1$ in Eq. (4.5)). Here we respectively use four topological metrics (JC, SC, LHN, and CN) to define f and rank the obtained three lists. Then, we measure the similarity of the ranked lists between Train-Test and between Train-Full by Rank-Biased Overlap (RBO) [130]. The averaged RBO values over all nodes $v \in \mathcal{V}$ on three datasets are shown in Table 4.4. It is clear that the RBO values on all these datasets are beyond 0.5, which verifies our hypothesis. The RBO value between Train-Full is always higher than the one between Train-Test because most interactions are in the training set.

Moreover, by combining two views of propagations, one from CAGC and one from LightGCN, CAGCN* achieves even higher performance with even less time. This is because keeping aggregating more information from neighbors with higher CIR (as CAGCN does) would prevent each user from aggregating information from his/her other neighbors. In addition, we report for the first time that our best CAGCN* variant achieves the best performance of LightGCN on each dataset in Table 4.5. We also report the preprocessing time for pre-calculating the CIR matrix $\mathbf{\Phi}$ for our model to avoid any bias. We could see that even considering the preprocessing time, it still takes significantly less time for CAGCN* to achieve the same best performance as LightGCN, which highlights the broad prospects to deploy CAGCN* in real-world recommendations.

Table 4.5: Efficiency comparison of CAGCN* with LightGCN. For a fair comparison, we track the first time CAGCN* achieves the best performance of LightGCN. CAGCN* achieves a significant efficiency boost over LightGCN, especially considering the training time.

Model	**Stage**	**Gowalla**	**Yelp**	**Amazon**	**Ml-1M**	**Loseit**	**News**
LightGCN	Training	16432.0	28788.0	81976.5	18872.3	39031.0	13860.8
CAGCN*	Preprocess	167.4	281.6	1035.8	33.8	31.4	169.0
	Training	2963.2	1904.4	1983.9	11304.7	10417.7	1088.4
	Total	3130.6	2186.0	3019.7	11338.5	10449.1	1157.4
Improve	Training	82.0%	93.4%	97.6%	40.1%	73.3%	92.1%
	Total	80.9%	92.4%	96.3%	39.9%	73.2%	91.6%



(a) Gowalla - L (b) Gowalla - $\widehat{L}$ (c) Amazon (d) Yelp2018

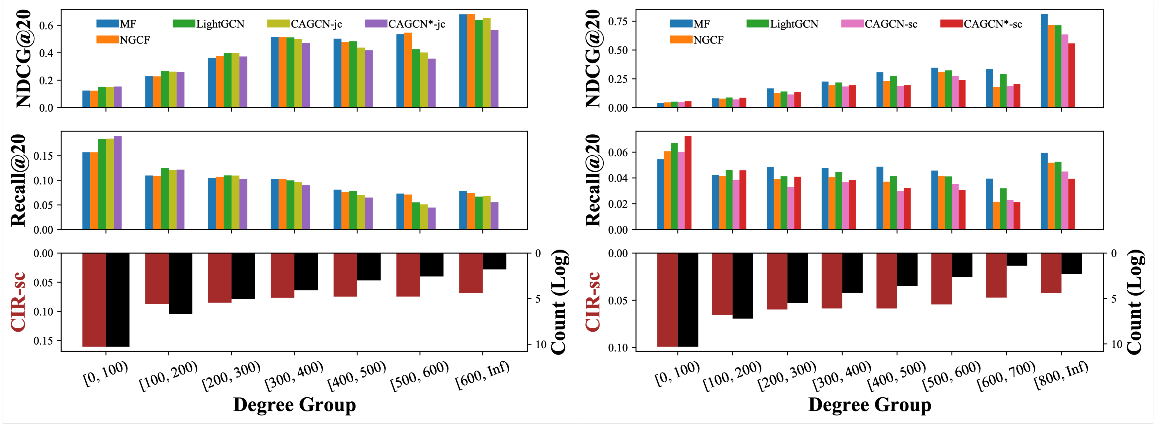
Figure 4.5: In (a)-(b), the performance first increases since we capture higher-layer neighborhood information and higher-hop topological interaction in calculating CIR as $L, \widehat{L}$ increase from 1 to 3. However, the performance decreases in (a) as L increases due to over-smoothing. In (c)-(d), we add the global top edges directly (rather than cycle each node) according to their CIR.

4.5.4 Further Probe

Impacts of propagation layers L and neighborhood hops $\widehat{L}$. Figure 4.5(a)-(b) visualize the performance of CAGCN* and LightGCN when the propagation layer L in Eq. (4.2) and the neighborhood hop $\widehat{L}$ in Eq. (4.5) increase. In (a), the performance first increases as L increases from 1 to 3 due to the incorporation of high-layer neighborhood information and then decreases due to over-smoothing. More importantly, our CAGCN* is always better than LightGCN at all propagation layers. In (b), the performance consistently increases as the number of neighborhood hops increases because we are allowed to consider even more higher topological interactions among each node's neighborhood in computing CIR.

Adding edges globally according to CIR. Figure 4.5(c)-(d) visualize the performance change when we add edges randomly and according to CIR. Unlike Figure 4.2 where we add edges by cycling each node, here we directly select the global top edges regardless of each center node according to their CIR and then evaluate the LightGCN with the pre-trained user-item embeddings. In the first stage, we observe a similar trend that adding edges according to JC, SC, and LHN leads to faster performance gain. However, since we don't cycle over each node, we would keep adding so many edges with larger CIR to the same node, which fails to bring performance gain anymore and hence cannot maximize our performance benefit under the node-centric evaluation metric.

Performance grouped by node degrees. Here, we group nodes in Gowalla by degree and visualize the average performance of each group in Figure 4.6(a). Comparing non-graph-based models (e.g., MF), graph-based models (e.g., LightGCN, CAGCN(*)) achieve higher performance for lower degree nodes $[0, 300)$ while lower performance for higher degree nodes $[300, \text{Inf})$. Since node degree follows the power-law distribution [131], the average performance of graph-based models is still higher than MF. On the one hand, graph-based models leverage neighborhood to augment the weak supervision for low-degree nodes. On the other hand, they introduce noisy interactions for higher-degree nodes. It is also interesting to see the opposite performance trends under different evaluation metrics: NDCG prefers high-degree nodes, while recall prefers low-degree nodes. This indicates that different evaluation metrics have different sensitivity to node degrees, and an unbiased node-centric evaluator is desired. To demonstrate the generality of our observation in Figure 4.6(a), we further perform exactly the same analysis on Yelp shown in Figure 4.6(b) and derive almost the same insights: 1) Graph-based recommendation models achieve higher performance than non-graph-based ones for lower degree nodes; 2) the opposite performance trends between NDCG and Recall indicates that different evaluation metrics have different levels of sensitivity to node degrees.



(a) Gowalla **(b) Yelp**

Figure 4.6: Performance with respect to node degree on Gowalla and Yelp

4.6 Conclusion

In this chapter, we find that message-passing captures a collaborative effect by leveraging interactions between neighborhoods. The strength of the captured collaborative effect depends on the embedding similarity, the weight of paths, and the contribution of each propagation layer. To determine whether the captured collaborative effect would benefit the prediction of user preferences, we propose the Common Interacted Ratio (CIR) and empirically verify that leveraging collaborations from neighbors with higher CIR contributes more to users' ranking. Furthermore, we propose CAGCN(*) to selectively aggregate neighboring nodes' information based on their CIRs. We further define a new type of isomorphism, bipartite-subgraph-isomorphism, and prove that our CAGCN* can be more expressive than 1-WL in distinguishing subtree(subgraph)-isomorphic yet non-bipartite-subgraph-isomorphic graphs. Experimental results demonstrate the advantages of the proposed CAGCN(*) over other baselines. Specifically, CAGCN* outperforms the most representative graph-based recommendation model, LightGCN [79], by around 10% in Recall@20 but also achieves roughly more than 80% speedup. In the future, we will explore the imbalanced performance improvement among nodes in different degree groups as seen in Figure 4.6, especially from the perspective of GNN fairness [6, 9].

4.7 Appendix

4.7.1 Graph Topological Metrics for CIR

We demonstrate that by configuring different f and $\widehat{L}$, $\phi_u^{\widehat{L}}(j)$ can express many graph similarity metrics.

$$\phi_u^{\widehat{L}}(j) = \frac{1}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \sum_{l=1}^{\widehat{L}} \beta^{2l} \sum_{P_{ji}^{2l} \in \mathscr{P}_{ji}^{2l}} \frac{1}{f(\{\mathcal{N}_k^1 | k \in P_{ji}^{2l}\})} \tag{4.9}$$

- **Jaccard Similarity (JC) [125]:** The JC score measures the similarity between neighborhood sets as the ratio of the intersection of two neighborhood sets to the union of these two sets:

$$\text{JC}(i,j) = \frac{|\mathcal{N}_i^1 \cap \mathcal{N}_j^1|}{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|} \tag{4.10}$$

Let $\widehat{L} = 1$ and set $f(\{\mathcal{N}_k^1 | k \in P_{ji}^2\}) = |\mathcal{N}_i^1 \cup \mathcal{N}_j^1|$, then we have:

$$\phi_u^1(j) = \frac{1}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \beta^2 \sum_{P_{ji}^2 \in \mathscr{P}_{ji}^2} \frac{1}{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|} = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \frac{|\mathcal{N}_i^1 \cap \mathcal{N}_j^1|}{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|} = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \text{JC}(i,j) \tag{4.11}$$

- **Salton Cosine Similarity (SC) [124]:** The SC score measures the cosine similarity between the neighborhood sets of two nodes:

$$\text{SC}(i,j) = \frac{|\mathcal{N}_i^1 \cap \mathcal{N}_j^1|}{\sqrt{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|}} \tag{4.12}$$

let $\widehat{L} = 1$ and set $f(\{\mathcal{N}_k^1 | k \in P_{ji}^2\}) = \sqrt{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|}$, then we have:

$$\phi_u^1(j) = \frac{1}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \beta^2 \sum_{P_{ji}^2 \in \mathscr{P}_{ji}^2} \frac{1}{\sqrt{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|}} = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \frac{|\mathcal{N}_i^1 \cap \mathcal{N}_j^1|}{\sqrt{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|}} = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \text{SC}(i,j) \tag{4.13}$$

- **Common Neighbors (CN) [117]:** The CN score measures the number of common neighbors of two nodes and is frequently used for measuring the proximity between two nodes:

$$\text{CN}(i,j) = |\mathcal{N}_i^1 \cap \mathcal{N}_j^1| \tag{4.14}$$

Let $\widehat{L} = 1$ and set $f(\{\mathcal{N}_k^1 | k \in P_{ji}^2\}) = 1$, then we have:

$$\phi_u^1(j) = \frac{1}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \beta^2 \sum_{P_{ji}^2 \in \mathscr{P}_{ji}^2} 1 = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} |\mathcal{N}_i^1 \cap \mathcal{N}_j^1| = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \text{CN}(i,j) \tag{4.15}$$

Since CN does not contain any normalization to remove the bias of degree in quantifying proximity and hence performs worse than other metrics as demonstrated by our recommendation experiments in Table 4.2.

- **Leicht-Holme-Nerman (LHN) [115]:** LHN is very similar to SC. However, it removes the square root in the denominator and is more sensitive to the degree of node:

$$\text{LHN}(i, j) = \frac{|\mathcal{N}_i^1 \cap \mathcal{N}_j^1|}{|\mathcal{N}_i^1| \cdot |\mathcal{N}_j^1|} \tag{4.16}$$

Let $\widehat{L} = 1$ and set $f(\{\mathcal{N}_k^1 | k \in P_{ji}^2\}) = |\mathcal{N}_i^1| \cdot |\mathcal{N}_j^1|$, then we have:

$$\phi_u^1(j) = \frac{1}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \beta^2 \sum_{P_{ji}^2 \in \mathscr{P}_{ji}^2} \frac{1}{|\mathcal{N}_i^1| \cdot |\mathcal{N}_j^1|} = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \frac{|\mathcal{N}_i^1 \cap \mathcal{N}_j^1|}{|\mathcal{N}_i^1| \cdot |\mathcal{N}_j^1|} = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \text{LHN}(i, j) \tag{4.17}$$

We further emphasize that our proposed CIR is a generalized version of these four existing metrics and can be delicately designed to satisfy downstream tasks and datasets. We leave such exploration on the choice of f as one potential future work.

4.7.2 Experimental Setting

Baselines. We compare our proposed CAGCN(*) with the following baselines: **MF [107]:** Most classic collaborative filtering method equipped with the BPR loss; **NGCF [96]:** The first GNN-based collaborative filtering model; **LightGCN [79]:** The most popular GNN-based collaborative filtering model, which removes feature transformation and nonlinear activation; **UltraGCN [122]:** The first model approximating regularization weights by infinite layers of message passing, and leveraging higher-order user-user relationships; **GTN [103]:** This model leverages a robust and adaptive propagation based on the trend of the aggregated messages to avoid unreliable user-item interactions.

CAGCN(*)-variants. For CAGCN, $\gamma_i = \sum_{r \in \mathcal{N}_i^1} d_i^{-0.5} d_r^{-0.5}$ to ensure that the total edge weights for messages received by each node are the same as LightGCN. Therefore, Eq. (4.7) becomes:

$$\mathbf{e}_i^{l+1} = \sum_{j \in \mathcal{N}_i^1} \Big(\big(\sum_{r \in \mathcal{N}_i^1} d_i^{-0.5} d_r^{-0.5} \big) \frac{\mathbf{\Phi}_{ij}}{\sum_{k \in \mathcal{N}_i^1} \mathbf{\Phi}_{ik}} \Big) \mathbf{e}_j^l, \forall i \in \mathcal{V}. \tag{4.18}$$

For CAGCN*, $\gamma_i = \gamma$ as a constant controlling the trade-off between contributions from message-passing according to LightGCN and according to CAGC. Eq. (4.7) becomes:

$$\mathbf{e}_i^{l+1} = \sum_{j \in \mathcal{N}_i^1} \Big(\gamma \frac{\mathbf{\Phi}_{ij}}{\sum_{k \in \mathcal{N}_i^1} \mathbf{\Phi}_{ik}} + d_i^{-0.5} d_j^{-0.5} \Big) \mathbf{e}_j^l, \forall i \in \mathcal{V}, \gamma \in \{1, 1.2, 1.5, 1.7, 2.0\} \tag{4.19}$$

4.7.3 Graph Isomorphism

We review the concepts of subtree/subgraph-isomorphism [120].

Definition 5. ***Subtree-isomorphism:*** *$\mathcal{S}_u$ and $\mathcal{S}_i$ are subtree-isomorphic, denoted as $\mathcal{S}_u \cong_{subtree} \mathcal{S}_i$, if there exists a bijective mapping $h : \widetilde{\mathcal{N}}_u^1 \to \widetilde{\mathcal{N}}_i^1$ such that $h(u) = i$ and $\forall v \in \widetilde{\mathcal{N}}_u^1, h(v) = j, \mathbf{e}_v^l = \mathbf{e}_j^l$.*

Definition 6. ***Subgraph-isomorphism:*** *$\mathcal{S}_u$ and $\mathcal{S}_i$ are subgraph-isomorphic, denoted as $\mathcal{S}_u \cong_{subgraph} \mathcal{S}_i$, if there exists a bijective mapping $h : \widetilde{\mathcal{N}}_u^1 \to \widetilde{\mathcal{N}}_i^1$ such that $h(u) = i$ and $\forall v_1, v_2 \in \widetilde{\mathcal{N}}_u^1, e_{v_1 v_2} \in \mathcal{E}_{\mathcal{S}_u}$ iff $e_{h(v_1)h(v_2)} \in \mathcal{E}_{\mathcal{S}_i}$ and $\mathbf{e}_{v_1}^l = \mathbf{e}_{h(v_1)}^l, \mathbf{e}_{v_2}^l = \mathbf{e}_{h(v_2)}^l$.*

Corresponding to the backward($\Longleftarrow$) proof of Theorem 3, here we show two of such graphs $\mathcal{S}_u, \mathcal{S}'_u$, which are subgraph isomorphic but non-bipartite-subgraph-isomorphic. Assuming u and u' have exactly the same neighborhood feature vectors $\mathbf{e}$, then directly propagating according to 1-WL or even considering node degree as the edge weight as GCN [15] can still end up with the same propagated feature for u and u'. However, if we leverage JC to calculate CIR as introduced in Appendix 4.7.1, then we would end up with $\{(d_u d_{j_1})^{-0.5}\mathbf{e}, (d_u d_{j_2})^{-0.5}\mathbf{e}, (d_u d_{j_3})^{-0.5}\mathbf{e}\} \neq \{(d_{u'}^{-0.5} d_{j'_1}^{-0.5} + \widetilde{\boldsymbol{\Phi}}_{u'j'_1})\mathbf{e}, (d_{u'}^{-0.5} d_{j'_2}^{-0.5} + \widetilde{\boldsymbol{\Phi}}_{u'j'_2})\mathbf{e}, (d_{u'}^{-0.5} d_{j'_3}^{-0.5} + \widetilde{\boldsymbol{\Phi}}_{u'j'_3})\mathbf{e}\}$. Since g is injective by Lemma 1, CAGCN would yield two different embeddings for u and u'.

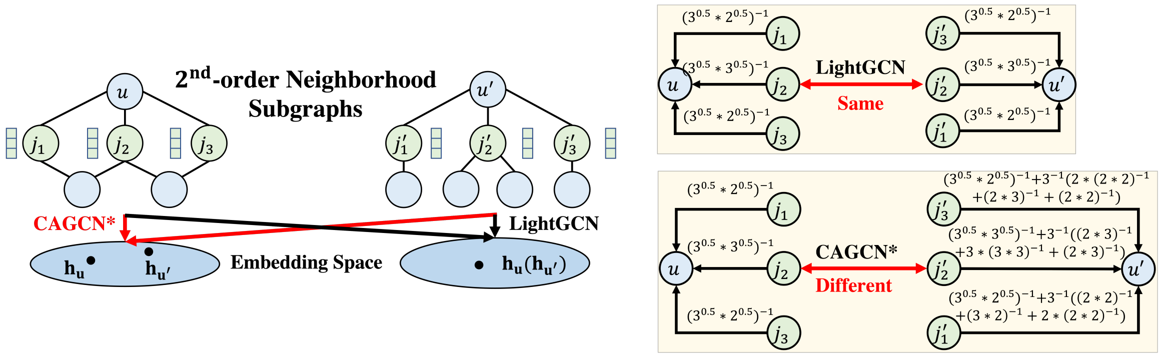


Figure 4.7: An example showing two neighborhood subgraph $\mathcal{S}_u, \mathcal{S}_{u'}$ that are subgraph-isomorphic but not bipartite-subgraph-isomorphic.

CHAPTER 5

Topology Issue: Overcoming the Noisy Topology Issue in Session-Recommendation

Session-based recommender systems (SRSs) predict the next items users will likely interact with by capturing their interests from historical activities. While most SRSs capture users' purchasing intentions locally within each session, capturing items' global information across different sessions is crucial in characterizing their general properties. Previous works capture this cross-session information by constructing a global graph and incorporating items' neighbor information. However, the incorporation of neighboring information cannot vary adaptively according to the unique intention of each session, and the constructed graphs consist of only one type of user-item interaction. To address this noisy topology issue when coupling the constructed graph with local sessions, we propose knowledge graph-based session recommendation with session-adaptive propagation. Specifically, we build a knowledge graph by connecting items with multi-typed edges to characterize various user-item interactions. Then, we adaptively aggregate items' neighbor information considering the user intention within the learned session context so that noisy neighboring information irrelevant to the current session would be removed. Experimental results demonstrate that equipping our constructed knowledge graph and session-adaptive propagation enhances existing session recommendation backbones by 10%-20%, and our best-performing model configuration exceeds existing baselines on average by 4%. Moreover, we provide an industrial case study showing our proposed framework achieves 2% performance boost over an existing well-deployed model at The Home Depot e-platform and visualize the attentions learned for the same item across different sessions, verifying the denoising effect of our proposed session-adaptive propagation.

5.1 Introduction

Transformer-based models have shown state-of-the-art performance for session-based recommender systems (SRS) by leveraging attention mechanisms and deep learning capabilities [132, 133]. While transformers are adept at capturing local session data and individual item preferences [134, 135, 136, 137], they are limited in capturing global transitional patterns among items [138, 139, 140]. This limitation has sparked research into hybrid graph-based SRSs that combine the strengths of transformers with Graph Neural Networks (GNNs) to capture both local and global dependencies [139, 138, 140, 137]. For example, in Figure 5.1(b), borrowing the transitional information between flower and lopper in Figure 5.1(a) characterizes the intent of session A as decorating garden and hence increases the probability of predicting next item to be watering can.

However, the existing hybrid models face two significant challenges. Firstly, the lack of full connectivity between GNN and transformer models limits their ability to capture the session dynamics [141, 142, 143,

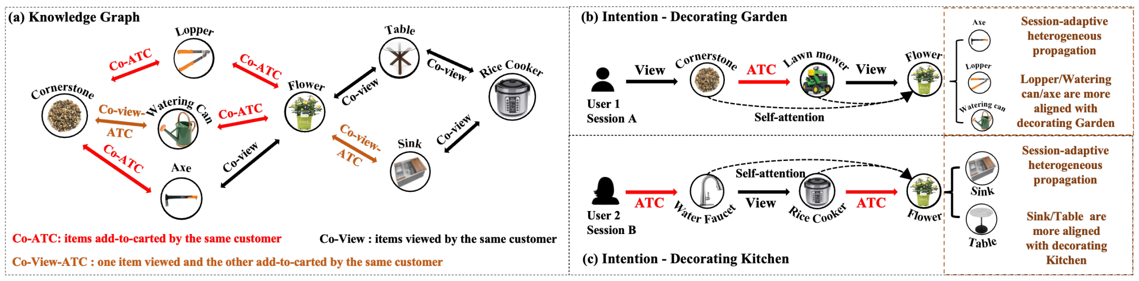

(a) Knowledge Graph
Lopper
Co-ATC
Co-ATC
Table
Co-view
Rice Cooker
Cornerstone
Co-view-ATC
Watering Can
Co-ATC
Flower
Co-view
Co-ATC
Axe
Co-view
Co-view-ATC
Sink
Co-view
Co-ATC: items add-to-carted by the same customer
Co-View : items viewed by the same customer
Co-View-ATC : one item viewed and the other add-to-carted by the same customer
(b) Intention - Decorating Garden
User 1
Session A
View
Cornerstone
ATC
Lawn mower
View
Flower
Self-attention
Axe
Lopper
Watering can
Session-adaptive heterogeneous propagation
Lopper/Watering can/axe are more aligned with decorating Garden
User 2
Session B
ATC
Water Faucet
Self-attention
View
Rice Cooker
ATC
Flower
Sink
Table
Session-adaptive heterogeneous propagation
Sink/Table are more aligned with decorating Kitchen
(c) Intention - Decorating Kitchen

Figure 5.1: Since the user in Session A (b) intends to decorate the garden, while the user in Session B (c) intends to decorate the kitchen, the corresponding neighbors from the knowledge graph (a) are different for the same flower. By our proposed session-adaptive propagation, the flower aggregates more information from the lopper/watering can in (b) while more information from the sink/table in (c).

144]. While GNNs excel at capturing item-item relationships, they often struggle to incorporate the broader session context, and the learned item representations lack full awareness of the session context. For example, in Figure 5.1, although both sessions A and B have involved the flower, their intentions are quite different: one for decorating the garden while the other for decorating the kitchen. In contrast to blindly aggregating all neighbors' information to the flower without considering the session context [140, 139, 138], our approach learns session-aware item embeddings by selectively propagating information from relevant neighbors based on the current session. For instance, when determining which neighbors' information is aggregated to the flower, we recommend incorporating the lopper and watering can when decorating the garden in (b), while the sink and the table when decorating the kitchen in (c).

Secondly, the global transitional patterns among items in these hybrid models are typically constructed based on one type of interaction, such as co-purchase patterns [139, 140, 143, 144]. However, on e-commerce platforms [142], items could form multiple relationships: substitution items are typically co-viewed and complementary items are typically co-add-to-carted(co-ATC) [145, 146] by the same user. Uniformly using neighbor information may result in the dilution of diverse relationships among items and consequentially lead to unsatisfactory recommendations for users. To overcome the above two challenges, we propose a knowledge graph-based SRS with session-adaptive propagation. Our contributions are summarized as follows:

- For the first challenge, we propose a session-adaptive graph propagation to adaptively aggregate items' neighbor information based on the session contexts obtained by the transformer model.
- For the second challenge, a knowledge graph is constructed based on item co-relations with users, and the heterogeneous graph transformer [147] is used to aggregate neighborhoods based on relations.
- We experimentally verify the proposed framework in enhancing existing SBRs, compare the performance improvement caused by different types of edges, and verify the session-adaptive propagation by visualizing the changing attentions of the same item in aggregating neighbors in different sessions.

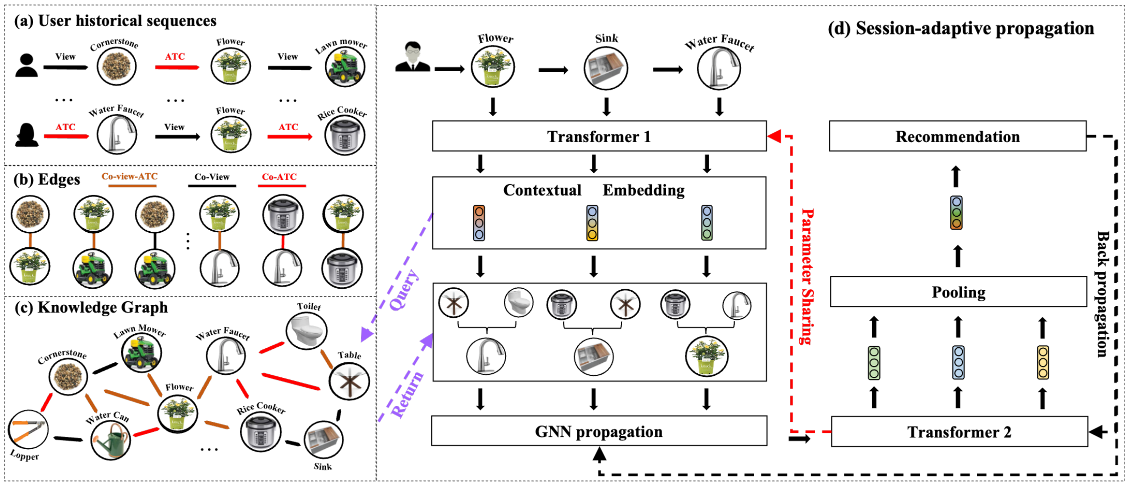


Figure 5.2: In (a)-(c), we first extract three types of edges from historical sessions to construct item knowledge graph. Then in (d), we forward the given session through the 1^{st} transformer layer to obtain items' contextual embeddings, which are used for query-relevant neighbors for GNNs to perform graph propagation. The propagated item embeddings are fed into the 2^{nd} transformer with a pooling layer afterward to obtain session embedding for the recommendation.

5.2 Related Work

Graph-based Session Recommendation. Graph, as a general data structure representing relations of entities, has been widely adopted to assist session recommendation. Previous works explore global transitional patterns across different sessions by querying the global item graph [138, 139, 140]. [139] designs a global context-enhanced inter-session relation encoder to capture the inter-session item-wise dependencies. [138] constructs the dual session graph to model the pair-wise transition relationship between items based on the global connections. [140] constructs the global graph by merging all individual session graphs. The very recent work KSTT [148] resorts to an item-category knowledge graph for session recommendation. However, the proposed models in all the above works learn item embeddings from the global graph without any session-tailored modification. Only GCE-GNN [144] and GCARM [143] consider session adaptation in aggregating neighbors' information. However, GCE-GNN quantifies neighbor importance based on their similarity to the whole session without differentiating central items. GCARM treats all transitions similarly without distinguishing different types of interactions. To handle these two issues, we design a session-adaptive propagation to query neighbors based on session contexts and interaction types, the effectiveness of which is verified in Section 5.4.5. Note that in this work, although three types of item-item co-interaction edges are considered, the constructed knowledge graph has only one type of node, the item. We leave the inclusion of different node types as one future work, such as adding user nodes, which could provide a way to personalize the session recommendations.

5.3 The Proposed Framework

Our framework, as illustrated in Figure 11.2, comprises of a GNN that obtains item embeddings by adaptively aggregating information from neighboring items based on the target session, and a transformer model that acquires session embeddings for predicting the next item. In the subsequent sections, we first explain the construction of the item knowledge graph, and then the GNN-based message passing model and the transformer-based prediction model.

5.3.1 Item Knowledge Graph Construction

As items typically exhibit two types of correlations, substitution and complementary [138], we extract three distinct types of edges, as depicted in Figure 11.2(a)-(c), by examining whether two items co-occur within the same session: co-view, co-ATC, and co-view-ATC edges. For instance, in the first session shown in Figure 11.2(a), the user first views the cornerstone, then adds the flower to the cart, and subsequently views the lawn mower. This sequence forms three edges: the co-view edge between the cornerstone and the lawn mower, the co-view-ATC edge between the cornerstone and the flower, and the co-view-ATC edge between the flower and the lawn mower. More formally, we define the edge weight from item v_j to v_i of type co-t_1-t_2 as follows:

$$w_{i \leftarrow j}^{(t_1,t_2)} = \frac{\sum_{m=1}^{M} \mathbb{1}(v_i, v_j \in \mathcal{S}_m,\ \tau(v_i, \mathcal{S}_m) = t_1,\ \tau(v_j, \mathcal{S}_m) = t_2)}{\sum_{m=1}^{M} \mathbb{1}(v_i \in \mathcal{S}_m,\ \tau(v_i, \mathcal{S}_m) = t_1)}, \tag{5.1}$$

where $\mathcal{S}_m$ is the m^{th}-session, M is the total number of sessions in the historical data, and $t_1, t_2 \in \mathcal{T} = \{\text{view}, \text{ATC}\}$ and $\mathbb{1}$ is an indicator function. Specifically, $\mathbb{1}(v_i \in \mathcal{S}_m, v_j \in \mathcal{S}_m, \tau(v_i, \mathcal{S}_m) = t_1, \tau(v_j, \mathcal{S}_m) = t_2) = 1$ if **(a)** both v_i, v_j belong to the m^{th} session $\mathcal{S}_m$, **(b)** user interacts with v_i following type t_1 and **(c)** interacts with v_j following type t_2 in $\mathcal{S}_m$ and otherwise 0. Note that, we normalize edge weights based on the degree of the head node to avoid popular items from dominating the message-passing of GNNs [60, 149].

Based on our empirical experiments, it was observed that the obtained graph may contain items with over a hundred neighbors. Therefore, in order to mitigate the heavy computational issue [50] and prevent over-smoothing [2, 150] during the message passing, we sparsify the graph by retaining only the top-K neighbors for each neighborhood type, based on the acquired edge weights. The statistics of the constructed networks are provided in Table 5.1.

5.3.2 Session-adaptive Propagation

As mentioned earlier, a straightforward approach to utilize item graphs in SBRs is to employ a GNN model to obtain item embeddings from the graph. These acquired embeddings can then be utilized in a transformer-based model to predict the next item based on the target session [138, 139, 140].

However, seen in Figure 5.1, a key limitation of this architecture is that the contribution of each neighboring item remains unchanged in obtaining the center item's embedding, regardless of the session from which the item originates. As such, the item embeddings obtained from the GNN is unaware of the contextual information specific to the target session. This approach is sub-optimal since the purchasing intention associated with the same item naturally differs across different sessions, thereby necessitating changes in the selection of compatible neighbors based on the session-specific intention. Motivated by this observation, we propose a session-adaptive propagation that dynamically propagates neighbor information according to the item's unique context within each session.

To implement the above idea, given the target session, we input the initial item embedding within the session into a transformer, resulting in the generation of its session-aware representation. Subsequently, this obtained embedding is utilized to determine the edge weights for the GNN model. Finally, following the message-passing process, the updated item embeddings are once again fed into another transformer model to make the final prediction. More formally, we first obtain item initial embeddings $\mathbf{E}^1$ by integrating item meta-attributes such as item title and category. Then, given an item v_i in the session $\mathcal{S}_m$, we use the item initial embeddings $\mathbf{E}^1$ to obtain its contextual embedding $\mathbf{c}_i^m$:

$$\mathbf{c}_i^m = transformer_1(v_i, \mathbf{E}^1, \mathcal{S}_m), \quad \forall v_i \in \mathcal{S}_m \tag{5.2}$$

where $transformer_1$ is the 1st transformer. With the contextual embedding $\mathbf{c}_i^m$ as the query, inspired by [147], we perform heterogeneous graph transformer-based propagation to adaptively aggregate v_i's neighbors' information relevant to the current session intention:

$$\mathbf{h}_i^{m,l} = \frac{1}{|\mathcal{T}|^2} \sum_{t \in \mathcal{T} \times \mathcal{T}} ||_{h=1}^{H} \sum_{v_j \in \mathcal{N}_i^t} \alpha_{i \leftarrow j}^{h,l,t,m} \mathbf{V}^{h,l,t} \mathbf{h}_j^{m,l-1} \tag{5.3}$$

$$\alpha_{i \leftarrow j}^{h,l,t,m} = \frac{(\mathbf{Q}^{h,l,t}\mathbf{h}_i^{m,l-1})^\top (\mathbf{K}^{h,l,t}\mathbf{c}_j^m)}{\sqrt{d/H}}, \forall v_i \in \mathcal{S}_m, \mathcal{S}_m \in \mathscr{S}, \tag{5.4}$$

where $\alpha_{i \leftarrow j}^{h,l,t,m}$ denotes the graph attention from item v_j to v_i under the head h, edge type t at layer l. $\mathbf{Q}^{h,l,t}, \mathbf{K}^{h,l,t}, \mathbf{V}^{h,l,t}$ represent the query, key, and value matrix at the head h, edge type t, layer l of graph attention. H is the total number of heads. After L layers graph transformer-based propagation, we obtain the final item embeddings $\mathbf{h}_i^{m,L}$. *Since different session contexts provide different contextual embeddings* $\mathbf{c}_i^m \neq \mathbf{c}_i^{m'}$*, the calculated attention coefficients would also be different, i.e.,* $\alpha_{i \leftarrow j}^{h,l,t,m} \neq \alpha_{i \leftarrow j}^{h,l,t,m'}$*. Then* $\mathbf{h}_i^{m,L}$ *and* $\mathbf{h}_i^{m',L}$*) would be different, and they would only include the neighborhood information that is relevant to the item's unique intention provided by the corresponding session context* $\mathcal{S}_m(\mathcal{S}_{m'})$*.*

The effectiveness of the above proposed session-adaptive propagation in learning neighborhood attention based on the session context has been verified in Section 5.4.5.

Table 5.1: Statistics of datasets used for experiments and their corresponding knowledge graphs.

Dataset	**Train/Val/Test Seqs**	**# Edges**	**# Nodes**	**Sparse Seqs**	**Meta data**
Diginetica	675,673/43,541/68,571	1,576,571	123,273	13,867	Title/Category/Price
Yoochoose 1/64	369,142/45,864/55,898	1,025,176	52,739	15,385	–
The Home Depot (THD)	3,169,140/672,873/672,873	4,162,712	1,317,149	227,941	Title/Category/Brand/Color /Manufacturer/Class/Department

5.4 Experiments

5.4.1 Experimental Setup

Datasets. We conduct experiments on the following datasets:

- **Diginetica**[1] comes from CIKM Cup 2016. We follow the same pre-processing as [137]: sessions in the last and second-to-last week are used as testing and validation data. We filter out sessions of length less than 1 and items appearing less than 5 times. We extract item co-purchase edges from all sessions in train-purchase.csv and item co-view edges from only training/validation sessions in train-item-view.csv to avoid data leakage. The meta-attribute includes item title, category, and price.

- **Yoochoose 1/64**[2] comes from the RecSys Challenge 2015. Sessions on the last and second last day are used as testing and validation data. We filter out sessions of length less than 1 and items appearing less than 5 times. We extract item co-purchase edges from all sessions in yoochoose-buys.dat and item co-view edges from only training/validation sessions in yoochoose-clicks.dat to avoid data leakage. Since items in this dataset have only one category attribute and it represents different meanings, e.g., 1-12 for item real categories, 'S' for special offer, and 8-10 digit numbers for item brand, we only use item ID to demonstrate the effectiveness of session-adaptive heterogeneous propagation.

- **THD is a real industrial-level dataset** from The Home Depot, the largest home improvement retailer in the USA. We sample 3,169,140/672,873/672,873 Add-to-Cart (ATC) sessions chronologically for constructing the train/valid/testing data. The multiplex graph is constructed by extracting co-view, co-ATC, and co-view-ATC edges. Items in this dataset have 7 meta-attributes: product title, hierarchical categories (i.e., L1, L2, L3, Leaf), brand, manufacturer, color, department, and class name.

Statistics of the three datasets are summarized in Table 5.1. Note that sparse sessions refer to the ones containing items appearing less than 5 times in the whole dataset. These sessions are used to evaluate our framework on sessions including cold-start items.

[1]https://competitions.codalab.org/competitions/11161
[2]https://www.kaggle.com/datasets/chadgostopp/recsys-challenge-2015

Item embedding initialization. We initialize item embeddings based on their meta-attributes [93]. For each numerical attribute, we directly embed it as a real-valued number. For each categorical attribute, we initialize a unique learnable embedding matrix. For textual attributes such as title and description, we first construct the token embedding matrix and then the title/description embedding is computed by mean pooling over the embeddings of corresponding tokens in that sentence. Note that pre-trained NLP models are not preferred here to avoid capturing noisy semantic signals. For example, even though silver sinks and creamy white stones share semantic-similar colors, they are essentially purposed for decorating different rooms. We empirically observe that utilizing this token-based embedding avoids capturing noisy semantic signals since they correspond to different tokens. We concatenate different types of embeddings to form the final item meta-embeddings and feed them into the transformer model.

Backbones. Note that our constructed knowledge graph and the proposed session-adaptive propagation can be applied to enhance any embedding-based SRS. To demonstrate this, we select three fundamentally different but representative backbones and equip them with our framework: **GRU4Rec** [132]: The very first model leveraging RNNs to characterize item sessions for session recommendation. **SASRec** [135]: The very first model leveraging the self-attention from the transformer to draw context from all user-item interactions in the same session. **KGHT** [142]: A graph-based model constructing the item relational graph and leveraging graph attention to capture item relations for recommendations. Since this work is not initially designed for session recommendation, we modify it to align with our problem setting. Equipping each of the above three backbones with the item meta-attribute embedding layer and the session-adaptive propagation layer, we end up with 11 model configurations. We name each new configuration by combining its backbone and the equipped techniques, e.g., GRU4Rec$_m$ denotes the backbone of GRU4Rec with item meta-attribute embedding layer, SASRec$_{m*}$ denotes the backbone of SASRec with both item meta-attribute and ID, and KGHT$_q$ denotes the backbone of KGHT equipped with session-adaptive propagation layer.

Evaluation Metric and Implementation Details. Two popular metrics: NDCG@K and MRR@K are adopted for evaluation. We set the default value of K as 10 and report the average of NDCG@10(N@10) and MRR@10(M@10) over all sequences in the test set. We assign a dedicated embedding layer for each attribute, and the embedding dimension is determined based on the total number of distinct tokens of the corresponding attribute vocabulary. To ensure a fair comparison, we tune the following hyperparameters for each model individually: the dimension of ID embedding layer $\{64, 128, 256\}$, the number of layers for self-attention and graph propagation $\{1, 2, 3\}$, the number of attention heads $\{1, 4, 8\}$, the dimension of hidden embeddings $\{100, 512\}$, learning rate $\{1e^{-4}, 1e^{-3}, 1e^{-2}\}$, the L2 penalty $\{0, 5e^{-4}\}$, training epochs $\{100, 200\}$, the batch size $\{100, 512\}$, dropout ratio $\{0.1, 0.25\}$ across all models. We save the model performing best on validation sessions and evaluate it on testing sessions.

Table 5.2: An architecture comparison of different backbones.

Backbone	**ID**	**Meta-attribute**	**Graph**	**Session-adaptive propagation**	**Multi-task learning**
GRU4Rec	✓	✗	✗	✗	✗
GRU4Rec$_m$	✗	✓	✗	✗	✗
GRU4Rec$_m*$	✓	✓	✗	✗	✗
M-GRU4Rec$_m*$	✓	✓	✗	✗	✓
SASRec	✓	✗	✗	✗	✗
SASRec$_m$	✗	✓	✗	✗	✗
SASRec$_m*$	✓	✓	✗	✗	✗
M-SASRec$_m*$	✓	✓	✗	✗	✓
KGHT	✓	✗	✓	✗	✗
KGHT$_m$	✗	✓	✓	✗	✗
KGHT$_m*$	✓	✓	✓	✗	✗
KGHT$_q$	✓	✗	✓	✓	✗
KGHT$_{qm*}$	✓	✓	✓	✓	✗
M-KGHT$_{qm*}$	✓	✓	✓	✓	✓

5.4.2 Model Configuration Analysis

To demonstrate the effectiveness of the proposed knowledge graph and session-adaptive propagation, we compare three backbones GRU4Rec, SASRec, and KGHT with their corresponding enhanced versions in Table 5.3. For brevity, we represent any of the three backbones as X in the following text:

- Compared with X, X_{m*} incorporates item meta-attributes and improves the performance by around $12\% - 19\%$ on average. This is because ID-based embedding only captures the topological proximity of each item with respect to all other items and cannot provide generalizability, especially when items' topological information is noisy/sparse. Leveraging meta-attributes alleviates this issue by transferring the learned information among items sharing the same meta-attribute. Since THD has more abundant types of well-curated meta-attributes than the ones of Diginetica, as evidenced in Table 5.1, X_{m*} achieves an even larger performance gain over X on THD than on Diginetica.

- Compared with X, X_m achieves comparable and sometimes slightly worse performance on Diginetica, e.g., 9.55% for GRU4Rec while 9.27% for GRU4Rec$_m$ in MRR@10. This is because solely relying on meta-attribute to represent items may lose topological information hidden in the sessions. Two items sharing the same meta-attributes will be encoded exactly the same, even though they may be involved in significantly different sessions. This is also evidenced by the better performance of X_m* than X_m after combining both item ID and item meta-attribute. Different from Diginetica, X_m always achieves higher performance than X on THD because more abundant meta-attributes there enable the concatenated embeddings to be more unique and hence can somewhat mimic the function of Item ID in capturing topological information embedded in the sessions.

Table 5.3: Performance comparison (%) of utilizing meta-attribute embedding layer and session-adaptive heterogeneous propagation layer. The best and runner-up are in **bold** and underlined. Note that N(M)@10 represents NDCG(MRR)@10.

		Diginetica				Yoochoose 1/64				THD				
Backbone		N@10		M@10		N@10		M@10		N@10		M@10		**Increase** ($\uparrow$)
		Full	Sparse	Full	Sparse	Full	Sparse	Full	Sparse	Full	Sparse	Full	Sparse	
RNN	GRU4Rec	12.66	11.04	9.55	8.19	32.77	28.32	26.52	22.81	14.40	7.66	12.34	6.53	–
	GRU4Rec$_m$	12.38	10.62	9.27	7.77	–	–	–	–	17.09	10.42	14.19	8.41	10.55%
	GRU4Rec$_m*$	13.01	11.47	9.73	8.37	–	–	–	–	18.13	11.33	15.23	9.34	18.88%
Trans-former	SASRec	14.70	13.00	11.11	9.57	33.68	28.59	27.16	22.87	15.27	8.11	12.97	6.83	–
	SASRec$_m$	14.68	12.79	11.08	9.28	–	–	–	–	17.87	11.11	14.87	9.01	11.94%
	SASRec$_m*$	15.31	13.68	11.60	10.12			–	–	18.53	11.72	15.53	9.64	18.28%
Graph	KGHT	17.73	15.99	13.33	11.74	$\underline{34.81}$	$\underline{30.14}$	$\underline{28.04}$	$\underline{24.10}$	16.59	9.29	14.02	7.74	–
	KGHT$_m$	17.70	15.78	13.27	11.52	–	–	–	–	18.47	11.68	15.49	9.59	8.45%
	KGHT$_m*$	17.81	$\underline{16.88}$	13.38	$\underline{12.49}$	–	–	–	–	$\underline{18.78}$	$\underline{11.89}$	**15.74**	$\underline{9.79}$	11.59%
	KGHT$_q$	$\underline{18.64}$	16.74	$\underline{13.97}$	12.28	**35.80**	**31.71**	**28.80**	**25.21**	17.30	10.20	14.45	8.35	4.25%
	KGHT$_{qm*}$	**18.93**	**17.16**	**14.22**	**12.65**	–	–	–	–	**18.83**	**12.17**	$\underline{15.67}$	**9.91**	14.10%

X: Backbone X using item ID but no meta-attributes but no ID; $\mathbf{X}_m$: Backbone X using item meta-attributes but no ID; $\mathbf{X}_{m*}$: Backbone X using both item meta-attributes and ID; $\mathbf{X}_q$: Backbone X using Graph and Session-adaptive propagation; $\mathbf{X}_{qm*}$: Backbone X using Graph, Session-adaptive propagation, and meta-attribute.

- Comparing among different backbones, graph-based models achieve higher performance than non-graph-based ones, which aligns with the notion that incorporating global information across different sessions is conducive to the recommendation [138, 139, 140]. More specifically, KGHT$_q$ gains 4.25% improvement over KGHT because the designed session-adaptive propagation only aggregates the most relevant neighbor information to the session context and avoids introducing unrelated neighbors.

5.4.3 Influence of Different Types of Edges

We further analyze the influence of co-view, co-ATC and co-view-ATC edges in the constructed knowledge graph. Specifically, we use **KGHT**$_{qm*}$ as the baseline and respectively remove three types of edges, e.g., **KGHT**$_{qm*}$ w/o v removes the co-view edges and **KGHT**$_{qm*}$ w/o diff treat different types of edges uniformly by employing the same graph attention layer[3]. The performance of removing each specific type of edges is reported in Table 5.4 and we draw three observations:

- On Diginetica and Yoochoose 1/64, removing co-view edges decreases the performance most because of the following two reasons. First, sessions in these two datasets track user click activities rather than purchase activities and consist of more substitution items rather than complementary ones [145, 146]. Therefore, removing co-view edges that essentially capture the substitution relationship between any two items hurts the performance more than removing other types of edges. Secondly, co-view edges are constructed from both training and validation sessions, message-passing along which captures more recent transitional patterns encoded in the validation sequences. The performance decrease here indicates the distribution shift in transitional patterns from training sessions to validation sessions. One promising direction is to treat sessions at different time stamps differently, as recent transition patterns may be more indicative of the future sessions than the past ones [82].

[3]Instead of averaging aggregated embeddings across all $\mathcal{T}^2$ types of edges in Eq (5.3), we use a uniformed query, value and key matrices.

Table 5.4: Ablation study on different types of edges.

Backbone	**Diginetica**		**Yoochoose 1/64**		**THD**		**Decrease** ($\downarrow$)
	N@10	M@10	N@10	M@10	N@10	M@10	
KGHT$_{qm*}$	18.94	14.22	35.80	28.80	18.83	15.67	—
– w/o diff	18.34	13.80	35.15	28.22	18.68	15.50	2.02%
– w/o v	15.21	11.46	34.36	27.68	18.66	15.53	9.78%
– w/o a	18.70	13.97	35.04	28.09	18.76	15.64	1.39%
– w/o va	/	/	/	/	18.79	15.63	0.23%

* **w/o diff**: uniformly aggregate different types of neighbors with no differentiation.
* **w/o v/a/va**: with no co-view/co-ATC/co-view-ATC edges.

- Conversely, on THD dataset, since the session is composed of the user's sequentially add-to-cart activity, removing co-view edges causes minor performance degradation compared with the one on the other two datasets. Moreover, removing co-view-ATC edges causes little-to-no performance change, which indicates that capturing the view-to-ATC transition patterns may not help predict users' next clicked items.

5.4.4 Performance Comparison with baselines

From the analysis in previous sections, we select the best model configuration: M-KGHT$_{qm*}$ and compare it with state-of-the-art baselines **NARM** [151], **SR-GNN** [137] and **M2TRec** [93]. For the implementation of M2TRec and M-KGHT$_{qm*}$ on Yoochoose 1/64 with no item meta-attributes, we directly use item ID as the input. We modify the implementation of NARM and SR-GNN to include the validation performance and hence align with our experimental setting. In Table 5.5, M2TRec performs worse than SR-GNN and NARM on Diginetica and Yoochoose 1/64 while better on THD. This is because M2TRec designs an item meta-embedding layer to integer item meta-features and the more informative meta-features on THD than the ones on Diginetica/Yoochoose maximize the benefit of using M2TRec. However, our proposed M-KGHT$_{qm*}$ achieves the best performance across all three datasets for both NDCG@10 and MRR@10. This exhibits the general ability of the proposed framework to realize superior performance over existing methods across datasets with varying real-world dynamics, i.e., having varying amounts of meta-attributes, user-item interaction types, and session lengths. Note that because THD is an industrial-scale dataset (THD has around 3 million training sessions that are 5/10 times larger than Diginetica/Yoochoose), the performance gain on THD is slightly weaker than the gains on the other two datasets.

Table 5.5: Performance comparison among different baselines.

Baseline	**Diginetica**		**Yoochoose 1/64**		**THD**	
	N@10	M@10	N@10	M@10	N@10	M@10
NARM	16.06	12.12	34.45	28.10	17.44	14.62
SR-GNN	$\underline{17.10}$	$\underline{12.82}$	$\underline{35.44}$	$\underline{28.66}$	17.08	14.45
M2TRec	15.66	11.83	33.68	27.16	$\underline{18.80}$	$\underline{15.78}$
M-KGHT$_{qm*}$	**18.95**	**14.23**	**35.80**	**28.80**	**18.98**	**15.81**

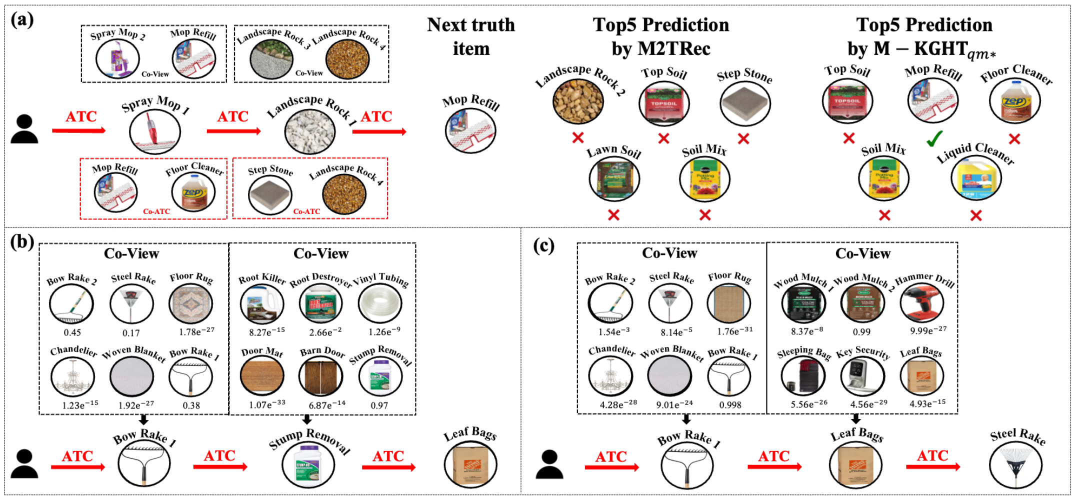


Figure 5.3: Case study of our proposed **M-KGHT**. (a) Comparing the Top5 recommendation by M2TRec and **M-KGHT**. By leveraging neighborhood information of Spray Mop1, the correct item Mop Refill appears in the recommendation list. (b)-(c) visualizes the learned attention of one attention head over co-view neighbors. Since users in both of these two sessions intend to clean the gardens, the session-adaptive graph propagation successfully learns the higher intention for garden-related items.

5.4.5 Industrial-level Case Study

We further deploy our designed system in the industrial-level setting by training it with 6 million sessions spanning the last two years and evaluating over 0.1 million sessions in the following month. We achieved 2% performance improvement over the previously deployed model at The Home Depot. Next, we conduct some case studies to visualize the effect of our proposed system.

Visualizing the recommendation of M2TRec and M-KGHT$_{qm*}$. To interpret the advantage of the proposed **M-KGHT**$_{qm*}$ over the second best model **M2TRec** on THD dataset, we select the sessions where the top-5 recommendation list given by **M-KGHT**$_{qm*}$ hit the next truth item while the one given by **M2TRec** does not. We visualize one example in Figure 5.3(a) where the customer sequentially add-to-cart the spray mop and landscape rock. Items on the top-5 recommendation list given by **M2TRec** are uniformly aligned with the landscape rock while recommended items by **M-KGHT**$_{qm*}$ align with both the spray mop and landscape. Furthermore, because **M-KGHT**$_{qm*}$ aggregates neighbor information of mop refill to spray mop, the recommendation hits the true item, which demonstrates the benefits of leveraging neighborhood information.

Visualizing the attention of session-adaptive propagation. We further visualize the graph attention learned by our session-adaptive propagation in Figure 5.3(b)-(c). Clearly, users generating these two sequences (b)-(c) intend to clean their own gardens and successfully, the model learns to aggregate less information from irrelevant neighbors, e.g., $1.78e^{-27}$ from floor rug to bow rake 1 in (b) and $9.99e^{-27}$ from hammer drill to leaf bags in (c). Interestingly, we find model sometimes pays attention to only one relevant neighbor. For example, even though both bow rake 1 and steel rake are aligned with the intention of our second customer in (c), the model pays its whole attention to bow rake 1. This aligns with diversity observation in [152] and motivates the design of multi-head attention to focus on different important neighbors. More importantly, we can find the neighborhood attention of the same item bow rake 1 varies from 0.38 in (b) to 0.998 in (c), demonstrating that even though the neighborhoods are the same, the attention assigned to them changes if the sequence content changes. This verifies the effectiveness of our proposed session-adaptive propagation.

5.5 Conclusion

Characterizing user intention by modeling global transitional patterns of user-item interactions is essential in the session-based recommendation. Traditional transformer-based models fail to capture global transitional patterns among items. More recent GNN-augmented transformers ignore the session context and only consider one type of customer-item interaction. Given these problems, we propose a knowledge graph-based session recommendation framework with session-adaptive propagation. We construct the graph by extracting three different types of user-item interactions and design a session-adaptive propagation for aggregating neighbors' information based on their consistency with the session intention. A comprehensive ablation analysis shows the proposed strategies provide a 10%-20% improvement. Moreover, our case study on recommendation interpretation demonstrates that learned neighborhood attention is highly determined by the consistency of the neighbor with the session intention.

CHAPTER 6

Imbalance Issue: Overcoming Imbalance Issue in Node Classification

Recent years have witnessed the significant success of applying graph neural networks (GNNs) in learning effective node representations for classification. However, current GNNs are mostly built under the balanced data-splitting, which is inconsistent with many real-world networks where the number of training nodes can be extremely imbalanced among the classes. Thus, directly utilizing current GNNs on imbalanced data would generate coarse representations of nodes in minority classes and ultimately compromise the classification performance. This therefore portends the importance of developing effective GNNs for handling imbalanced graph data. In this work, we propose a novel Distance-wise Prototypical Graph Neural Network (DPGNN), which proposes a class prototype-driven training to balance the training loss between majority and minority classes and then leverages distance metric learning to differentiate the contributions of different dimensions of representations and fully encode the relative position of each node to each class prototype. Moreover, we design a new imbalanced label propagation mechanism to derive extra supervision from unlabeled nodes and employ self-supervised learning to smooth representations of adjacent nodes while separating inter-class prototypes. Comprehensive node classification experiments and parameter analysis on multiple networks are conducted and the proposed DPGNN almost always significantly outperforms all other baselines, which demonstrates its effectiveness in imbalanced node classification[1].

6.1 Introduction

Graph neural networks (GNNs) have become one of the most promising paradigms in graph representation learning for node classification [15, 153], where a classifier is trained by labeled nodes and then used for categorizing labels of all remaining nodes. Although many GNN variants have been proposed to complete this task [14, 16, 26, 112], prevailing prior works follow a (semi-)supervised setting where labeled nodes are assumed to be balanced among different classes [154]. However, this setting requires sufficient balanced labeled nodes for each class, which is over-idealized and inconsistent with reality.

In many real-world networks, the class distribution of labeled nodes is inherently skewed [155, 63, 156] where a large portion of classes (minority classes) only contain a limited number of labeled nodes (minority nodes) while few classes (majority classes) contain enough labeled nodes (majority nodes). Since most GNNs are designed without considering the potential of class imbalance, directly using them on an imbalanced dataset would undermine the learned representations of minority nodes. The imbalanced node classification

[1] https://arxiv.org/abs/2110.12035

with GNNs naturally inherits existing challenges of deep learning in imbalanced classification: the inclination to learning towards majority classes [157] and the catastrophic forgetting of previously learned instances in minority classes [158]. First, deep learning models improve representations for completing target tasks via backpropagation from the training loss. However, in the class imbalanced scenario, the main component of the training loss comes from the majority of classes. Thus, the gradient is dominated by majority classes such that the model is updated towards behaving significantly better on majority classes than minority ones. Second, deep learning models are notorious for demanding big data for updating parameters [158], which limits their ability to learn from minority classes that may only have a few training instances in the imbalanced setting. In addition, over-smoothing [159, 160] that occurs as a general issue in GNNs would become even worse in the imbalanced setting: representations of minority nodes would become similar to the majority ones and deviate from their spectrum due to the imbalance bias introduced in the message passing.

Traditional methods for handling imbalance are either augmenting data via under(over)-sampling [161], or assigning weights to adjust the portion of training loss from different classes [162]. Most of these methods are proposed for non-graph structured data. DR-GCN [156] is the pioneer in exploring node imbalance classification by adversarial training each class distribution and enforcing the consistency between the labeled and unlabeled data distributions. However, the adversarial training may yield unrepresentative minority class distribution given few minority nodes [163]. Further, RECT [164] is proposed by leveraging topological regularization to derive extra supervision for minority nodes. At the same time, it is designed for entirely imbalanced data, and its performance on partially imbalanced data is unclear. GraphSMOTE [63] generalizes SMOTE [165] to the graph domain by pre-training an edge generator and hence adding relational information for the new synthetic nodes from SMOTE. However, the computation of calculating the similarity between all pairs of nodes and pre-training the edge generator is extremely heavy.

To tackle the aforementioned challenges of imbalanced node classification, we present a Distance-wise Prototypical Graph Neural Network (DPGNN), which first applies class prototype-driven training to balance the training loss of different classes and then leverages distance metric learning to differentiate the contribution of each dimension of distance from each query node to all class prototypes. Ultimately, the classification is performed by comparing the similarity of the learned distance metric representations of the nodes with the ones of class prototypes. Additionally, DPGNN introduces a novel imbalance label propagation scheme to augment training data and employs self-supervised learning to smooth representations of adjacent nodes while separating inter-class prototypes. The main contributions are summarized as follows:

- We construct a balanced training scheme inspired by episodic training and further introduce distance metric learning to capture node distances to class prototypes better.

- We design a new imbalanced label propagation scheme to augment the training data and employ self-supervised learning to further improve the distance metric representations.
- We perform extensive imbalanced node classification experiments on real-world datasets across various levels of class imbalance with detailed parameter analysis to corroborate the effectiveness of our model.

6.2 Related Work

6.2.1 Class Imbalance Problem

Class imbalance exists in many real-world applications [166], where classes with more (or less) training instances are termed as majority (or minority) classes. The imbalance in the number of training instances among different classes significantly affects the performance of supervised learning and hence has become a classical research direction [157]. Generally, the approaches against this problem are summarized into three levels, i.e., data-level, algorithm-level, and hybird [167]. Data-level methods aim to improve the data by balancing the training instances (e.g., up(down)-sampling [161]), whereas algorithm-level methods [162] attempt to improve the training process by modifying the learning algorithm such as re-weighting. In this work, we research the class imbalance in the graph domain. At the algorithm-level, our work balances the training loss by comparing labeled nodes with each class of prototypes. At the data-level, we propose an imbalanced label propagation to obtain additional minority samples. Thus, our DPGNN is a hybrid approach.

6.2.2 Graph Neural Networks

GNNs have achieved unprecedented success on graph-structured data due to the combination of feature propagation and prediction. However, most prior work on GNNs fails to consider the class imbalance problem, which unfortunately widely exists in real-world applications [63, 164]. RECT [164] was developed by merging a GNN and proximity-based embedding component for the completely imbalanced setting (i.e., where some classes can even have no labeled nodes). DR-GCN [156] explored node imbalance classification by adversarial training each class distribution and enforcing the consistency between the labeled and unlabeled data distributions. Unfortunately, the adversarial trained generator and discriminator may be overfitted to a few minority nodes and hence lose generalizability [163]. More recently, GraphSMOTE [63] was designed using a GNN encoder to learn node embeddings and an extra edge generator to generate edges connecting synthetic minority nodes. However, the model is time-consuming and somewhat learned in an ad-hoc fashion. Motivated by this and to provide a grounded approach for imbalanced node classification, we design a novel GNN-based framework that utilizes class prototypes to balance the training loss and distance metric learning to fully encode the relative position of each node to each class prototype.

6.3 Problem Statement

We denote an attributed graph by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ where $\mathcal{V} = \{v_1, ..., v_n\}$ is the set of n nodes, $\mathcal{E}$ is the set of m edges with e_{ij} being the edge between nodes v_i and v_j, and $\mathbf{X} = [\mathbf{x}_1^\top, ..., \mathbf{x}_n^\top] \in \mathbb{R}^{n \times d}$ is the node feature matrix with $\mathbf{x}_i$ indicating the features of node v_i. The network topology is described by its adjacency matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$, where $\mathbf{A}_{ij} = 1$ denotes an edge between nodes v_i and v_j, and $\mathbf{A}_{ij} = 0$ otherwise. The diagonal matrix of node degrees is denoted as $\mathbf{D} \in \mathbb{R}^{n \times n}$, where $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$ calculates the degree of the node v_i. We let $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ represent the adjacency matrix with added self-loops and similarly let $\widetilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}$. Then the normalized adjacency matrix can be defined as $\widehat{\mathbf{A}} = \widetilde{\mathbf{D}}^{-0.5}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-0.5}$. The neighborhood node set of the center node v_i is given by $\mathcal{N}_i = \{v_j | e_{ij} \in \mathcal{E}\}$. Finally, $\mathcal{C} = \{1, 2, ..., C\}$ is the set of C classes and $\mathbf{Y} \in \mathbb{R}^{n \times C}$ is the one-hot node label matrix. Furthermore, we denote $\mathcal{S}_c$, $\mathcal{Q}_c$ as the support/query set of class c. Let $\mathbf{H}^{\mathcal{S}_c}/\mathbf{H}^{\mathcal{Q}_c}$ be the representation matrix of nodes in support/query set $\mathcal{S}_c$, $\mathbf{P}$ be the prototypical representation matrix, and $\mathbf{G}$ be the distance representation matrix. Now, given the previously defined notations, the imbalanced node classification can be formalized as follows:

Given an attributed network $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ with labels for a subset of nodes $\mathcal{V}_l \subset \mathcal{V}$ that are imbalanced among the C classes, we aim to learn a node classifier $f : f(\mathcal{V}, \mathcal{E}, \mathbf{X}) \to \mathbf{Y}$ that can work well for both majority and minority classes.

6.4 The proposed framework

In this section, we present our proposed Distance-wise Prototypical Graph Neural Network (DPGNN) that can solve the challenges imposed on deep graph learning when given imbalanced training data. The framework is shown in Figure 6.1, including four main components: (a) class prototype-driven balanced training, (b) distance metric learning, (c) imbalanced label propagation, and (d) self-supervised learning. Specifically, we borrow the idea of episodic training from prototypical networks [168] to first balance the training loss across different classes via prototype-driven balanced training (Figure 6.1(a)). However, unlike traditional prototypical networks where labels of query nodes are assigned based on their nearest class prototypes, we further employ distance metric learning to construct another distance metric space that can differentiate different distance dimensions and fully characterize the position of each query node relative to all class prototypes (Figure 6.1(b)). To fully incorporate the graph topology information in alleviating the imbalance problem, we also design a novel imbalanced label propagation scheme and two self-supervised components to aid in learning high-quality distance metric representations (Figure 6.1(c) and 6.1(d), respectively). Next, we describe each component in detail.

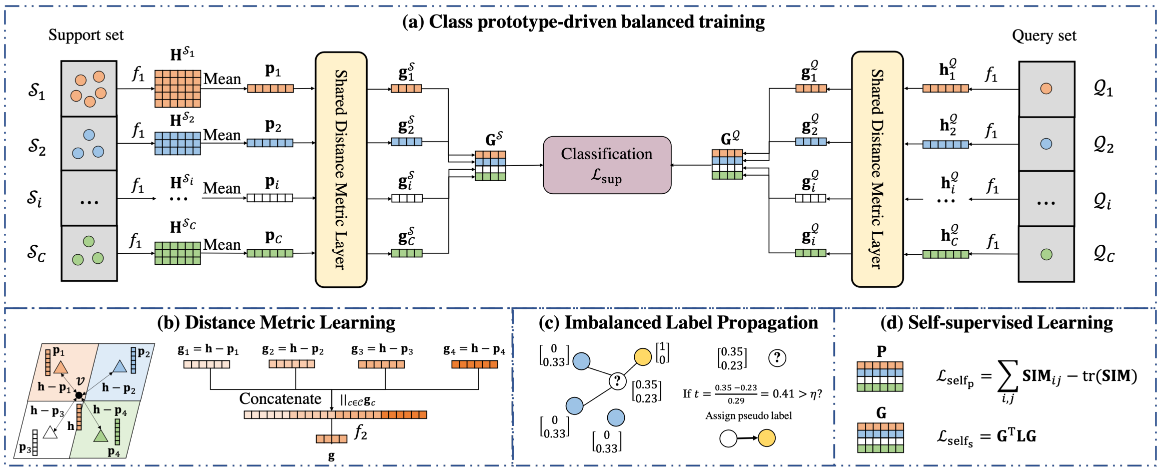

(a) Class prototype-driven balanced training
Support set
Query set
Mean
Shared Distance Metric Layer
Classification
$\mathcal{L}_{\text{sup}}$
(b) Distance Metric Learning
Concatenate
(c) Imbalanced Label Propagation
Assign pseudo label
(d) Self-supervised Learning
$\mathcal{L}_{\text{self}_p} = \sum_{i,j} \mathbf{SIM}_{ij} - \text{tr}(\mathbf{SIM})$
$\mathcal{L}_{\text{self}_s} = \mathbf{G}^T\mathbf{L}\mathbf{G}$

Figure 6.1: Overview of the Distance-wise Prototypical Graph Neural Network, with four main components: (a) class prototype-driven balanced training, (b) distance metric learning, (c) imbalanced label propagation, and (d) self-supervised learning.

6.4.1 Class Prototype-driven Balanced Training

To balance the training loss from majority and minority classes, we leverage the idea of episodic training by sampling support and query sets and then calculating representations of prototypes for each class. To avoid using the original sparse and high-dimensional node features and to allow learned complex feature transformation, we first apply a GNN-based encoder $f_1 : \mathbb{R}^{n\times d} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times d'}$ to obtain d'-dimensional node representations $\mathbf{H} \in \mathbb{R}^{n\times d'}$. Most GNN-based encoders f_1 can be decomposed into two components: neighborhood propagation and feature transformation, which can be generally formalized as:

$$\mathbf{h}_i^l = \text{TRAN}^l(\text{AGGR}^l(\mathbf{h}_i^{l-1}, \text{PROP}^l(\{\mathbf{h}_j^{l-1} | v_j \in \mathcal{N}_i\}))), \tag{6.1}$$

where in each layer l, the representations $\{\mathbf{h}_j^{l-1} | v_j \in \mathcal{N}_i\}$ of the neighbors are first propagated via PROP^l to node v_i and aggregated with its own representation $\mathbf{h}_i^{l-1}$ by AGGR^l, then the combined representation is further transformed by TRAN^l to output the representation $\mathbf{h}_i^l$ of node v_i after layer l of the GNN.

With the learned node embeddings $\mathbf{H}$ coming from the GNN-based encoder's last layer, we aim to compute representations of class prototypes $\mathbf{P} \in \mathbb{R}^{C\times d'}$. During each training epoch, the model is fed with an episode sampled from the labeled training nodes $\mathcal{V}_l$, which is further divided into support sets $\mathcal{S} = \{\mathcal{S}_c | c \in \mathcal{C}\}$ and query sets $\mathcal{Q} = \{\mathcal{Q}_c | c \in \mathcal{C}\}$ of each class. The query sets only contain one training sample from each class and provide a balanced way to train the encoder. The support sets are formed by grouping the remaining training samples from each class except the query samples from the query sets, which serve as anchors to characterize the class prototypes of each class. Following the idea of Prototypical Networks [168], we define class prototypes to be closely surrounded by nodes of the same class, such that they can precisely represent their class. More specifically, the prototype $\mathbf{p}_c$ of class c is computed by:

$$\mathbf{p}_c = \text{PROTO}(\mathbf{H}^{\mathcal{S}_c}) = \text{PROTO}(\{\mathbf{h}_i | v_i \in \mathcal{S}_c\}), c \in \mathcal{C}, \tag{6.2}$$

where PROTO is the prototype computation which calculates the representation of a class prototype $\mathbf{p}_c$ based on node representations $\mathbf{H}^{\mathcal{S}_c}$ that come from the support set $\mathcal{S}_c$. For instance, in the vanilla Prototypical Network [168], the mean-pooling is employed here:

$$\mathbf{p}_c = \frac{1}{|\mathcal{S}_c|} \sum_{v_i \in \mathcal{S}_c} \mathbf{h}_i, c \in \mathcal{C}. \tag{6.3}$$

Applying mean-pooling to calculate class prototypes assumes that each class can be represented using only one class prototype, which might be underrepresentative when an unimodal distribution assumption is violated. Thus, multi-prototypes to represent each class could be used, such as replacing mean-pooling with another permutation-invariant function (e.g., K-means clustering [169]), or even iteratively learn multi-prototypes according to the complexity of the class distribution [170]. We leave this as one future direction.

Prototypes serve as representatives of each class and can be used to classify query nodes by selecting their nearest prototypes. However, directly employing a softmax over distances to prototypes in the embedding space to obtain the class probability distribution [158, 168, 155] will make different dimensions of distance contribute equally to the classification and in high dimensional data pairwise Euclidean distances tend to converge [171]. Furthermore, using only the embedding distance of a node to the nearest prototype, omit its embedding distance to all other non-nearest prototypes that may encode extra information about the node's class. Thus, instead of classifying query nodes directly based on their nearest prototype [155, 168], we devise a distance metric layer to project nodes from the original embedding space to another distance metric space, where query nodes are classified by comparing their learned distance metric representations with the ones of class prototypes.

6.4.2 Distance Metric Learning

To project each node from the original embedding space to the distance metric space, we first concatenate the difference of its embedding from each class prototype. Next, we apply a linear transformation on top of that to pay different attention to each original distance dimension and adaptively extract useful distance information. Figure 6.1(b) demonstrates the detailed procedure of computing distance metric representation $\mathbf{g}$ of node v.

Given a node v with embedding $\mathbf{h} \in \mathbb{R}^{d'}$, we calculate its distance metric representation $\mathbf{g}_c \in \mathbb{R}^{d'}$ to each class prototype c as:

$$\mathbf{g}_c = \mathbf{h} - \mathbf{p}_c, c \in \mathcal{C}, \tag{6.4}$$

where $\mathbf{h} - \mathbf{p}_c$ calculates the difference of the embedding between each node and each class prototype. For each node, only considering its embedding difference to one class prototype cannot fully locate its position. Therefore, we concatenate the difference of the node's embedding to all class prototypes $||_{c\in\{1,\ldots,C\}}\mathbf{g}_c$ and further apply a linear transformation $f_2 : \mathbb{R}^{d'C} \rightarrow \mathbb{R}^{d''}$ to pay different levels of attention to different dimensions of the embedding difference and adaptively extract useful embedding difference information.

$$\mathbf{g} = f_2(||_{c\in\mathcal{C}}\mathbf{g}_c). \tag{6.5}$$

The distance metric representation $\mathbf{g}$ encodes the distance information of the node v to all class prototypes, which, as a result, precisely captures its relative position to all class prototypes. To use these distance metric representations to prototypes for reference to classify query nodes, we feed the representations of prototypes and query nodes $\mathbf{p}_c, \mathbf{h}_c^{\mathcal{Q}}$, for $c \in \mathcal{C}$ into the shared distance metric layer to learn their distance metric representations $\mathbf{g}_c^{\mathcal{S}}, \mathbf{g}_c^{\mathcal{Q}}$, for $c \in \mathcal{C}$. Then, we stack them across all classes to compute the distance metric representations of prototypes and query nodes:

$$\mathbf{G}^{\mathcal{M}} = \text{stack}(\mathbf{g}_c^{\mathcal{M}} | c \in \mathcal{C}), \mathcal{M} \in \{\mathcal{Q}, \mathcal{S}\} \tag{6.6}$$

Next, the predicted class distribution for each query node is:

$$\mathbf{F} = \text{softmax}(\mathbf{G}^{\mathcal{Q}}(\mathbf{G}^{\mathcal{S}})^\top), \tag{6.7}$$

where $\mathbf{F}_c$ gives the predicted probability distribution of class c's query node over all classes, which is then used to calculate the supervised classification loss:

$$\mathcal{L}_{\text{class}} = \frac{1}{C}\sum_{c=1}^{C}\ell(\mathbf{F}_c, c), \tag{6.8}$$

where $\ell(\cdot,\cdot)$ is a loss function to measure the difference between predictions and ground-truth labels, such as cross-entropy.

6.4.3 Imbalanced Label Propagation

Network homophily [172] assumes that connected nodes tend to share similar attributes or belong to the same class, which is commonly the case in various real-world networks [36, 32]. Thus, it can naturally be harnessed to augment the training data, especially for increasing the minority training nodes. Motivated by this, we perform imbalanced label propagation to annotate node labels based on their neighboring nodes' labels and filter out unconfident ones by computing their topology information gain [173].

Since we only have access to the class information of labeled nodes $\mathcal{V}_l$ and to balance the voting effect between the majority and the minority classes, for unlabeled nodes we mask their one-hot labels by filling zero vector $\mathbf{0} \in \mathbb{R}^C$, and for labeled nodes we multiply their one-hot labels by the weighting factor γ_i:

$$\widetilde{\mathbf{Y}}_i = \begin{cases} \gamma_i \mathbf{Y}_i, & v_i \in \mathcal{V}_l \\ \mathbf{0}, & v_i \in \mathcal{V} \backslash \mathcal{V}_l, \end{cases} \quad \text{where } \gamma_i = \frac{|\mathcal{V}_l|}{\sum_{v_j \in \mathcal{V}_l} \mathbf{Y}_{j\phi(v_i)}} \tag{6.9}$$

computes the inverse ratio of the number of labeled nodes in the node v_i's class $\phi(v_i)$, $\sum_{v_j \in \mathcal{V}_l} \mathbf{Y}_{j\phi(v_i)}$, to the total number of labeled nodes $|\mathcal{V}_l|$. Then we propagate this unbiased label distribution $\widetilde{\mathbf{Y}}$ to their neighbors that are at most$k$-hops away by considering k^{th}-order adjacency matrix as follows:

$$\widehat{\mathbf{Y}} = \widehat{\mathbf{A}}^k \widetilde{\mathbf{Y}}. \tag{6.10}$$

Note that $\widehat{\mathbf{A}}^k \widetilde{\mathbf{Y}}$ can be computed efficiently by applying power iteration sequentially from right to left [29].

However, this leads to another issue that $\widehat{\mathbf{Y}}$ cannot be used directly in class prototype-driven balance training to sample support and query sets since $\widehat{\mathbf{Y}}$ is a soft-label (instead of having hard label assignments). Moreover, nodes that fall in the topological boundary between different classes may possess noisy label distribution and mislead the training process. Therefore, we utilize the idea of topological information gain (TIG) [173] to filter out nodes with weak and obscure label distributions. TIG describes the task information effectiveness that the node obtains from the labeled source along the network topology [173]. We regard the maximum entry of the soft-label $\widehat{\mathbf{Y}}_i$ to be the possible class type that the node v_i can be and the other entries as confusing information. Then the topological information gain $\mathbf{t}_i$ for node v_i is calculated as:

$$\mathbf{t}_i = \frac{\max(\widehat{\mathbf{Y}}_i) - \frac{(\sum_{c \in \mathcal{C}} \widehat{\mathbf{Y}}_{ic} - \max(\widehat{\mathbf{Y}}_i))}{C-1}}{\frac{1}{C} \sum_{c \in \mathcal{C}} \widehat{\mathbf{Y}}_{ic}}. \tag{6.11}$$

A high $\mathbf{t}_i$ means the label distribution of node v_i is sharp/strong, and thus it lies in the cluster of nodes in class $\text{argmax}(\widehat{\mathbf{Y}}_i)$. By network homophily, its label is also conjectured to be $\text{argmax}(\widehat{\mathbf{Y}}_i)$. As such, we generate the hard pseudo label of non-training node v_i by binary thresholding its topological information gain $\mathbf{t}_i$ as:

$$\check{\mathbf{Y}}_{ic} = \begin{cases} 1 & \text{, if } \mathbf{t}_i > \eta \text{ and } c = \text{argmax}(\widehat{\mathbf{Y}}_i) \\ 0 & \text{, if } \mathbf{t}_i \leq \eta \text{ or } c \neq \text{argmax}(\widehat{\mathbf{Y}}_i) \end{cases}, v_i \in \mathcal{V} \backslash \mathcal{V}_l, \tag{6.12}$$

where η is a hyperparameter that controls the trade-off between the quality and the number of the augmented labels. Higher η leads to precise labels with sharp distribution and thus guarantees the labeling quality, but less nodes could be augmented. Lower η leads to more imprecise and noisy labels with even distribution. Note that for the initial labeled nodes $v_i \in \mathcal{V}_l$, we still utilize their original labels and thus the final labels we use for sampling support and query sets in class prototype-driven balanced training:

$$\bar{\mathbf{Y}}_i = \begin{cases} \mathbf{Y}_i & v_i \in \mathcal{V}_l \\ \check{\mathbf{Y}}_i & v_i \in \mathcal{V} \backslash \mathcal{V}_l \end{cases} \tag{6.13}$$

6.4.4 Self-Supervised Learning (SSL)

Although node embeddings computed from the GNN-based encoder f_1 embed network topology information, the prototype computation PROTO in Eq. (6.2) and the distance metric learning in Eq. (6.4) do not consider network topology. Inspired by the intuition that different class prototypes should have different representations and adjacent nodes should have similar distance metric representations, we design two GNN self-supervised learning (SSL) [174, 175] pretext tasks to emphasize the topological information learned by DPGNN. To ensure that different prototypes have different representations, we minimize the representation similarity of prototypes from different classes as follows:

$$\mathcal{L}_{\text{ssl}_\text{p}} = \sum_{i=1}^{C} \sum_{j=1}^{C} \big(\mathbf{SIM}_{ij}\big) - \text{tr}(\mathbf{SIM}), \text{ where } \mathbf{SIM}_{ij} = \frac{\mathbf{p}_i{}^\top \mathbf{p}_j}{||\mathbf{p}_i|| ||\mathbf{p}_j||} \tag{6.14}$$

To smooth the distance metric representations between adjacent nodes and reinforce the graph structure in the learned representations, we adopt the following objective function:

$$\mathcal{L}_{\text{ssl}_\text{s}} = \mathbf{G}^\top \mathbf{L} \mathbf{G}, \tag{6.15}$$

where $\mathbf{G}$ is the distance metric representations by transforming the concatenated embedding distance between each node to each class prototype (i.e., Eq. (6.4) and Eq. (6.5)), and $\mathbf{L}$ is the Laplacian matrix of the underlying network. If we adopt the normalized Laplacian matrix, i.e., $\mathbf{L} = \mathbf{I} - \widehat{\mathbf{A}}$, Eq. (6.15) can be rewritten as:

$$\mathcal{L}_{\text{ssl}_\text{s}} = \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}_i} \Big(\frac{\mathbf{g}_i}{\sqrt{d_i}} - \frac{\mathbf{g}_j}{\sqrt{d_j}}\Big)^2. \tag{6.16}$$

It is clear that $\mathcal{L}_{\text{ssl}_\text{s}}$ is small when adjacent nodes share similar distance metric representations to class prototypes. Collecting one supervised classification loss and two SSL loss, the overall objective function is:

$$\mathcal{L} = \mathcal{L}_{\text{class}} + \lambda_1 \mathcal{L}_{\text{ssl}_\text{p}} + \lambda_2 \mathcal{L}_{\text{ssl}_\text{s}}, \tag{6.17}$$

where λ_1 and λ_2 are two hyperparameters that control the contribution of the two SSL losses (i.e., $\mathcal{L}_{\text{ssl}_\text{p}}$ and $\mathcal{L}_{\text{ssl}_\text{s}}$) in addition to the supervised classification loss $\mathcal{L}_{\text{class}}$.

6.4.5 Complexity Analysis

Having introduced all components of DPGNN, next, we compare DPGNN with vanilla GNN-based encoders by analyzing the additional complexity in terms of time and model parameters.

Compared to vanilla GNN-based encoders, additional computational requirements come from three components: distance metric learning, imbalanced label propagation, and self-supervised learning. For distance metric learning, the calculation of the pairwise embedding difference between each node and each class prototype requires $O(|\mathcal{V}|C)$ time complexity if implemented naïvely. Typically the class number C is multiple orders of magnitude less than the node number $|\mathcal{V}|$ in a network and, therefore, can be treated as a constant, which leads to linear time complexity $O(|\mathcal{V}|)$. For uncommon networks that have a vast number of classes with the same magnitude as the size of the network, we could select sub-classes $\mathcal{C}_{\text{sub}}$ as anchors and approximate the distance metric representations by considering the distance to these anchors rather than all classes [176, 177, 178]. In imbalanced label propagation, the most computational part comes from propagating labels in Eq. (6.10), which can be completed efficiently by applying power iteration from the edge view in $O(k|\mathcal{E}|)$ compared to $O(|\mathcal{V}|^3)$ for matrix calculation of $\widehat{\mathbf{A}}^k$. Among the two SSL components, the heaviest computation comes from smoothing distance metric embeddings of adjacent nodes in Eq. (6.15), which can be calculated from the edge view as Eq. (6.16); thus, is linear with the number of edges $O(|\mathcal{E}|)$.

For the model complexity, apart from the parameters of the GNN-based encoder, additional parameters of DPGNN come from the linear transformation f_2, which are $\mathbf{W}^{f_2}$ and $\mathbf{b}^{f_2}$. Hence, compared with vanilla GNN-based encoders, the overall additional parameters are $O(d'C \times d'')$ where $d'C$ is the dimension of concatenated distance metric representations to all class prototypes $||_{c\in\{1,\dots,C\}}\mathbf{g}_c$ and d'' is the dimension of distance metric representation $\mathbf{g}$ (as in Eq. (6.5)). Typically, if the number of classes C is small or if we only select some sub-classes as anchors, $d'C$ will be far less than the original high-dimensional node attributes d. Therefore, the extra model complexity $O(d'C \times d'')$ could practically be ignored compared to the complexity of the GNN-based encoder, which has $O(d \times d')$.

Table 6.1: Basic dataset statistics for imbalanced node classification.

Networks	Nodes	Edges	Features	Classes	Homophily*	Type
Cora	2,708	5,429	1,433	7	0.81	Citation
Citeseer	3,327	4,732	3,703	6	0.74	Citation
Pubmed	19,717	44,338	500	3	0.80	Citation
Cora-ML	2,995	8,416	2,879	7	0.79	Citation
DBLP	17,716	105,734	1,639	4	0.83	Citation
Amazon Computers	13,381	245,778	767	10	0.78	Online Product
Amazon Photo	7,487	119,043	745	8	0.83	Online Product
Twitch PT	1,912	64,510	128	2	0.58	Social

6.5 Experiment

In this section, we conduct extensive experiments on imbalanced node classification to evaluate the effectiveness of DPGNN. In particular, we target to answer the following three questions:

- **Q1:** How effective is DPGNN compared to other baselines on Imbalanced node classification under different imbalance ratios?
- **Q2:** How do different components of DPGNN contribute to performance improvement in imbalanced node classification?
- **Q3:** How does the threshold η in imbalanced label propagation affect the performance of DPGNN?

6.5.1 Experiment Settings

Datasets. We experiment on widely-adopted citation networks [45], Amazon product networks [179], and an online social network [180]. Table 6.1 presents the basic network statistics for these datasets. In the five citation networks and the online social network, class distributions are relatively balanced, so we use an imitative imbalanced setting: we choose the first 5, 4, 2, 5, 3, 1 class(es) as a minority and down-sample their training nodes to 2 compared to 20 for other majority class(es), which creates an imbalanced class distribution with imbalance ratio 10. For the Amazon product networks whose class distributions are genuinely imbalanced, we use their original class ratios and set the total training nodes as 50 and 30, respectively. For validation and testing sets, 500 and 1000 nodes are selected respectively for all eight datasets, which is commonly employed in the literature [15, 16]. This setting is used throughout the paper unless otherwise stated.

Baselines. To evaluate the effectiveness of the proposed DPGNN framework, we select six representative approaches for handling imbalance classification, including the current state-of-the-art methods, where the first three target point-based imbalance classification. At the same time, the last three are explicitly designed for imbalance node classification in networks. **Up-sampling:** A classical approach that repeats samples from minority classes. Following [63], we implement this in the embedding space by duplicating minority nodes' representations; **Re-weight [181]:** A cost-sensitive approach that assigns class-specific loss weights, we set the weights of each class as the inverse ratio of the total training nodes to the number in that class; **SMOTE [165]:** Synthetic minority samples are created by interpolating minority samples with their nearest neighbors within the same class based on the output of the last GNN layer; **GraphSMOTE [63]:** Advancing SMOTE [165], GraphSMOTE has two types of edge generators that can be pre-trained to connect synthetic nodes to the original graph. We report the result of the best generator variant for each dataset; **RECT [164]:** A supervised model leveraging both a GNN and an unsupervised node proximity-based embedding, which is designed for a completely imbalanced label setting; **DR-GCN [156]:** Two types of regularization to tackle class imbalance are proposed: class-conditional adversarial training to separate labeled nodes and unlabeled nodes latent distribution constraint to maintain training equilibrium.

Equipping the first three baselines with a 2-layer GCN [15] encoder, we then have 7 baselines: GCN, GCN_{us}, GCN_{rw}, GCN_{st}, GraphSMOTE, RECT, and DR-GCN that are used for a comprehensive empirical analysis on imbalanced node classification.

Evaluation Metrics. Following existing work in imbalanced classification [182], we measure performance with F1-macro, F1-micro, and F1-weighted scores. F1-macro and F1-weighted scores evaluate model performance across different classes, with the former taking the unweighted mean over the accuracy of each class and the latter weighted mean to account for label imbalance. F1-micro is taken over all testing examples, which gives an overall evaluation of the performance while undervalues nodes in minority classes.

Parameter Settings. We implement our proposed DPGNN and some necessary baselines Pytorch Geometric. For GraphSmote[2], RECT[3], and DR-GCN[4], we use the authors' original code with any needed modifications from their GitHub repositories. Aiming to provide a rigorous/fair comparison, we tune hyperparameters for all models individually on each dataset around the default/best settings reported in their paper.

For DPGCN, we tune the following hyperparameters: dropout rate $\in \{0, 0.5\}$, the coefficient balancing the loss contribution $\lambda_1, \lambda_2 \in \{1, 10\}$, the threshold $\eta \in [0, 6]$, and the training epoch $\in \{1000, 3000, 6000\}$. We select the 2-layer GCN with 256 hidden units as the encoder f_1 due to its simplicity and efficiency, and termed our framework as DPGCN in the following. Note that any other GNN-based encoder can be used instead.

Table 6.2: Node classification performance on eight datasets with the best performance emboldened and second underlined.

	Dataset (Homophily Value)											
	Cora (0.81)			**Citeseer** (0.74)			**Pubmed** (0.80)			**Cora-ML** (0.79)		
Model	F1-macro	F1-weight	F1-micro	F1-macro	F1-weight	F1-micro	F1-macro	F1-weight	F1-micro	F1-macro	F1-weight	F1-micro
GCN	0.5205	0.5195	0.5212	0.3870	0.4169	0.4692	0.5501	0.5569	0.5928	0.5205	0.5195	0.5212
GCN_{us}	0.5631	0.5659	0.5727	0.4503	0.4822	0.5220	0.6272	0.6323	0.6451	0.5656	0.5516	0.5611
GCN_{rw}	0.5609	0.5660	0.5724	0.4457	0.4800	0.5156	0.6169	0.6178	0.6327	0.5609	0.5660	0.5724
GCN_{st}	0.5488	0.5398	0.5519	0.4462	0.4794	0.5127	0.5861	0.5964	0.6186	0.5488	0.5398	0.5519
GSMOTE[5]	0.5845	0.6026	0.5820	0.4236	0.4774	0.5020	0.6122	0.5998	0.6110	0.6233	0.6450	0.6130
RECT	0.5234	0.5025	0.5448	0.4002	0.4243	0.4549	0.5713	0.5597	0.6002	0.5530	0.5560	0.6026
DR-GCN	0.5513	0.5362	0.5520	0.3924	0.4414	0.4880	0.5628	0.5730	0.5559	0.5412	0.4716	0.5060
DPGCN	**0.7115**	**0.7029**	**0.7111**	**0.4838**	**0.5180**	**0.5397**	**0.7018**	**0.7176**	**0.7189**	**0.7273**	**0.7278**	**0.7305**

Model	**DBLP** (0.83)			**Amazon Computers** (0.78)			**Amazon Photo** (0.83)			**Twitch PT** (0.58)		
	F1-macro	F1-weight	F1-micro	F1-macro	F1-weight	F1-micro	F1-macro	F1-weight	F1-micro	F1-macro	F1-weight	F1-micro
GCN	0.3482	0.3829	0.3876	0.5343	0.6808	0.6975	0.6999	0.7617	0.7666	0.4557	0.4510	0.4656
GCN_{us}	0.4214	0.4599	0.4795	0.5757	0.6876	0.6883	0.7135	0.7645	0.7632	0.4917	0.5088	0.5131
GCN_{rw}	0.4379	0.4744	0.4892	0.5732	0.6845	0.6841	0.7204	0.7683	0.7670	0.4963	0.5168	0.5193
GCN_{st}	0.3757	0.4302	0.4522	0.5863	0.6999	0.7107	0.7302	0.7782	0.7800	0.5002	0.5267	0.5301
GSMOTE[6]	0.4844	0.4938	0.4530	0.5509	0.6213	0.6370	0.7227	0.7716	0.7750	0.3922	0.3558	0.4130
RECT	0.3438	0.3602	0.3810	0.5222	0.7002	**0.7351**	0.6858	0.7763	0.8007	0.4898	0.4843	0.4904
DR-GCN	0.3797	0.4190	0.4510	0.5357	0.6745	0.7100	0.7434	**0.7979**	**0.8130**	0.4791	0.5524	0.5750
DPGCN	**0.6167**	**0.6665**	**0.6597**	**0.6702**	**0.7280**	0.7310	**0.7600**	0.7943	0.7917	**0.5600**	**0.5944**	**0.5915**

6.5.2 Performance Comparison

In this subsection, we answer the first question by comparing the performance of DPGCN with other baselines and report the average performance per metric across 20 different data splits along with the edge homophily [32] for each dataset. We observe that DPGCN performs the best in all 8 datasets by F1-macro with a large margin and best on 7(6) of the 8 datasets according to F1-weight (F1-micro) over other baselines. The F1-micro of GCN is higher than F1-macro on all datasets except Cora, which indicates accuracy is vastly different across different classes and therefore signifies the detrimental effect imposed by class imbalance.

$GCN_{us, rw, st}$ improve the performance over GCN on all datasets, which demonstrates their generalizability and effectiveness in handling the imbalance issue. Specifically, GCN_{us} and GCN_{rw} share similar performance since essentially both of them balance the training loss while the performance of GCN_{st} is unstable since randomly selecting linear interpolation coefficient cannot guarantee the generated instances follow the ground-truth class distributions. GraphSMOTE performs better than other baselines on datasets with higher homophily, such as Cora, Cora-ML, and DBLP, while worse on datasets with lower homophily, such as Citeseer and Twitch PT. This is because edges in lower homophily networks tend to link nodes from different classes. Therefore, the pre-trained edge generator is also guided to link nodes from different classes. In this case, aggregating neighborhood features incorporates more noise information and, therefore, compromises the downstream classification. Even though two advanced models, RECT and DR-GCN, achieve performance gain over the GCN baseline, both of their improvements are surprisingly not obvious compared with other baselines. For example, RECT achieves comparable performance to GCN on Cora and DBLP. For RECT, we

[2]GraphSMOTE Code: https://github.com/TianxiangZhao/GraphSmote
[3]RECT Code: https://github.com/zhengwang100/RECT
[4]DR-GCN Code: https://github.com/codeshareabc/DRGCN

hypothesize that the embeddings obtained by GNNs already encode the structural information and, therefore, further optimizing based on node proximity derives limited useful supervision. Another potential reason is that RECT is proposed for completely imbalanced labels while here we still have some training instances for each minority class. For DR-GCN, the reason for no significant performance improvement is that the limited training nodes may cause over-fitting of their generator and discriminator such that the class distribution is ill-defined via adversarial training [163].

Moreover, we investigate the stability of the performance improvement achieved by our model by varying the imbalance ratio from 1:1 to 1:10. Specifically, we fix the number of training nodes as 2 for minority classes and gradually increase the number of training nodes from 2 to 40 for majority classes, which exhausts the imbalance scenarios from being balanced to the heavily imbalanced. In Figure **??**, we can see DPGCN always performs the best when the imbalance ratio is beyond 1:4 on the shown datasets, and the gap grows larger as the imbalance ratio further increases, which demonstrates the superiority of DPGCN over other baselines in handling heavily imbalanced data. Besides, we observe that our model also achieves higher performance when data is balanced (imbalance ratio 1:1) on most datasets. This is because the imbalanced label propagation derives extra training supervision from pseudo-labeled data and indicates the potential of our model in the balanced data setting. On Citeseer, our model achieves lower performance on the first few imbalance ratios because of the inefficiency of imbalanced label propagation and smoothing of adjacent nodes due to lower homophily (i.e., 0.74) of Citeseer compared with the other four datasets.

6.6 Conclusion

In this chapter, we focused on the imbalanced node classification problem in graphs, which widely exists in real-world settings such as malicious user detection and drug function testing. Noticing that imbalanced node classification naturally inherits issues of deep learning for imbalance classification: the inclination to learn towards majority classes and the catastrophic forgetting of previously learned instances in minority classes, we use a class prototype-driven balance training scheme to balance the training loss between different classes. To further alleviate the issue of unrepresentative class prototypes, we construct another distance metric space to fully leverage the distance information of query nodes to all class prototypes. Moreover, we derive extra supervision from network topology by an imbalanced label propagation scheme and smooth the learned distance metric representation between adjacent nodes. Experiments on 8 real-world datasets demonstrate the effectiveness of the proposed DPGNN framework in relieving the class imbalance issue. For future work, we plan to study imbalanced graph classification and expect to utilize the graph topology as extra supervision.

CHAPTER 7

Imbalance Issue: Overcoming Imbalance Issue in Graph Classification

Graph Neural Networks (GNNs) have achieved unprecedented success in identifying categorical labels of graphs. However, most existing graph classification problems with GNNs follow the protocol of balanced data splitting, which misaligns with many real-world scenarios in which some classes have fewer labels than others. Training GNNs under this imbalanced scenario may lead to uninformative representations of graphs in minority classes, and compromise the overall classification performance, which signifies the importance of developing effective GNNs for imbalanced graph classification. Existing methods are either tailored for non-graph structured data or designed specifically for imbalanced node classification while few focus on imbalanced graph classification. To this end, we introduce a novel framework, Graph-of-Graph Neural Networks (G^2GNN), which alleviates the graph imbalance issue by deriving extra supervision globally from neighboring graphs and locally from stochastic augmentations of graphs. Globally, we construct a graph of graphs (GoG) based on kernel similarity and perform GoG propagation to aggregate neighboring graph representations. Locally, we employ topological augmentation via masking node features or dropping edges with self-consistency regularization to generate stochastic augmentations of each graph that improve the model generalibility. Graph classification experiments conducted on seven benchmark datasets demonstrate our proposed G^2GNN outperforms numerous baselines by roughly 5% in both F1-macro and F1-micro scores[1].

7.1 Introduction

Employing graph representations for classification has recently attracted significant attention due to the emergence of Graph Neural Networks (GNNs) associated with its unprecedented power in expressing graph representations [18]. A typical GNN architecture for graph classification begins with an encoder that extracts node representations by propagating neighborhood information, followed by pooling operations that integrate node representations into graph representations, which are then fed into a classifier to predict graph labels [183]. Although numerous GNN variants have been proposed by configuring different propagation and pooling schemes, most works are framed under the setting of balanced data-split where an equal number of labeled graphs are provided as the training data for each class [184]. However, collecting such balanced data tends to be time-intensive and resource-expensive, and thus are often impossible in reality [185].

In many real-world graph datasets, the distribution of graphs across classes varies from a slight bias to a severe imbalance where a large portion of classes contain a limited number of labeled graphs (minority

[1]https://dl.acm.org/doi/abs/10.1145/3511808.3557356

classes) while few classes contain enough labeled graphs [155, 186] (majority classes). For example, despite the huge chemical space, few compounds are labeled active with the potential to interact with a target biomacromolecule; the remaining majority are labeled inactive [187, 51, 188]. Since most GNNs are designed and evaluated on balanced datasets, directly employing them on imbalanced datasets would compromise the overall classification performance. As one sub-branch of deep learning on graph-structured data, GNNs similarly inherit two severe problems from traditional deep learning on imbalanced datasets: the inclination to learning towards majority classes [157] and poor generalization from given scarce training data to abounding unseen testing data [184, 189]. Aiming at these two challenges, traditional solutions include augmenting data via under- or over-sampling [161, 190], assigning weights to adjust the portion of training loss of different classes [162], and constructing synthetic training data via interpolation over minority instances to balance the training data [165]. However, these methods have been primarily designed on point-based data, and their performance on graph-structured data is unclear.

An imbalance in graph-structured data could lie either in the node or graph domain, where nodes (graphs) in different classes have different amounts of training data. Nearly all related GNN works focus on imbalanced node classification by either pre-training or adversarial training to reconstruct the graph topology [156, 164, 63, 8], while to the best of our knowledge, imbalanced graph classification with GNNs remains largely unexplored. On the one hand, unlike node classification, where we can derive extra supervision for minority nodes from their neighborhoods, graphs are individual instances that are isolated from each other, and we cannot aggregate information directly from other graphs by propagation. On the other hand, compared with an imbalance on regular grid or sequence data (e.g., images or text) where the imbalance lies in the feature or semantic domain, the imbalance of graph-structured data could also be attributed to the graph topology since unrepresentative topology presented by limited training graphs may ill-define minority classes that hardly generalize to the topology of diverse unseen testing graphs. To address the aforementioned challenges, we present Graph-of-Graph Neural Networks (G^{2}GNN), which consist of two essential components that seamlessly work together to derive supervision globally and locally. In summary, the main contributions are as follows:

- **Problem:** We study imbalanced graph classification, which is heavily unexplored in the GNN literature.
- **Algorithm:** We propose a novel framework G^{2}GNN for imbalanced graph classification, which derives extra supervision by globally aggregating from neighboring graphs and locally augmenting graphs with self-consistency regularization.
- **Experiments:** We perform extensive experiments on various real-world datasets to corroborate the effectiveness of G^{2}GNN on imbalanced graph classification.

We first define the imbalanced graph classification problem and review related work in section 7.2. The proposed framework, G^{2}GNN, is given in Section 7.3, consisting of the global graph of graph construction/propagation and local graph augmentation. In Section 7.4, we conduct extensive experiments to validate the effectiveness of G^{2}GNN. Finally, we conclude and discuss future work in Section 7.5.

Let $G = (\mathcal{V}^G, \mathcal{E}^G, \mathbf{X}^G)$ denote an attributed graph with node feature $\mathbf{X}^G \in \mathbb{R}^{|\mathcal{V}^G| \times d}$ and adjacency matrix $\mathbf{A}^G \in \mathbb{R}^{|\mathcal{V}^G| \times |\mathcal{V}^G|}$ where $\mathbf{A}^G_{ij} = 1$ if there is an edge between nodes v_i, v_j and vice versa. In graph classification, given a set of N graphs $\mathcal{G} = \{G_1, G_2, ..., G_N\}$ with each graph $G_i = (\mathcal{V}^{G_i}, \mathcal{E}^{G_i}, \mathbf{X}^{G_i})$ as defined above and their labels $\mathbf{Y} \in \mathbb{R}^{N \times C}$ where C is the total number of classes, we aim to learn graph representations $\mathbf{P} \in \mathbb{R}^{N \times d'}$ with $\mathbf{P}_i$ for each $G_i \in \mathcal{G}$ that is well-predictive of its one-hot encoded label $\mathbf{Y}_i$. The problem of imbalanced graph classification can be formalized as:

Problem 1. *Given a set of attributed graphs $\mathcal{G}$ with a subset of labeled graphs $\mathcal{G}^\ell$ that are imbalanced among different classes, we aim to learn a graph encoder and classifier $\mathcal{F}$: $\mathcal{F}(\mathbf{X}^{G_i}, \mathbf{A}^{G_i}) \rightarrow \mathbf{Y}_i$ that works well for graphs in both majority and minority classes.*

7.2 Related Work

Graph Imbalance Problem. Graph imbalance exists in many real-world scenarios [63] where graph topology can be harnessed to derive extra supervision for learning graph/node representations. DR-GCN [156] handles multi-class imbalance by class-conditional adversarial training and latent distribution regularization. RECT [164] merges a GNN and proximity-based embeddings for the completely-imbalanced setting (i.e., some classes have no labeled nodes during training). GraphSMOTE [63] attempts to generate edges by pre-training an edge generator for isolated synthetic nodes generated from SMOTE [165]. Most recently, imGAGN [191] simulates both distributions of node attributes in minority classes and graph structures via generative adversarial model. However, all of these recent powerful deep learning works are proposed for node imbalance classification. Graph imbalance classification [192], remains largely unexplored, especially in GNN domain. Therefore, this work tackles this problem and we expect to leverage the graph topology via graph kernels to construct graph of graphs (GoG) and perform propagation on the constructed GoG.

Graph Augmentations. Recent years have witnessed successful applications of data augmentation in computer vision (CV) [193] and natural language processing (NLP) [194]. As its derivative in graph domain, graph augmentation enriches the training data [65, 195, 196] and therefore can be naturally leveraged to alleviate class imbalance. In this work, we augment graphs by randomly removing edges and masking node features [197, 102] to enhance the model generalizability and further employ self-consistency regularization to enforce the model to output low-entropy predictions [198].

7.3 Methodology

In this section, we introduce our proposed G^{2}GNN framework. Figure 7.1 presents an overview of G^{2}GNN, composed of two modules from a global and local perspective. Globally, a graph kernel-based GoG construction is proposed to establish a k-nearest neighbor (kNN) graph and hence enable two-level propagation, where graph representations are first obtained via a GNN encoder, and then neighboring graph representations are aggregated together through the GoG propagation on the established kNN GoG. Locally, we employ graph augmentation via masking node features or removing edges with self-consistency regularization to create novel supervision from stochastic augmentations of each individual graph. The GoG propagation serves as global governance to retain the model discriminability by smoothing intra-class graphs while separating inter-class graphs. Meanwhile, the topological augmentation behaves as a local explorer to enhance the model's generalizability in discerning unseen non-training graphs. Next, we introduce details of each module.

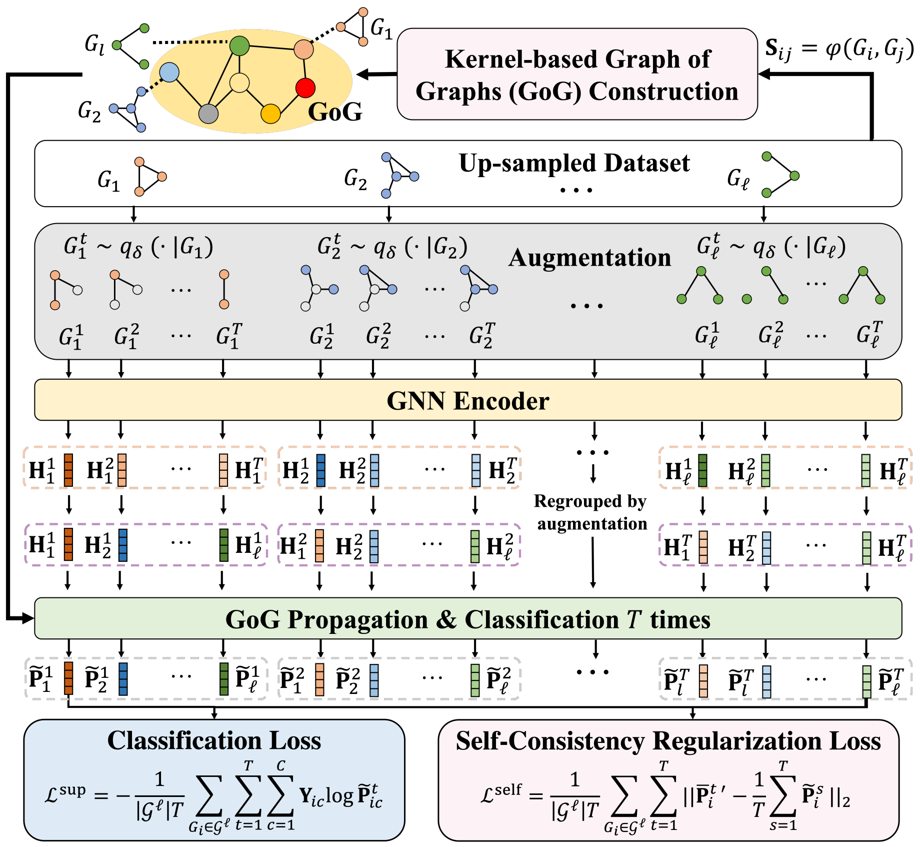
7.3.1 Global Imbalance Mitigation: Graph-of-Graph Construction/Propagation

Graph representations obtained by forwarding each graph through GNN encoders cannot be well-learned given the scarce labeled training graphs in minority classes. Therefore, we construct a GoG to connect independent graphs and perform GoG propagation to aggregate the information of neighboring graphs. The intuition is that feature propagation and aggregation would mimic the way of SMOTE [165] and mixup [199], which are two of the most fundamental approaches to handling the issue of class imbalance and poor generalizability. Aggregating representations of graphs of the same/different class/classes would simulate the interpolation of SMOTE/mixup with coefficients being determined by the specific graph convolution we use in propagation. In the following, we first introduce the basic GNN encoder to obtain graph representations, which will be later used for GoG propagation. Then, we construct GoG and empirically demonstrate its high homophily, which naturally motivates the GoG propagation.

Basic GNN Encoder. In this work, we employ a graph isomorphism network (GIN) as the encoder to learn graph representation given its distinguished discriminative power of different topologies [18]. However, our framework holds for any other GNN encoder. One GIN layer is defined as:

$$\mathbf{X}^{G_i,l+1} = \text{MLP}^l((\mathbf{A}^{G_i} + (1+\epsilon)\mathbf{I})\mathbf{X}^{G_i,l}), \forall l \in \{1, 2, ..., L\} \tag{7.1}$$

where $\mathbf{X}^{G_i,l}$ is the intermediate node representation at layer l, $\mathbf{X}^{G_i,0} = \mathbf{X}^{G_i}$ is the initial node feature in the graph G_i, and MLP is a multi-layer perceptron at layer l. After L GIN convolutions, each node aggregates information from its neighborhoods up to L hops away and a readout function integrates node representations



Kernel-based Graph of Graphs (GoG) Construction
$S_{ij} = \varphi(G_i, G_j)$
GoG
G_l
G_1
G_2
Up-sampled Dataset
G_ℓ
Augmentation
$G_1^t \sim q_\delta(\cdot|G_1)$
$G_2^t \sim q_\delta(\cdot|G_2)$
$G_\ell^t \sim q_\delta(\cdot|G_\ell)$
GNN Encoder
Regrouped by augmentation
GoG Propagation & Classification T times
Classification Loss
$\mathcal{L}^{\text{sup}} = -\frac{1}{|\mathcal{G}^\ell|T}\sum_{G_i\in\mathcal{G}^\ell}\sum_{t=1}^{T}\sum_{c=1}^{C}\mathbf{Y}_{ic}\log\widetilde{\mathbf{P}}_{ic}^t$
Self-Consistency Regularization Loss
$\mathcal{L}^{\text{self}} = \frac{1}{|\mathcal{G}^\ell|T}\sum_{G_i\in\mathcal{G}^\ell}\sum_{t=1}^{T}||\widetilde{\mathbf{P}}_i^{t\prime} - \frac{1}{T}\sum_{s=1}^{T}\widetilde{\mathbf{P}}_i^s||_2$

Figure 7.1: An overview of the Graph-of-Graph Neural Network (G^{2}GNN). To reduce imbalance effect on graph classification, we up-sample minority graphs, augment each graph T times followed by a GNN encoder to get their representations and regroup them according to their augmentation order, perform GoG propagation on constructed GoG T times with each time using all graph representations from that specific augmentation t, and finally forward the propagated representations through a classifier to compute classification loss and self-consistency regularization loss.

into the graph representation $\mathbf{H}_i$ for each graph G_i as:

$$\mathbf{H}_i = \text{READOUT}(\{\mathbf{X}_j^{G_i,L}|v_j \in \mathcal{V}^{G_i}\}) \tag{7.2}$$

Then, we construct a kNN graph on top of each graph and perform GoG propagation to borrow neighboring graphs' information. Here, we employ global-sum pooling as our READOUT function, which adds all nodes' representations to obtain the graph representation.

Graph of Graphs Construction. Given a set of graphs $\mathcal{G}$, we expect to construct a high-level graph where every graph $G_i \in \mathcal{G}$ is represented by a node, and an edge links two graphs if they are similar. In this work, we determine the graph similarity based on their topological similarity since graphs with similar topology typically possess similar functions or belong to the same class, such as scaffold hopping [200] and enzyme identification [201]. Here, we leverage the graph kernel to quantify topological similarity between pairs of graphs [202] and further use it to construct GoG. Denote the similarity matrix as $\mathbf{S} \in \mathbb{R}^{N\times N}$ where each entry $\mathbf{S}_{ij}$ measures the topological similarity between each pair of graphs (G_i, G_j) and is computed by the kernel function ϕ as:

$$\mathbf{S}_{ij} = \phi(G_i, G_j), \tag{7.3}$$

where multiple choices of the kernel function ϕ could be adopted here depending on specific types of topological similarity required by downstream tasks, and in this work, we choose the Shortest Path Kernel due to its simplicity and effectiveness as demonstrated in Section 7.4. Then we construct a kNN graph $\mathcal{G}^{\text{kNN}}$ by connecting each graph G_i with its top-k similar graphs based on the similarity matrix $\mathbf{S}$ and then measure its edge homophily as:

$$\chi^{\mathcal{G}^{\text{kNN}}} = \frac{|\{(G_i, G_j) \in \mathcal{E}^{\mathcal{G}^{\text{kNN}}} : \mathbf{Y}_i = \mathbf{Y}_j\}|}{|\mathcal{E}^{\mathcal{G}^{\text{kNN}}}|}, \tag{7.4}$$

where high $\chi^{\mathcal{G}^{\text{kNN}}}$ means most edges connect graphs of the same class and by varying k, we end up with multiple $\mathcal{G}^{\text{kNN}}$ with different homophily level. Figure 7.2 visualizes the homophily of $\mathcal{G}^{\text{kNN}}$ constructed using Shortest-Path and Weisfeiler-Lehman kernels on three graph datasets populating in the literature [197, 203]. We can clearly see that edge homophily decreases as k increases because graphs with lower topological similarity have higher chance to be selected as neighborhoods while they likely belong to different classes from corresponding center graphs. However, edge homophily even when k is up to 5 is still in $[0.7, 0.8]$ and comparable to Citeseer dataset (Citeseer is a well-known GNN node classification benchmark dataset [184].), which indicates that most edges in the constructed $\mathcal{G}^{\text{kNN}}$ connects graphs of the same class. Motivated by this observation, we perform GoG propagation on the generated kNN graph $\mathcal{G}^{\text{kNN}}$ to aggregate neighboring graph information.

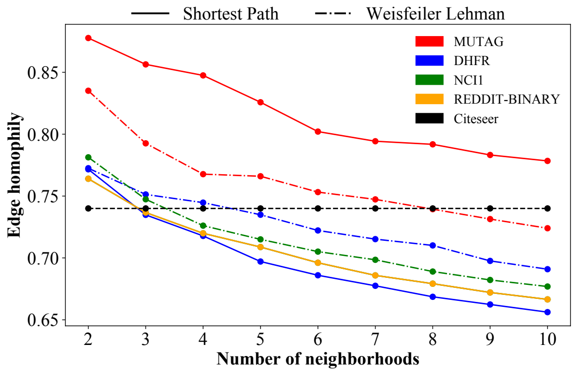


Figure 7.2: Edge homophily of constructed kNN GoGs.

Graph of Graphs Propagation. Denoting the adjacency matrix with added self-loops of the constructed graph $\mathcal{G}^{\text{kNN}}$ as $\hat{\mathbf{A}}^{\text{kNN}} = \mathbf{A}^{\text{kNN}} + \mathbf{I}$ and the corresponding degree matrix as $\hat{\mathbf{D}}^{\text{kNN}}$, the l^{th}-layer GoG propagation is formulated as:

$$\mathbf{P}^{l+1} = (\hat{\mathbf{D}}^{\text{kNN}})^{-1}\hat{\mathbf{A}}^{\text{kNN}}\mathbf{P}^{l}, l \in \{1, 2, ..., L\} \tag{7.5}$$

where $\mathbf{P}^0 = \mathbf{H}$ includes representations of all individual graphs $\mathbf{H}_i$ that are previously obtained from GIN followed by the graph pooling, as Eqs. (7.1)-(7.2). Note that here we do not differentiate between layers l, L used in GoG propagation here and layers used in GIN convolution since their difference is straightforward based on the context. After L layers propagation, the representation of a specific graph $\mathbf{P}_i^L$ aggregates information from neighboring graphs up to L hops away, which naturally smooths neighboring graphs and their labels by the following theorem [204]:

Theorem 4. *Suppose that the latent ground-truth mapping $\mathcal{M} : \mathbf{P}_i^l \to \mathbf{Y}_i$ from graph representations to graph labels is differentiable and satisfies $\mu-$Lipschitz constraints, i.e., $|\mathcal{M}(\mathbf{P}_i^l) - \mathcal{M}(\mathbf{P}_j^l)| \leq \mu ||\mathbf{P}_i^l - \mathbf{P}_j^l||_2$ for any pair of graphs G_i, G_j (μ is a constant), then the label smoothing is upper bounded by the feature smoothing among graph G_i and its neighboring graphs $\hat{\mathcal{N}}_i$ through* (7.6) *with an error $\boldsymbol{\epsilon}_i^l = \mathbf{P}_i^{l+1} - \mathbf{P}_i^l$:*

$$\underbrace{(\hat{\mathbf{d}}_i^{-1} \sum_{G_j \in \hat{\mathcal{N}}_i} \mathbf{Y}_j - \mathbf{Y}_i)}_{\text{Label smoothing}} - \underbrace{(\hat{\mathbf{d}}_i^{-1} \sum_{G_j \in \hat{\mathcal{N}}_i} o(||\mathbf{P}_j^l - \mathbf{P}_i^l||_2))}_{\text{Feature smoothing}} \leq \mu \boldsymbol{\epsilon}_i^l. \tag{7.6}$$

Proof of Theorem 4 is provided in [204]. Specifically, $\boldsymbol{\epsilon}_i^l$ quantifies the difference of the graph G_i's representation between l^{th} and $(l+1)^{\text{th}}$ propagation, which decreases as propagation proceeds [205] and eventually converges after infinite propagation $\lim_{l \to \infty} \boldsymbol{\epsilon}_i^l = 0$ [26]. Treating each graph G_i as a node in $\mathcal{G}^{\text{kNN}}$ and its representation $\mathbf{P}_i^l$ gradually converges since $\lim_{l \to \infty} \epsilon_i^l = 0$. Such feature smoothing further leads to the label smoothing based on Theorem 4. Therefore, propagating features according to Eq. (7.5) is equivalent to propagating labels among neighboring graphs, which derives extra information for imbalance classification. Given the high homophily of the $\mathcal{G}^{\text{kNN}}$ in Figure 7.2, i.e., neighboring graphs tend to share the same class, the extra information derived from feature propagation (label propagation) is very likely beneficial to the performance of downstream classification.

7.3.2 Local Imbalance Mitigation: Self-consistency Regularization

Even though feature propagation globally derives extra label information for graphs in minority classes from their neighboring graphs, training with limited graph instances still restricts the model's power in recognizing numerous unseen non-training graphs. To retain the model generalizability, we further leverage two augmenting schemes, removing edges and masking node features [197] in the next.

Removing Edges. For each graph $G_i \in \mathcal{G}$, we randomly remove a subset of edges $\widehat{\mathcal{E}}^{G_i}$ from the original edge set $\mathcal{E}^{G_i}$ with probability: $P(e_{uv} \in \widehat{\mathcal{E}}^{G_i}) = 1 - \delta_{uv}^{G_i}$, where $\delta_{uv}^{G_i}$ could be uniform or adaptive for different edges. Since uniformly removing edges (i.e., $\delta_{uv}^{G_i} = \delta$) already enjoys a boost over baselines as shown in Section 7.4.2, we leave the adaptive one as future work.

Masking Node Features. Instead of directly removing nodes that may disconnect the original graph into several components, we retain the graph structure by simply zeroing the features of some nodes following [25, 197]. Randomly masking entire features of some nodes enables each node to only aggregate information from a random subset of its neighborhoods multi-hops away, which reduces its dependency on particular neighborhoods. Compared with partially zeroing some feature channels, we empirically find that zeroing entire features performs better. Formally, we randomly sample a binary mask $\eta_j^{G_i} \sim \text{Bernoulli}(1 - \delta_j^{G_i})$ for each node v_j in graph G_i and multiply it with the node feature, i.e., $\widehat{\mathbf{X}}_j^{G_i} = \eta_j^{G_i}\mathbf{X}_j^{G_i}$ [189].

For model simplicity, we unify the probability of removing edges and masking node features as a single augmentation ratio δ. Note that by using these augmentations after feature propagation, features of each node are stochastically mixed with its neighborhoods and create multiple augmented representations, which significantly increases the model generalizability if these augmented representations overlap with unseen non-training data. However, arbitrary modification of graph topology without any regularization could unintentionally introduce invalid or even abnormal topology. Therefore, we leverage self-consistency regularization to enforce the model to output low-entropy predictions [25].

Self-Consistency Regularization. Formally, given a set of T augmented variants of a graph G_i, $\widehat{\mathcal{G}}_i = \{G_i^1, G_i^2, ..., G_i^T | G_i^t \sim q_\delta(\cdot|G_i)\}$ where $q_\delta(\cdot|G_i)$ is the augmentation distribution conditioned on the original graph G_i parameterized by the augmentation ratio δ, we feed them through a graph encoder by Eq. (7.1)-(7.2) and the GoG propagation by Eq. (7.5) to obtain their representations $\{\mathbf{P}_i^1, \mathbf{P}_i^2, ..., \mathbf{P}_i^T\}$. More specifically, we forward the set of representations of all t^{th}-augmented graphs $\{\mathbf{H}_i^t | i \in \{1, 2, ..., |\mathcal{G}|\}\}$ through GoG propagation parallelly T times to obtain their representations $\{\mathbf{P}_i^t | i \in \{1, 2, ..., |\mathcal{G}|\}, t \in \{1, 2, ..., T\}\}$. Then we further apply the classifier to obtain their predicted label distributions $\{\widetilde{\mathbf{P}}_i^t = \sigma(g_{\boldsymbol{\theta}_g}(\mathbf{P}_i^t)) | i \in \{1, 2, ..., |\mathcal{G}|\}, t \in \{1, 2, ..., T\}\}$ where σ is the softmax normalization and $g_{\boldsymbol{\theta}_g}$ is a trainable classifier parametrized by $\boldsymbol{\theta}_g$. After that, we propose to optimize the consistency of predictions among T augmentations for each graph. We first calculate the center of label distribution by taking the average of predicted distribution of all augmented variants for each specific graph G_i, i.e., $\hat{\mathbf{P}}_i = \frac{1}{T}\sum_{t=1}^{T}\widetilde{\mathbf{P}}_i^t$. Then we sharpen [198] this label distribution center:

$$\bar{\mathbf{P}}_{ij} = (\hat{\mathbf{P}}_{ij})^\tau / \sum_{c=1}^{C}(\hat{\mathbf{P}}_{ic})^\tau, \forall j \in \{1, 2, ..., C\}, i \in \{1, 2, ..., |\mathcal{G}|\} \tag{7.7}$$

where $\tau \in [0, 1]$ acts as the temperature to control the sharpness of the predicted label distribution and as $\tau \to \infty$, the sharpened label distribution of each graph approaches a one-hot distribution and hence becomes more informative. Then the self-consistency regularization loss for the graph G_i is formulated as the average L_2 distance between the predicted distribution of each augmented graph $\widetilde{\mathbf{P}}_i^t$ and their sharpened average predicted distribution:

$$\mathcal{L}_i^{\text{self}} = \frac{1}{T}\sum_{t=1}^{T}||\bar{\mathbf{P}}_i - \widetilde{\mathbf{P}}_i^t||_2. \tag{7.8}$$

Optimizing (7.8) requires the encoder and classifier to output similar predicted class distribution of different augmentations of each graph to the center one; this prevents the decision boundary of the whole model from passing through high-density regions of the marginal data distribution [206]. Also, as we increase τ, we can enforce the model to output low-entropy (high-confidence) predictions.

7.3.3 Objective Function and Prediction

The overall objective function of G^{2}GNN is formally defined as:

$$\mathcal{L} = \underbrace{-\frac{1}{|\mathcal{G}|T}\sum_{G_i\in\mathcal{G}}\sum_{t=1}^{T}\sum_{c=1}^{C}\mathbf{Y}_{ic}\log\widetilde{\mathbf{P}}_{ic}^t}_{\mathcal{L}^{\text{sup}}} + \underbrace{\frac{1}{|\mathcal{G}|T}\sum_{G_i\in\mathcal{G}}\sum_{t=1}^{T}||\bar{\mathbf{P}}_i - \widetilde{\mathbf{P}}_i^t||_2}_{\mathcal{L}^{\text{self}}}, \tag{7.9}$$

where $\mathcal{L}^{\text{sup}}$ is the cross entropy loss over all training graphs in $\mathcal{G}$ with known label information as previously defined with C graph classes to be predicted, and $\mathcal{L}^{\text{self}}$ is the self-consistency regularization loss defined by Eq. (7.8) over all training graphs.

To predict classes of graphs in validation/testing set, instead of forwarding each individual unlabeled graph through the already-trained encoder $f_{\boldsymbol{\theta}_f}$ and the classifier $g_{\boldsymbol{\theta}_g}$ to predict its label, we first generate T augmented variants of each unlabeled graph $\widehat{\mathcal{G}}_i = \{G_i^1, G_i^2, ..., G_i^T | G_i^t \sim q_\delta(\cdot|G_i)\}, \forall G_i \in \mathcal{G}/\mathcal{G}$ following Section 7.3.2 and then collectively forward the group of augmented graphs through $f_{\boldsymbol{\theta}_f}$, GoG propagation and the classifier $g_{\boldsymbol{\theta}_g}$ to obtain their predicted label distribution $\{\widetilde{\mathbf{P}}_i^1, \widetilde{\mathbf{P}}_i^2, ..., \widetilde{\mathbf{P}}_i^T\}$, then the final predicted distribution of graph $\mathcal{G}_i$ is averaged over all augmented variants as $\frac{1}{T}\sum_{t=1}^{T}\widetilde{\mathbf{P}_i}^t, \forall\mathcal{G}_i \in \mathcal{G}/\mathcal{G}$ and the final predicted class is the one that owns the highest class probability, i.e., $y_i = \underset{j\in\{1,2,...,C\}}{\arg\max}\ \frac{1}{T}\sum_{t=1}^{T}\widetilde{\mathbf{P}}_{ij}^t$.

7.3.4 Algorithm

In Algorithm 1, we present a holistic overview of the key stages in the proposed G^{2}GNN framework. Note that the GoG propagation and the graph augmentation with self-consistency regularization are both proposed to create more supervision from scarce minority training graphs, which can only handle the poor generalization problem. To avoid the problem of inclination to learning towards majority classes as mentioned in Section 7.1, we up-sample minority labeled graphs till the graphs in training and validation set are both balanced among different classes before starting the whole training processes as step 3 shows here. Balancing the labeled graphs in training set cannot only balance the training loss computed by Eq. (7.9) but also provide sufficient graphs from minority class to construct GoG. Otherwise given only few graphs in the minority class, the top-k similar graphs to one graph in minority class would be more likely come from majority class, which would further cause inter-class feature smoothing when performing GoG propagation and hence compromise

Algorithm 1: The algorithm of G^{2}GNN

```
Input: The imbalanced set of labeled graphs $\mathcal{G}$, the kernel function $\phi$, the augmentation distribution $q_\delta$, the encoder $f_{\boldsymbol{\theta}_f}$ and
       the classifier $g_{\boldsymbol{\theta}_g}$ with their learning rate $\alpha_f, \alpha_g$.
1  Compute pairwise similarity matrix $\mathbf{S}$ by Eq. (7.3) and construct $\mathcal{G}^{\text{kNN}}$ following Section 7.3.1
2  Up-sample minority graphs in $\mathcal{G}$ for both training and validation sets
3  while not converged do
4      for mini-batch of graphs $\mathcal{G}^B = \{G_i | G_i \sim \mathcal{G}, i = \{1, 2, ..., |\mathcal{G}^B|\}\}$ do
5          Find top-$k$ similar graphs for each $G_i \in \mathcal{G}^B$ based on $\mathbf{S}$ and incorporate them into $\mathcal{G}^B$      // Section 7.3.1
6          Obtain the subgraph $\mathcal{G}^{\text{kNN},B}$ from $\mathcal{G}^{\text{kNN}}$ induced by graphs in $\mathcal{G}^B$
7          For each $G_i \in \mathcal{G}^B$, generate $T$ augmented graphs $\widehat{\mathcal{G}}_i = \{G_i^1, G_i^2, ..., G_i^T | G_i^t \sim q_\delta(\cdot|G_i)\}$
            // Section 7.3.2
8          Apply graph encoder $f_{\boldsymbol{\theta}_f}$ by Eqs. (7.1)-(7.2), the GoG propagation by Eq. (7.5), and the classifier $g_{\boldsymbol{\theta}_g}$ to predict
            graph class distribution $\{\widetilde{\mathbf{P}}_i^t | G_i \in \mathcal{G}, t \in T\}$
9      Compute loss by Eq. (7.9) and update parameters
10     $\boldsymbol{\theta_g} \leftarrow \boldsymbol{\theta_g} - \alpha_g * \nabla_{\boldsymbol{\theta}_g}\mathcal{L}, \boldsymbol{\theta_f} \leftarrow \boldsymbol{\theta_f} - \alpha_f * \nabla_{\boldsymbol{\theta}_f}\mathcal{L}$      // Section 7.3.3
```

the classification performance. Balancing the labeled graphs in validation set could avoid the imbalanced bias introduced in determining which model should be preserved for later evaluation. Note that in Table 7.2, we show that even equipping other baselines with up-sampling to remove the imbalanced training bias, G^{2}GNN still achieves better performance, which demonstrates that the performance improvement is not solely caused by the technique of up-sampling but also by the proposed GoG propagation and augmentation with self-consistency regularization.

7.3.5 Complexity Analysis

Next, we compare our proposed G^{2}GNN with vanilla GNN-based encoders by analyzing the time and model complexity. Since we employ the shortest path kernel for all experiments in this work, we only analyze our models with this specific graph kernel.

Compared to vanilla GNN-based encoders, additional computational requirements come from kernel-based GoG construction and topological augmentation. In kernel-based GoG construction, applying shortest path kernel to calculate the similarity between every pair of graphs requires $O(n^3)$ [207] time and thus the total time complexity of this part is $O(\binom{|\mathcal{G}|}{2}\tilde{n}^3)$ ($\tilde{n} = \max_{G_i \in \mathcal{G}}(|\mathcal{V}^{G_i}|)$) due to the total $|\mathcal{G}|$ graphs. After computing the pairwise similarity, we can construct the GoG by naively thresholding out the top$-k$ similar graphs for each graph, and the time complexity here is $O(|\mathcal{G}|k)$. By default $k \leq |\mathcal{G}|$, we directly have $O(|\mathcal{G}|k) < O(|\mathcal{G}|^2) = O(\binom{|\mathcal{G}|}{2}) < O(\binom{|\mathcal{G}|}{2}\tilde{n}^3)$ and hence the time complexity of the first module is $O(\binom{|\mathcal{G}|}{2}\tilde{n}^3)$. Despite the prohibitively heavy computation of $O(\binom{|\mathcal{G}|}{2}\tilde{n}^3)$, the whole module is a pre-procession computation once and for all and we can further save the already computed similarity matrix $\mathbf{S}$ for future use, which therefore imposes no computational challenge. We augment graphs T times during each training epoch in topological augmentation. Each time we either go over all its edges or nodes, therefore the total time complexity of this module during each training epoch is $O(T\sum_{G_i \in \mathcal{G}^B}(|\mathcal{V}^{G_i}| + |\mathcal{E}^{G_i}|))$. Since augmenting graphs multiple times gains no further improvement than 2 [197], we fix T to be the constant

Table 7.1: Statistics of datasets for imbalanced graph classification

Networks	**# Graphs**	**# Avg-Node**	**# Avg-Edge**	**# Attr**	**Time**(s)*
PTC-MR [209]	344	14.29	14.69	18	0.257
NCI1 [210]	4110	29.87	32.30	37	11.21
MUTAG [211]	188	17.93	19.79	7	0.212
PROTEINS [202]	1113	39.06	72.82	3	11.36
D&D [212]	1178	284.32	715.66	89	574.71
DHFR [212]	756	42.43	44.54	3	3.70
REDDITB [202]	2000	429.63	497.75	$\backslash$	3376

* The column 'time' represents the actual time used for applying Shortest Path kernel to compute $\mathbf{S}$ for each dataset.

2, and therefore, the total complexity of this part is linearly proportional to the size of each graph, which imposes no additional time compared with GNN encoders. Among the GoG propagation component, the most computational part comes from propagation in Eq. (7.5), which can be efficiently computed by applying power iteration from the edge view in $O(K|\mathcal{E}^{\mathcal{G}^{\text{kNN},B}}|)$ for each subgraph induced by graphs in batch $\mathcal{G}^B$. Based on experimental results in Figure 7.5(a)-(b), we usually choose k to be small to ensure the sparsity and the high homophily of GoG, then $O(K|\mathcal{E}^{\mathcal{G}^{\text{kNN},B}}|)$ can be neglected compared with applying GNN encoders to get representations of each graph, $O(K\sum_{G_i\in\mathcal{G}^B}|\mathcal{E}^{G_i}|)$. For the model complexity, besides the parameters of GNN encoders, G^{2}GNN adds no additional parameters and therefore its model complexity is exactly the same as traditional GNN encoders.

In summary, our model introduces no extra model complexity but $O(\binom{|\mathcal{G}|}{2}\tilde{n}^3)$ extra time complexity in the pre-procession stage. We further presents the actual time used for applying Shortest Path kernel to compute $\mathbf{S}$ in Table 7.1. It can be clearly see that similarity matrix $\mathbf{S}$ is calculated in a short time for each dataset other than D&D and REDDIT-B since graphs in these two dataset are on average denser than other datasets as shown in Table 7.1. However, we can simply pre-compute this $\mathbf{S}$ once for all and reuse it for G^{2}GNN. Moreover, we can make this computation feasible by either employing the fast shortest-path kernel computations by sampling-based approximation where we sample pairs of nodes and compute shortest paths between them [208].

7.4 Experiment

Here, we evaluate the effectiveness of G^{2}GNN by conducting extensive imbalanced graph classification on multiple graph datasets with different imbalance level. We introduce the experimental setup in the following.

7.4.1 Experimental Setup

Datasets. We conduct experiments on seven widely-adopted real-world datasets [202, 212], which include Chemical compounds (PTC-MR, NCI1, MUTAG), Protein compounds (PROTEINS, D&D, DHFR), and Social Network (REDDIT-B). Details of these datasets can be found in Table 7.1.

Baselines. To evaluate the effectiveness of the proposed G^{2}GNN, we select three models designed for graph classification, which include: **GIN** [18]: A basic supervised GNN model for graph classification due to its distinguished expressiveness of graph topology; **InfoGraph** [203]: An unsupervised GNN model for learning graph representations via maximizing mutual information between the original graph and its substructures of different scales; **GraphCL** [197]: Stepping further from InfoGraph, GraphCL proposes four strategies to augment graphs and learns graph representations by maximizing the mutual information between the original graph and its augmented variants.

Strategies. Since imbalanced datasets naturally provide weak supervision on minority classes, unsupervised GNNs outweigh supervised counterparts, and selecting them as baselines could more confidently justify the superiority of our model. All the above three baselines are proposed without consideration of imbalanced setting therefore we further equip these three backbones with strategies designed specifically for handling imbalance issue, which includes: **Upsampling** (*us*): A classical approach that repeats samples from minority classes [213]. We implement this directly in the input space by duplicating minority graphs; **Reweighting** (*rw*): A general cost-sensitive approach introduced in [181] that assigns class-specific weights in computing the classification loss term in Eq. (7.9); we set the weights of each class as the inverse ratio of the total training graphs to the number of training graphs in that class; **SMOTE** (*st*): Based on the ideas of SMOTE [63], synthetic minority samples are created by interpolating minority samples with their nearest neighbors within the same class based on the output of the last GNN layer. Since directly interpolating in the topological space may generate invalid graph topology, we first obtain GNN-based encoders and interpolate minority graph representations in the embedding space to generate more minority training instances. Here, the nearest neighbors are computed according to Euclidean distance.

Equipping each of the above three backbones with up-sampling, re-weighting, and SMOTE strategies explicitly tailored for imbalanced classification, we end up with 10 baselines. Specifically, we equip up-sampling and re-weighting with all three backbones and name each new baseline by combining the name of its backbone and the equipped strategy, e.g., GIN$_{us}$ represents the backbone GIN equipped with the up-sampling strategy. Since applying SMOTE empirically leads to similar or even worse performance gains, we only stack it on the GIN backbone.

Evaluation Metrics. Following existing work in imbalanced classification [63], we use two criteria: F1-macro and F1-micro to measure the performance of G^{2}GNN and other baselines. F1-macro computes the accuracy independently for each class and then takes the average (i.e., treating different classes equally). F1-micro computes accuracy overall testing examples at once, which may underweight the minority classes. Following [25], The whole GoG propagation is conducted in the transductive setting where graphs in the training set could aggregate representations of graphs in the validation and testing sets while the classification loss is only evaluated on the given training labels.

Table 7.2: Graph classification performance on seven datasets. The standard deviation is relatively higher since we focus on the imbalance problem and use 50 different data splits (i.e., having different training data distributions). G^{2}GNN$_e$ and G^{2}GNN$_n$ represent our proposed model using the removing edges and masking node features augmentation strategy. **Bold** (underline) denotes the best/runner-up model.

Model	**MUTAG** (5:45)		**PROTEINS** (30:270)		**D&D** (30:270)		**NCI1** (100:900)	
	F1-macro	F1-micro	F1-macro	F1-micro	F1-macro	F1-micro	F1-macro	F1-micro
GIN	52.50 ± 18.70	56.77 ± 14.14	25.33 ± 7.53	28.50 ± 5.82	9.99 ± 7.44	11.88 ± 9.49	18.24 ± 7.58	18.94 ± 7.12
GIN$_{us}$	78.03 ± 7.62	78.77 ± 7.67	65.64 ± 2.67	71.55 ± 3.19	41.15 ± 3.74	70.56 ± 10.28	59.19 ± 4.39	71.80 ± 7.02
GIN$_{rw}$	77.00 ± 9.59	77.68 ± 9.30	54.54 ± 6.29	55.77 ± 7.11	28.49 ± 5.92	40.79 ± 11.84	36.84 ± 8.46	39.19 ± 10.05
GIN$_{st}$	74.61 ± 9.66	75.11 ± 9.87	56.07 ± 7.95	57.85 ± 8.70	27.08 ± 8.63	39.01 ± 15.87	40.40 ± 9.63	44.48 ± 12.05
InfoGraph	69.11 ± 9.03	69.68 ± 7.77	35.91 ± 7.58	36.81 ± 6.51	21.41 ± 4.51	27.68 ± 7.52	33.09 ± 3.30	34.03 ± 3.68
InfoGraph$_{us}$	78.62 ± 6.84	79.09 ± 6.86	62.68 ± 2.70	66.02 ± 3.18	41.55 ± 2.32	71.34 ± 6.76	53.38 ± 1.88	62.20 ± 2.63
InfoGraph$_{rw}$	80.85 ± 7.75	81.68 ± 7.83	65.73 ± 3.10	69.60 ± 3.68	41.92 ± 2.28	72.43 ± 6.63	53.05 ± 1.12	62.45 ± 1.89
GraphCL	66.82 ± 11.56	67.77 ± 9.78	40.86 ± 6.94	41.24 ± 6.38	21.02 ± 3.05	26.80 ± 4.95	31.02 ± 2.69	31.62 ± 3.05
GraphCL$_{us}$	80.06 ± 7.79	80.45 ± 7.86	64.21 ± 2.53	65.76 ± 2.61	38.96 ± 3.01	64.23 ± 8.10	49.92 ± 2.15	58.29 ± 3.30
GraphCL$_{rw}$	80.20 ± 7.27	80.84 ± 7.43	63.46 ± 2.42	64.97 ± 2.41	40.29 ± 3.31	67.96 ± 8.98	50.05 ± 2.09	58.18 ± 3.08
G^{2}GNN$_e$	80.37 ± 6.73	81.25 ± 6.87	$\mathbf{67.70 \pm 2.96}$	73.10 ± 4.05	43.25 ± 3.91	77.03 ± 9.98	63.60 ± 1.57	72.97 ± 1.81
G^{2}GNN$_n$	$\mathbf{83.01 \pm 7.01}$	$\mathbf{83.59 \pm 7.14}$	67.39 ± 2.99	$\mathbf{73.30 \pm 4.19}$	$\mathbf{43.93 \pm 3.46}$	$\mathbf{79.03 \pm 10.78}$	$\mathbf{64.78 \pm 2.86}$	$\mathbf{74.91 \pm 2.14}$

Model	**PTC-MR** (9:81)		**DHFR** (12:108)		**REDDIT-B** (50:450)		**Ave. Rank**	
	F1-macro	F1-micro	F1-macro	F1-micro	F1-macro	F1-micro	F1-macro	F1-micro
GIN	17.74 ± 6.49	20.30 ± 6.06	35.96 ± 8.87	49.46 ± 4.90	33.19 ± 14.26	36.02 ± 17.38	12.00	12.00
GIN$_{us}$	44.78 ± 8.01	55.43 ± 14.25	55.96 ± 10.06	59.39 ± 6.52	66.71 ± 3.92	83.00 ± 5.18	5.00	4.43
GIN$_{rw}$	36.96 ± 14.08	43.09 ± 20.01	55.16 ± 9.47	57.78 ± 6.69	45.17 ± 8.46	51.92 ± 12.29	8.86	8.86
GIN$_{st}$	36.30 ± 11.45	40.04 ± 15.32	56.06 ± 9.60	58.48 ± 6.42	60.05 ± 4.14	73.59 ± 6.05	8.29	8.43
InfoGraph	25.85 ± 6.14	26.71 ± 6.50	50.62 ± 8.33	56.28 ± 4.58	57.67 ± 3.80	67.10 ± 4.91	10.00	10.14
InfoGraph$_{us}$	44.29 ± 4.69	48.91 ± 7.49	59.49 ± 5.20	61.62 ± 4.18	67.01 ± 3.34	78.68 ± 3.71	5.00	5.00
InfoGraph$_{rw}$	44.09 ± 5.62	49.17 ± 8.78	58.67 ± 5.82	60.24 ± 4.80	65.79 ± 3.38	77.35 ± 3.96	4.43	4.29
GraphCL	24.22 ± 6.21	25.16 ± 5.25	50.55 ± 10.01	56.31 ± 6.12	53.40 ± 4.06	62.19 ± 5.68	10.71	10.57
GraphCL$_{us}$	45.12 ± 7.33	53.50 ± 13.31	60.29 ± 9.04	61.71 ± 6.75	62.01 ± 3.97	75.84 ± 3.98	5.29	5.43
GraphCL$_{rw}$	44.75 ± 7.62	52.22 ± 13.24	60.87 ± 6.33	61.93 ± 5.15	62.79 ± 6.93	76.15 ± 9.15	5.00	5.29
G^{2}GNN$_e$	46.40 ± 7.73	56.61 ± 13.72	$\mathbf{61.63 \pm 10.02}$	$\mathbf{63.61 \pm 6.05}$	$\mathbf{68.39 \pm 2.97}$	$\mathbf{86.35 \pm 2.27}$	**1.71**	1.86
G^{2}GNN$_n$	$\mathbf{46.61 \pm 8.27}$	$\mathbf{56.70 \pm 14.81}$	59.72 ± 6.83	61.27 ± 5.40	67.52 ± 2.60	85.43 ± 1.80	**1.71**	**1.71**

Parameter Settings. We implement our proposed G^{2}GNN and some necessary baselines using Pytorch Geometric [48]. For InfoGraph[2] and GraphCL[3] we use the original authors' code with any necessary modifications. Aiming to provide a rigorous and fair comparison across models on each dataset, we tune hyperparameters for all models individually as: the weight decay $\in [0, 0.1]$, the encoder hidden units $\in \{128, 256\}$, the learning rate $\in \{0.001, 0.01\}$, the inter-network level propagation $L \in \{1, 2, 3\}$, the augmentation ratio $\delta \in \{0.05, 0.1, 0.2\}$, the number of neighboring graphs in constructing GoG $k \in \{2, 3, 4\}$, the augmentation number $T = 2$ and sharpening temperature $\tau = 0.5$. We employ Shortest Path Kernel to compute similarity matrix $\mathbf{S}$ and set the trainable classifier g as a 2-layer MLP. For REDDITB dataset, we use one-hot encoding of the node degree as the feature of each node following [197, 203]. For reproducibility, the code of the model with its corresponding hyperparameter configurations are publicly available[4].

7.4.2 Performance Comparison

In this subsection, we compare the performance of G^{2}GNN$_e$ and G^{2}GNN$_n$, which represent the G^{2}GNN framework with the edge removal or node feature masking as augmentation, respectively, against the aforementioned baselines. Since class distributions of most datasets are not strictly imbalanced, we use an imitative imbalanced setting: we randomly set 25%/25% graphs as training/validation sets and among each of them, we choose one class as minority and reduce the graphs of this class in the training set (increase the other one) till the imbalance ratio reaches 1:9, which creates a highly imbalanced scenario[5]. We average the performance

[2]https://github.com/fanyun-sun/InfoGraph

[3]https://github.com/Shen-Lab/GraphCL

[4]Code for G^{2}GNN: https://github.com/submissionconff/G2GNN

[5]We select the amount of training and validation data as 25% to ensure the sufficiency of minority instances in both training and validation set given the imitative data distribution is at such a skewed level

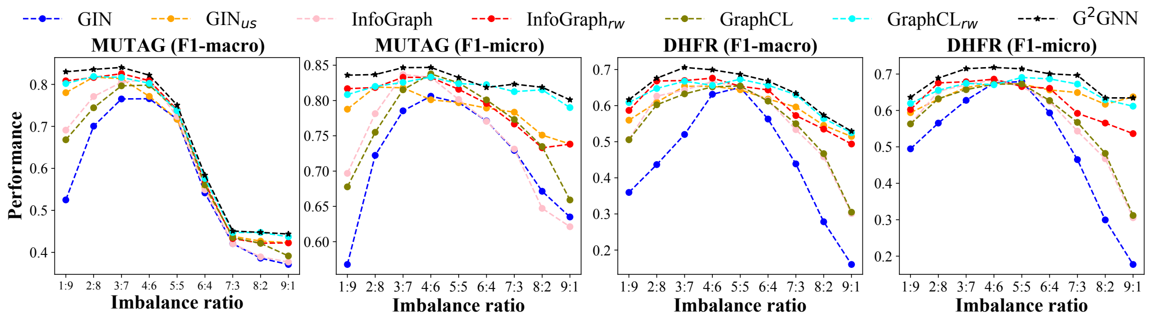


Figure 7.3: Graph classification results under different class imbalance ratios where 5:5 corresponds to a balanced scenario while 1:9 and 9:1 correspond to a highly imbalance scenario. Compared with GIN(blue), InfoGraph(pink), GraphCL(olive) designed not specifically for imbalanced scenario, our G^{2}GNN(black) model outperforms all of them in nearly all imbalance ratio settings and the margin further increases as the level of imbalance increases (i.e., deviates from the balanced scenario). We use the same training and validation graphs (25%/25%) as used in Table 7.2.

per metric across 50 different data splits to avoid bias from data splitting. Table 7.2 reports the performance's mean and standard deviation.

Table 7.2 shows that G^{2}GNN performs the best in all seven datasets under both F1-macro and F1-micro. Moreover, edge removing (i.e., G^{2}GNN$_e$) benefits more on the social network (i.e., REDDIT-B) while node feature masking (i.e., G^{2}GNN$_n$) enhances more on biochemical molecules (e.g., MUTAG, D&D, NCI1 and PTC-MR), which conforms to [197] and is partially attributed to no node attributes presented in the social network. Models specifically designed for tackling the class imbalance issue generally perform better than the corresponding bare backbones without any strategy handling imbalance. The inferior performance of GIN$_{rw(st)}$ to GIN$_{us}$ is because we either set weights for adjusting the training loss of different classes or generate synthetic samples based on training data at the current batch. Since the number of training instances in each batch may not strictly follow the prescribed imbalance ratio, the batch-dependent weight or synthetic samples hardly guarantee the global balance. InfoGraph(GraphCL)-based variants do not suffer from the issue introduced by batch training. Once we obtain graph representations from pre-trained models by mutual information maximization, we feed them through downstream classifiers all at once without involvement in the batch process. Therefore, the performance of InfoGraph(GraphCL)$_{rw(st)}$ is comparable to InfoGraph(GraphCL)$_{us}$. We emphasize that the larger standard deviation in our setting is due to the significantly different training data across different runs. We further argue that this standard deviation cannot be reduced by only increasing the number of runs due to the imbalanced nature of the problem. However, the higher average performance of our model still signifies its superiority in handling a wide range of imbalanced data splitting.

7.4.3 Influence of Imbalance Ratio

We further compare the performance of our model with other baselines under different imbalance ratios. We vary the imbalance ratio from 1:9 to 9:1 by fixing the total number of training and validation graphs as 25%/25% of the whole dataset as before and gradually varying the number of graphs in different classes. Note that for clear comparison, we only visualize the performance of the best variant among each of the three backbones in Figure 7.3. We can see that the performance of all models first increases and then decreases as the imbalance ratio increases from 0.1 to 0.9, demonstrating the detrimental effect of data imbalance on the model performance. This becomes even worse when the imbalance becomes more severe. Furthermore, the F1-macro score of our G^{2}GNN model outperforms all other baselines on both MUTAG and DHFR under each imbalance ratio, which soundly justifies the superiority and robustness of our model in alleviating the imbalance of different levels. Different from supervision presented from given labeled data, the extra supervision derived by leveraging neighboring graphs' information via propagation and topological augmentation is weakly influenced by the amount of training data. Therefore, the margin achieved by our model further grows when the imbalance ratio is either too low or too high compared with GIN, InfoGraph, and GraphCL, which are not explicitly designed for handling the imbalance scenario since the extra supervision derived in our model stays the same. Besides, our model also performs comparable or even slightly better than all other baselines under a balanced scenario, which additionally signifies the potentiality of our model in balanced data-splitting. Among other baselines, GraphCL$_{rw}$ performs the best since it applies a re-weight strategy to balance the training loss and further leverages the graph augmentation coupled with mutual information maximization to extract the most relevant information for downstream classification. An interesting observation is that the optimal performance is not always when the labeled data is strictly balanced, which reflects the uneven distribution of informatic supervision embedded across different classes.

7.4.4 Ablation Study

In this section, we conduct an ablation study to fully understand the effect of each component in G^{2}GNN on alleviating the imbalance issue. In Figure 7.4, we present performance improvement over the baseline GIN$_{us}$ achieved by our proposed framework (**G^{2}GNN$_{e(n)}$**) along with variants that remove the GoG propagation (**G^{2}GNN$_{e(n)}$ (w/o kNN)**)

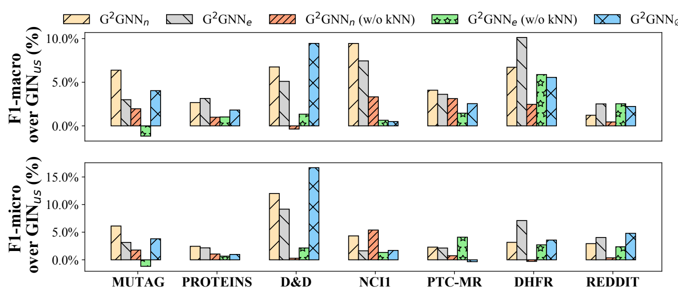


Figure 7.4: Ablation study of G^{2}GNN where we report the improvement over GIN$_{us}$ due to its simplicity and effectiveness (seen in Table 7.2) for understanding relative improvements of each G^{2}GNN component.

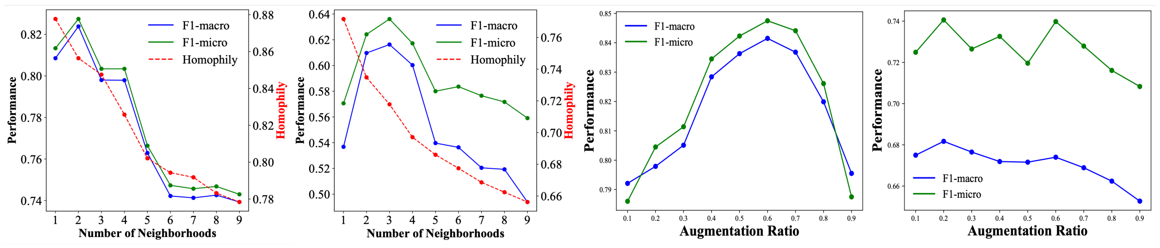


Figure 7.5: **(a)-(b)**: Relationship between neighborhood number, edge homophily, and performance on MUTAG and DHFR. The performance first increases and then decreases as the number of neighborhoods increases on $\mathcal{G}^{\text{kNN}}$. The reported result here is averaged over 20 runs. **(c)-(d)**: Relationship between augmentation ratio δ and performance on MUTAG and PROTEINS. The performance first increases and then decreases as augmentation ratio increases. The reported result here is averaged over 20 runs.

and remove the topological augmentation

(**G^{2}GNN$_\emptyset$**). **(1)** We notice that solely employing GoG propagation (**G^{2}GNN$_\emptyset$**) increases the performance on all datasets according to F1-macro, demonstrating the effectiveness of GoG propagation in alleviating the imbalance issue. **(2)** Augmenting via removing edges hurts the performance of MUTAG. This is because the size of each graph in MUTAG is relatively small and thus removing edges may undermine crucial topological information related to downstream classification. **(3)** We observe that the proposed GoG propagation and graph augmentation generally achieve more performance boost on F1-macro than F1-micro. This is because the derived supervision significantly enhances the generalizability of training data in minority classes. However, for the majority of classes where the majority of training instances already guarantee high generalizability, the enhancement would be minor. **(4)** Combining GoG propagation and graph augmentation is better than only applying one of them in most cases, which indicates that the extra supervision derived by globally borrowing neighboring information and locally augmenting graphs are both beneficial to downstream tasks and not overlapped with each other as the accumulating benefit shown here. **(5)** On NCI1, despite the minor improvement of applying only one of the proposed two modules, combining them leads to significant progress. This is because instead of propagating original graphs' representations, we leverage augmented graphs in GoG propagation, and the derived local supervision is further enhanced by the global propagation to create more supervision and enhance the model generalizability on the minority.

7.4.5 Further Probe

Effect of Neighborhood Numbers. Here, we investigate the influence of the number of neighboring graphs on the performance of G^{2}GNN$_n$ on MUTAG and DHFR. The experimental setting is the same as Section 7.4.1 except that we alter the k among $\{1, 2, ..., 9\}$. In Figure 7.5(a)-(b), we see that both the F1-macro and F1-micro increase first as k increases to 2 on MUTAG and 3 on DHFR since higher k means more number of neighboring graphs sharing the same label, as the homophily level at this stage is generally higher given the red line, therefore we derive more beneficial supervision. However, as we further increase k to 6, the performance begins to decrease since most of the added neighborhoods share different labels due to low homophily in this middle stage, providing adverse information that compromises classification. In the last stage, the performance gradually becomes stable when k increases beyond 6. This is because directly linking each graph with its 6-top similar graphs leads to a very dense GoG, and propagation on this dense GoG directly incorporates information from most of the other graphs, and therefore, the neighboring information that each graph receives is too noisy and useless.

Effect of augmentation ratio. Then we investigate the effect of augmentation ratio δ among $\{0.1, 0.2, ..., 0.9\}$ on the performance of G^{2}GNN$_n$ on MUTAG and G^{2}GNN$_e$ on PROTEINS. In Figure 7.5, we see that the F1-macro on MUTAG and PROTEINS first increase and then decrease. This is because increasing the augmentation ratio would initially generate abundant unseen graphs and enhance the model generalizability, which conforms to the advantage of harder contrastive learning concluded in [197]. However, as we increase the augmentation ratio, the performance decreases because graphs of one class may be over-augmented, which destroys the latent relationship between graphs and their class or even mismatch graphs with other classes.

Efficient Analysis. Furthermore, we compare the efficiency of each model in Table 7.3 where the running time is averaged across 10 times. Without equipping any imbalance-tailored operation, GIN achieves the shortest running time. Equipping reweighting as GIN$_{\text{rw}}$ is faster than equipping upsampling as GIN$_{\text{us}}$ since upsampling increases the size of the dataset. Our proposed G^{2}GNN and its variants generally have longer running time due to topological augmentation and graph-level propagation.

Table 7.3: Running time (in seconds) of different models.

Dataset	GIN	GIN$_{\text{us}}$	GIN$_{\text{rw}}$	G^{2}GNN$_\emptyset$	G^{2}GNN$_e$	G^{2}GNN$_n$
MUTAG	5.2	8.9	5.6	16.8	24.4	22.6
PROTEINS	24.3	40.7	25.3	111.1	155.7	153.0

7.5 Conclusion

In this chapter, we focused on imbalanced graph classification. Unlike the node imbalance problem where we can propagate neighboring nodes' information to obtain extra supervision, graphs are isolated and have no connections. Therefore, we employ a kernel-based Graph of Graph (GoG) construction to establish a kNN graph and devise a two-level propagation to derive extra supervision from neighboring graphs globally. Moreover, we employ local augmentation and upsampling of minority graphs to enhance the model generalizability in discerning unseen non-training (especially minority) graphs. Experiments on seven real-world datasets demonstrate the effectiveness of G^{2}GNN in relieving the graph imbalance issue.

CHAPTER 8

Bias Issue: Overcoming the Social Interactional Bias Issue in Node Classification

Graph Neural Networks (GNNs) have shown great power in learning node representations on graphs. However, they may inherit historical prejudices from training data, leading to discriminatory bias in predictions. Although some work has developed fair GNNs, most of them directly borrow fair representation learning techniques from non-graph domains without considering the potential problem of sensitive attribute leakage caused by feature propagation in GNNs. However, we empirically observe that feature propagation could vary the correlation of previously innocuous non-sensitive features to the sensitive ones. This can be viewed as a leakage of sensitive information which could further exacerbate discrimination in predictions. Thus, we design two feature masking strategies according to feature correlations to highlight the importance of considering feature propagation and correlation variation in alleviating discrimination. Motivated by our analysis, we propose a Fair View Graph Neural Network (FairVGNN) to generate fair views of features by automatically identifying and masking sensitive-correlated features considering correlation variation after feature propagation. Given the learned fair views, we adaptively clamp weights of the encoder to avoid using sensitive-related features. Experiments demonstrate that FairVGNN enjoys a better trade-off between model utility and fairness.[1]

8.1 Introduction

As the world becomes more connected, graph mining plays a crucial role in many domains, such as drug discovery and recommendation systems [74, 59, 51]. As one of its major branches, learning informative node representation is a fundamental solution to many real-world problems such as node classification and link prediction [112, 21]. Numerous data-driven models have been developed for learning node representations, among which Graph Neural Networks (GNNs) have achieved unprecedented success owing to the combination of neural networks and feature propagation [15, 29, 26]. Despite the significant progress of GNNs in capturing higher-order neighborhood information [24], leveraging multi-hop dependencies [112], and recognizing complex local topology contexts [214], predictions of GNNs have been demonstrated to be unfair and perpetuate undesirable discrimination [215, 216, 217, 218, 219].

Recent studies have revealed that historical data may include previous discriminatory decisions dominated by sensitive features [220, 221]. Thus, node representations learned from such data may explicitly inherit the existing societal biases and exhibit unfairness when applied in practice. Besides the sensitive

[1]https://dl.acm.org/doi/10.1145/3534678.3539404

features, network topology also serves as an implicit source of societal bias [215, 222]. By the principle of network homophily [172], nodes with similar sensitive features tend to form closer connections than dissimilar ones. Since feature propagation smooths representations of neighboring nodes while separating distant ones, representations of nodes in different sensitive groups are further segregated and their corresponding predictions are unavoidably over-associated with sensitive features.

Besides the above topology-induced bias, feature propagation could introduce another potential issue, termed as sensitive information leakage. Since feature propagation naturally allows feature interactions among neighborhoods, the correlation between two feature channels is likely to vary after feature propagation, which is termed as correlation variation. As such, some original innocuous feature channels that have a lower correlation to sensitive channels and encode less sensitive information may become highly correlated to sensitive ones after feature propagation and hence encode more sensitive information, termed *sensitive attribute leakage*. Some research efforts have been invested in alleviating discrimination made by GNNs. However, they either borrow approaches from traditional fair representation learning such as bi-level optimization-based debiasing [215, 219] and contrastive learning [223] or directly debiasing node features and graph topology [218, 222] while overlooking the sensitive attribute leakage caused by correlation variation.

In this work, we study a novel and detrimental phenomenon where feature propagation can vary feature correlations and cause the leakage of sensitive information to innocuous features. To address this issue, we propose a principled framework, Fair View Graph Neural Network (FairVGNN) to effectively learn fair node representations and avoid sensitive attribute leakage. Our major contributions are as follows:

- **Problem**: We investigate the novel phenomenon that feature propagation could vary feature correlations and cause sensitive attribute leakage to innocuous feature channels.
- **Algorithm**: To prevent sensitive attribute leakage, we propose a novel framework, FairVGNN, to automatically learn fair views by identifying and masking sensitive-correlated channels and adaptively clamping weights to avoid leveraging sensitive-related features in learning fair node representations.
- **Evaluation**: We perform experiments on real-world datasets to corroborate that FairVGNN can approximate the model utility while reducing discrimination.

8.2 Related Work

Most prior work on GNNs exclusively focus on optimizing the model utility while totally ignoring the bias encoded in the learned node representations, which would unavoidably cause social risks in high-stake discriminatory decisions [222]. FairGNN [215] leverages a sensitive feature estimator to enhance the amount of sensitive attributes, which greatly benefits their bi-level optimization-based debiasing procedure. NIFTY [218] proposes a novel triplet-based objective function and a layer-wise weight normalization using the Lipschitz constant to promote counterfactual fairness and stability of the resulting node representations. EDITS [222] systematically summarizes the biased node representation learning into attribute and structure bias and employs the Wasserstein distance approximator to alternately debias node features and network topology. More recently, REFEREE [224] was proposed to provide structural explanations of bias in GNNs. Unlike previous work, we study a novel problem that feature propagation could cause correlation variation and sensitive leakage to innocuous features, and our proposed framework, FairVGNN, expects to learn which feature channels should be masked to alleviate discrimination considering the effect of correlation variation. Others have recently explored this concept of varying correlation during feature propagation towards developing deeper GNNs [225]. Besides the fairness issue by sensitive attributes, bias can also come from the node degree [60], graph condensation[226], or even class distribution [8], which we leave for future investigations.

8.3 Preliminaries

8.3.1 Fairness in Machine Learning

Group fairness and individual fairness are two commonly encountered fairness notions in real life [221]. Group fairness emphasizes that algorithms should not yield discriminatory outcomes for any specific demographic group [222] while individual fairness requires that similar individuals be treated similarly [227]. Here we focus on group fairness with a binary sensitive feature, i.e., $\mathbf{S} \in \{0, 1\}^n$. Following [215, 218, 222], we employ the difference of statistical parity and equal opportunity between two different sensitive groups, to evaluate the model fairness:

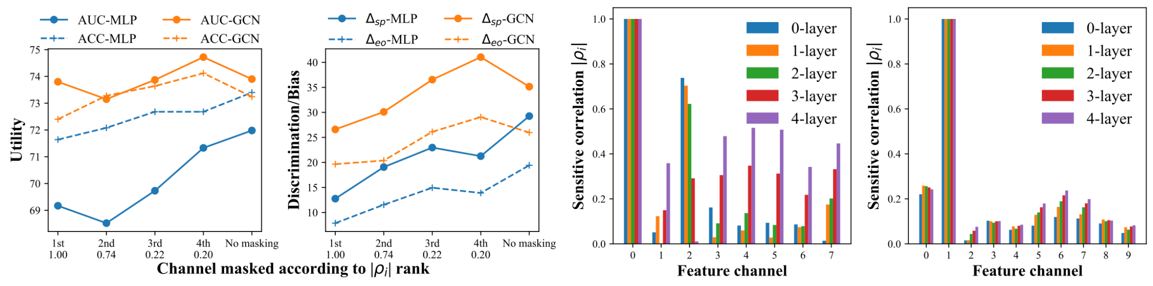
$$\Delta_{\text{sp}} = |P(\hat{y} = 1|s = 0) - P(\hat{y} = 1|s = 1)|, \tag{8.1}$$

$$\Delta_{\text{eo}} = |P(\hat{y} = 1|y = 1, s = 0) - P(\hat{y} = 1|y = 1, s = 1)|, \tag{8.2}$$

where $\Delta_{\text{sp}}(\Delta_{\text{eo}})$ measures the difference of the independence level of the prediction $\hat{y}$ (true positive rate) on the sensitive feature s between two groups. Since group fairness expects algorithms to yield similar outcomes for different demographic groups, fairer machine learning models seek lower Δ_{sp} and Δ_{eo}.

8.4 Sensitive Attribute Leakage and Correlation Variation

In this section, we study the phenomenon where sensitive information leaks to innocuous feature channels after their correlations to the sensitive feature increase during feature propagation in GNNs, which we define as *sensitive attribute leakage*. We first empirically verify feature channels with higher correlation to the



(a) Model utility **(b) Model bias** **Sensitive Correlation on (c) German and (d) Credit**

Figure 8.1: Initial empirical investigation on sensitive leakage and correlation variation on German dataset. (a)-(b) visualize the relationships between model utility/fairness and the sensitive correlation $\boldsymbol{\rho}_i$ of each masked feature channel[2]. Masking channel with less sensitive correlation leads to more biased predictions and sometimes higher model utility. (c)-(d) shows the correlation variation caused by feature propagation on German and Credit datasets. In (c), we can see sensitive correlations of the 2nd and 7th feature channel significantly change after propagation while in (d), the correlations do not change so much.

sensitive channel would cause more discrimination in predictions [228]. We denote the Pearson correlation coefficient of the i^{th}-feature channel to the sensitive channel as sensitive correlation and compute it as:

$$\boldsymbol{\rho}_i = \frac{\mathbb{E}_{v_j \sim \mathcal{V}}\big((\mathbf{X}_{ji} - \mu_i)(\mathbf{S}_j - \mu_s)\big)}{\sigma_i \sigma_s}, \forall i \in \{1, 2, ..., d\}, \tag{8.3}$$

where μ_i, σ_i denote the mean and standard deviation of the channel $\mathbf{X}_{:i}$. Intuitively, higher $\boldsymbol{\rho}_i$ indicates that the i^{th}-feature channel encodes more sensitive-related information, which would impose more discrimination in the prediction. To further verify this assumption, we mask each channel and train a 1-layer MLP/GCN followed by a linear layer to make predictions. As suggested by [218], we do not add any activation function in the MLP/GCN to avoid capturing any nonlinearity.

Figure 8.1(a)-(b) visualize the relationships between the model utility/bias and the sensitive correlation of each masked feature channel. We see that the discrimination still exists even though we mask the sensitive channel (1st). Compared with no masking situation, Δ_{sp} and Δ_{eo} almost always become lower when we mask other non-sensitive feature channels (2nd-4th), which indicates the leakage of sensitive information to other non-sensitive feature channels. Moreover, we observe the decreasing trend of Δ_{sp} and Δ_{eo} when masking channels with higher sensitive correlation since these channels encode more sensitive information and masking them would alleviate more discrimination.

Following the above observation, one natural way to prevent sensitive attribute leakage and alleviate discrimination is to mask the sensitive features as well as their highly-correlated non-sensitive features. However, feature propagation in GNNs could change feature distributions of different channels and consequentially vary feature correlations as shown by Figure 8.1(c) where we visualize the sensitive correlations of the first 8 feature channels on German after a certain number of propagations. Correlations between the sensitive

Table 8.1: Evaluating model utility/fairness when using strategies of feature masking (or no masking).

Encoder	**Strategy**	**German**				**Credit**			
		AUC	F1	Δ_{sp}	Δ_{eo}	AUC	F1	Δ_{sp}	Δ_{eo}
GCN	S_0	74.11	82.46	35.17	25.17	73.86	81.92	12.86	10.63
	S_1	73.78	81.65	11.39	9.60	72.92	81.84	12.00	9.70
	S_2	72.75	81.70	8.29	6.91	72.92	81.84	12.00	9.70
GIN	S_0	72.71	82.78	13.56	9.47	74.36	82.28	14.48	12.35
	S_1	71.66	82.50	3.01	1.72	73.44	83.23	14.29	11.79
	S_2	70.77	83.53	1.46	2.67	73.28	83.27	13.96	11.34

features and other channels change after propagation. For example, some feature channels that are originally irrelevant to the sensitive one, such as the 7th feature channel, become highly-correlated and hence encode more sensitive information.

After observing that feature propagation could vary feature correlation and cause sensitive attribute leakage, we devise two simple but effective masking strategies to highlight the importance of considering correlation variation and sensitive attribute leakage in alleviating discrimination. Specifically, we first compute sensitive correlations of each feature channel according to 1) the original features $\boldsymbol{\rho}^{\text{origin}}$ and 2) the propagated features $\boldsymbol{\rho}^{\text{prop}}$. Then, we manually mask top-k feature channels according to the absolute values of correlation given by $\boldsymbol{\rho}^{\text{origin}}$ and $\boldsymbol{\rho}^{\text{prop}}$, respectively, and train MLP/GCN/GIN on German/Credit dataset shown in Table 8.1. Detailed experimental settings are presented in Section 8.3. From Table 8.1, we have the following insightful observations: (1) Within the same encoder, masking sensitive and its related feature channels (S_1, S_2) would alleviate the discrimination while downgrading the model utility compared with no-masking (S_0). (2) GCN achieves better model utility but causes more bias than MLP on German and Credit. This implies graph structures also encode bias, and leveraging them could aggravate prediction discrimination, which is consistent with recent work [215, 222]. (3) Most importantly, S_2 achieves lower $\Delta_{\text{sp}}, \Delta_{\text{eo}}$ than S_1 for both MLP and GCN on German because the rank of sensitive correlation changes after feature propagation and masking according to S_2 leads to better fairness, which highlights the importance of considering feature propagation in determining which feature channels are more sensitive-correlated and required to be masked.

To this end, we argue that it is necessary to consider feature propagation in masking feature channels to alleviate discrimination. However, the correlation variation heavily depends on the propagation mechanism of GNNs. To tackle this challenge, we formulate our problem as follows:

Given an attributed network $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{A})$ with labels $\mathbf{Y}$ for a subset of nodes $\mathcal{V}_l \subset \mathcal{V}$, we aim to learn a fair view generator $g_{\boldsymbol{\Theta}_g} : g_{\boldsymbol{\Theta}_g}(\mathbf{X}) \to \widetilde{\mathbf{X}}$ with the expectation of simultaneously preserving task-related information and discarding sensitive information such that the downstream node classifier $f_{\boldsymbol{\Theta}_f} : f_{\boldsymbol{\Theta}_f}(\mathbf{A}, \widetilde{\mathbf{X}}) \to \mathbf{Y}$ trained on $\widetilde{\mathbf{X}}$ could achieve better trade-off between model utility and fairness.

[2]We respectively mask each feature channel and train a 1-layer MLP/GCN followed by a linear prediction layer. Dataset and experimental details are given in Section 8.6.1.

8.5 Framework

In this section, we give a detailed description of FairVGNN (shown in Figure 8.2), which includes the bi-level optimization-based debising module and the adaptive weight clamping module. In the first module, we learn a generator that generates different fair views of features to obfuscate the sensitive discriminator such that the encoder could obtain fair node representations for downstream tasks. In the second module, we propose to clamp weights of the encoder based on learned fair feature views and provide a theoretical justification on its equivalence to minimizing the upper bound of the difference of representations between two different sensitive groups. Next, we introduce the details of each component.

8.5.1 Bi-level optimization-based debising

This module includes a fair view generator $g_{\mathbf{\Theta}_g}$, a GNN-based encoder $f_{\mathbf{\Theta}_f}$, a sensitive discriminator $d_{\mathbf{\Theta}_d}$, and a classifier $c_{\mathbf{\Theta}_c}$ parametrized by $\mathbf{\Theta}_g, \mathbf{\Theta}_f, \mathbf{\Theta}_d, \mathbf{\Theta}_c$, respectively. We assume the view generator $g_{\mathbf{\Theta}_g}$ to be a learnable latent distribution from which we sample K-different masks and generate K-corresponding views $\widetilde{\mathbf{X}}^k, k \in \{1, 2, ..., K\}$. The latent distribution would be updated towards generating less-biased views $\widetilde{\mathbf{X}}$ and the stochasticity of each view would enhance the model generalizability. Then each of these K-different views $\widetilde{\mathbf{X}}^k$ are fed to the encoder $f_{\mathbf{\Theta}_f}$ together with the network topology $\mathbf{A}$ to learn node representations $\widetilde{\mathbf{H}}^k$ for downstream classifier $c_{\mathbf{\Theta}_c}$. Meanwhile, the learned node representations $\widetilde{\mathbf{H}}^k$ are used by the sensitive discriminator $d_{\mathbf{\Theta}_d}$ to predict nodes' sensitive features. This paves us a way to adopt bi-level optimization to obtain the optimal fair view generator $g_{\mathbf{\Theta}_g^*}$ where the generated views encode as much task-relevant information while discarding as much bias-relevant information as possible. We introduce the fairness-aware view generator $g_{\mathbf{\Theta}_g}$.

Fairness-aware View Generator As observed in Table 8.1, discrimination could be traced back to the sensitive features and their highly correlated non-sensitive features. Therefore, we propose to learn a view generator that automatically identifies and masks these features. More specifically, assuming the view generator as a conditional distribution $\mathbb{P}_{\widetilde{G}}$ parametrized by $\mathbf{\Theta}_g$, since bias originates from the node features $\mathbf{X}$ and is further varied by the graph topology $\mathbf{A}$, the conditional distribution of the view generator can be further expressed as a joint distribution of the attribute generator and the topological generator as $\mathbb{P}_{\widetilde{G}} = \mathbb{P}_{\widetilde{\mathbf{X}},\widetilde{\mathbf{A}}}$. Since our sensitive discriminator $d_{\mathbf{\Theta}_d}$ is directly trained on the learned node representations from GNN-based encoder $f_{\mathbf{\Theta}_f}$ as described in Section 8.5.1, we already consider the proximity-induced bias in alleviating discrimination and hence the network topology is assumed to be fixed here, i.e., $\mathbb{P}^{\mathbf{\Theta}_g}_{\widetilde{\mathbf{X}},\widetilde{\mathbf{A}}} = \mathbb{P}^{\mathbf{\Theta}_g}_{\widetilde{\mathbf{X}}}$. We will leave the joint generation of fair features and topological views as one future work.

Instead of generating $\widetilde{\mathbf{X}}$ from scratch that completely loses critical information for GNN predictions, we generate $\widetilde{\mathbf{X}}$ conditioned on the original node features $\mathbf{X}$, i.e., $\mathbb{P}^{\mathbf{\Theta}_g}_{\widetilde{\mathbf{X}}} = \mathbb{P}^{\mathbf{\Theta}_g}_{\widetilde{\mathbf{X}}}(\cdot|\mathbf{X})$. Following the preliminary

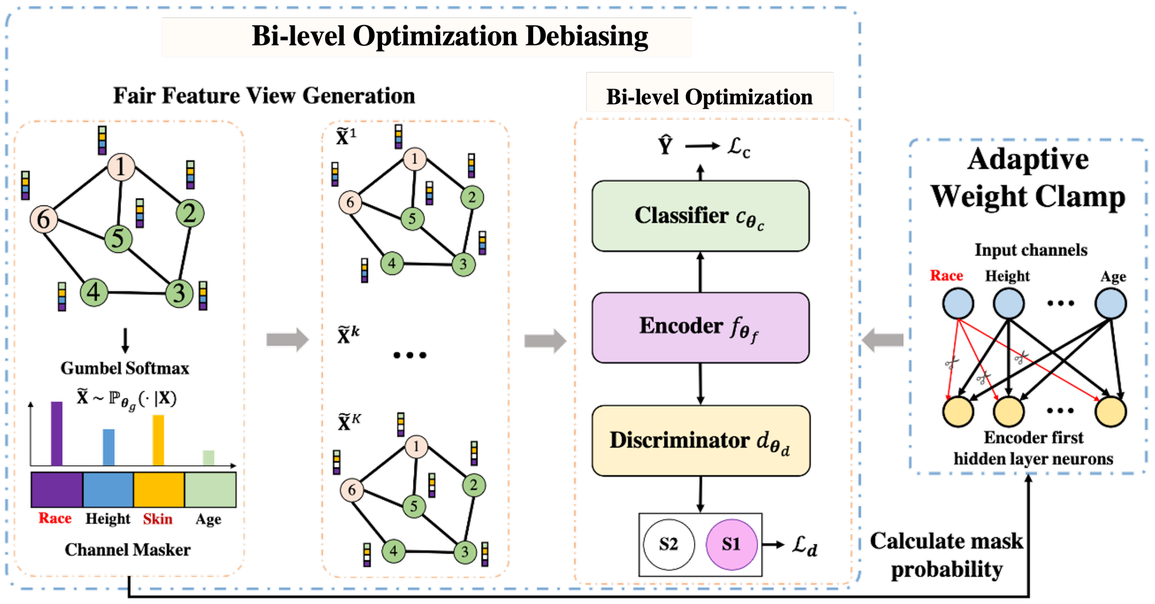

Bi-level Optimization Debiasing
Fair Feature View Generation
Bi-level Optimization
Adaptive Weight Clamp
Gumbel Softmax
$\widetilde{X} \sim \mathbb{P}_{\theta_g}(\cdot|X)$
Race Height Skin Age
Channel Masker
$\widetilde{X}^1$
$\widetilde{X}^k$
$\widetilde{X}^K$
$\hat{Y} \rightarrow \mathcal{L}_c$
Classifier c_{θ_c}
Encoder f_{θ_f}
Discriminator d_{θ_d}
S2 S1 $\rightarrow \mathcal{L}_d$
Input channels
Race Height Age
Encoder first hidden layer neurons
Calculate mask probability

Figure 8.2: An overview of the Fair View Graph Neural Network (FairVGNN), with two main modules: (a) bi-level optimization-based debising to learn the fair view of features and (b) adaptive weight clamping to clamp weights of sensitive-related channels of the encoder.

experiments, we model the generation process of $\widetilde{\mathbf{X}}$ as identifying and masking sensitive features and their highly correlated features in $\mathbf{X}$. One natural way is to select features according to their correlations $\boldsymbol{\rho}_i$ to the sensitive features $\mathbf{S}$ as defined in Eq. (8.3). However, as Figure 8.1(c) shows, feature propagation in GNNs triggers the correlation variation. Thus, instead of masking according to initial correlations that might change after feature propagation, we train a learnable mask for feature selections in a data-driven fashion. Denote our mask as $\mathbf{m} = [m_1, m_2, ..., m_d] \in \{0, 1\}^d$ so that:

$$\widetilde{\mathbf{X}} = \mathbf{X} \odot \mathbf{m} = [\mathbf{X}_1^\top \odot \mathbf{m}, \mathbf{X}_2^\top \odot \mathbf{m}, ..., \mathbf{X}_n^\top \odot \mathbf{m}], \tag{8.4}$$

then learning the conditional distribution of the feature generator $\mathbb{P}_{\widetilde{\mathbf{X}}}^{\boldsymbol{\Theta}_g}(\cdot|\mathbf{X})$ is transformed to learning a sampling distribution of the masker $\mathbb{P}_{\mathbf{m}}^{\boldsymbol{\Theta}_g}$. We assume the probability of masking each feature channel independently follows a Bernoulli distribution, i.e., $m_i \sim \text{Bernoulli}(1 - p_i), \forall i \in \{1, 2, ..., d\}$ with each feature channel i being masked with the learnable probability $p_i \in \mathbb{R}$. In this way, we can learn which feature channels should be masked to achieve less discrimination through gradient-based techniques. Since the generator $g_{\boldsymbol{\Theta}_g}$ aims to obfuscate the discriminator $d_{\boldsymbol{\Theta}_d}$ that predicts the sensitive features based on the already-propagated node representations $\widetilde{\mathbf{H}}$ from the encoder $f_{\boldsymbol{\Theta}_f}$, the generated fair feature view $\widetilde{\mathbf{X}}$ would consider the effect of correlation variation by feature propagation rather than blindly follow the order of the sensitive correlations computed by the original features $\mathbf{X}$. Generating fair feature view $\widetilde{\mathbf{X}}$ and forwarding it through the encoder $f_{\boldsymbol{\Theta}_f}$ and the classifier $c_{\boldsymbol{\Theta}_c}$ to make predictions involve sampling masks $\mathbf{m}$ from the categorical Bernoulli distribution, the whole process of which is non-differentiable due to the discreteness of masks. Therefore, we apply Gumbel-Softmax trick [229] to approximate the categorical Bernoulli distribution. Assuming for each channel i, we have a learnable sampling score $\boldsymbol{\pi}_i = [\pi_{i1}, \pi_{i2}]$ with π_{i1} score keeping while π_{i2} score masking the channel i. Then the categorical distribution Bernoulli$(1 - p_i)$ is softened by[3]:

$$p_{ij} = \frac{\exp(\frac{\log(\pi_{ij}) + g_{ij}}{\tau})}{\sum_{k=1}^{2}\exp(\frac{\log(\pi_{ik}) + g_{ik}}{\tau})}, \forall j = 1, 2, i \in \{1, 2, ..., d\}, \tag{8.5}$$

where $g_{ij} \sim$ Gumbel$(0, 1)$ and τ is the temperature factor controlling the sharpness of the Gumbel-Softmax distribution. Then, to generate $\widetilde{\mathbf{X}}$ after we sample masks $\mathbf{m}$ based on probability p_{i1}, we could either directly multiply feature channel $\mathbf{X}_{:i}$ by the probability p_{i1} or solely append the gradient of p_{i1} to the sampled hard mask[4], both of which are differentiable and can be trained end to end. After we approximate the generator $g_{\mathbf{\Theta}_g}$ via Gumbel-Softmax, we next model the GNN-based encoder $f_{\mathbf{\Theta}_f}$ to capture the information of both node features $\mathbf{X}$ and network topology $\mathbf{A}$.

GNN-based Encoder To learn from both the graph topology and node features, we employ $L-$layer GNNs as our encoder-backbone to obtain node representations $\mathbf{H}^L$. Different graph convolutions adopt different propagation mechanisms, resulting in different variations in feature correlations. Here we select GCN [15], GraphSAGE [43], and GIN [18] as our encoder-backbones. In order to consider the variation induced by the propagation of GNN-based encoders, we apply the discriminator $d_{\mathbf{\Theta}_d}$ and classifier $c_{\mathbf{\Theta}_c}$ on top of the obtained node representations $\mathbf{H}^L$ from the GNN-based encoders. Since both the classifier and the discriminator are to make predictions, one towards sensitive groups and the other towards class labels, their model architectures are similar. Hence, we introduce them together next.

Classifier and Discriminator Given node representations $\mathbf{H}^L$ obtained from any $L-$layer GNN-based encoder $f_{\mathbf{\Theta}_f}$, the classifier $c_{\mathbf{\Theta}_c}$ and the discriminator $d_{\mathbf{\Theta}_d}$ predict node labels $\hat{\mathbf{Y}}$ and sensitive attributes $\hat{\mathbf{S}}$ as:

$$\hat{\mathbf{Y}} = c_{\mathbf{\Theta}_c}(\mathbf{H}^L) = \sigma\big(\text{MLP}_c(\mathbf{H}^L)\big), \quad \hat{\mathbf{S}} = d_{\mathbf{\Theta}_d}(\mathbf{H}^L) = \sigma\big(\text{MLP}_d(\mathbf{H}^L)\big), \tag{8.6}$$

where we use two different multilayer perceptrons (MLPs): $\mathbb{R}^{d^L} \to \mathbb{R}$ for the classifier and the discriminator, and σ is the sigmoid operation. After introducing the fairness-aware view generator, the GNN-based encoder, the MLP-based classifier and the discriminator, we collect them together and perform bi-level optimization to them with the following objective function.

Bi-level Optimization We aim to learn fair views from the original graph that encodes as much task-relevant information while discarding as much sensitive-relevant information as possible. Therefore, we aim to op-

[3]We use p_{i1} instead of p_i thereafter to represent the probability of keeping channel i.

[4]$\mathbf{m} = \mathbf{m} - p_{i1}$.detach() $+ p_{i1}$

timize the whole framework from both the fairness and model utility perspectives. According to statistical parity, to optimize the fairness metric, a fair feature view should guarantee equivalent predictions between sensitive groups:

$$\mathbf{\Theta}_g^* = \arg\min_{\mathbf{\Theta}_g} \Delta_{\text{sp}} = \arg\min_{\mathbf{\Theta}_g} |P(\hat{y} = 1|s = 0) - P(\hat{y} = 1|s = 1)|, \tag{8.7}$$

where $P(\hat{y}|s)$ is the predicted distribution given the sensitive feature. Assuming $\hat{y}$ and s are conditionally independent given $\widetilde{\mathbf{H}}$ [230], to solve the global minimum of Eq. (8.7), we leverage bi-level optimization and compute the loss of the discriminator and generator $\mathcal{L}_d, \mathcal{L}_g$ as:

$$\max_{\mathbf{\Theta}_d} \mathcal{L}_{\text{d}} = \mathbb{E}_{\widetilde{\mathbf{X}} \sim \mathbb{P}^{\mathbf{\Theta}_g}_{(\widetilde{\mathbf{X}}|\mathbf{X})}} \mathbb{E}_{v_i \sim \mathcal{V}} \Big(\mathbf{S}_i \log \big(d_{\mathbf{\Theta}_d}(\widetilde{\mathbf{H}}_i^L)\big) + (1 - \mathbf{S}_i) \log(1 - d_{\mathbf{\Theta}_d}(\widetilde{\mathbf{H}}_i^L)) \Big), \tag{8.8}$$

$$\min_{\mathbf{\Theta}_g} \mathcal{L}_{\text{g}} = \mathbb{E}_{\widetilde{\mathbf{X}} \sim \mathbb{P}^{\mathbf{\Theta}_g}_{(\widetilde{\mathbf{X}}|\mathbf{X})}} \mathbb{E}_{v_i \sim \mathcal{V}} \big(d_{\mathbf{\Theta}_d}(\widetilde{\mathbf{H}}_i^L) - 0.5\big)^2 + \alpha ||\mathbf{m} - \mathbf{1}_d||_2^2, \tag{8.9}$$

where $\widetilde{\mathbf{H}}_i^L = f_{\mathbf{\Theta}_f}(\widetilde{\mathbf{X}}_i, \mathbf{A})$ and $||\mathbf{m} - \mathbf{1}_d||_2^2$ regularizes the mask to be dense, which avoids masking out sensitive-uncorrelated but task-critical information. α is the hyperparameter. Intuitively, Eq. (8.8) encourages our discriminator to correctly predict the sensitive features of each node under each generated view, and Eq. (8.9) requires our generator to generate fair feature views that enforce the well-trained discriminator to guess the sensitive features randomly.

In Theorem 5, we show that the global minimum of Eq. (8.8)-(8.9) is equivalent to the global minimum of Eq. (8.7):

Theorem 5. *Given $\widetilde{\mathbf{h}}^L$ as the representation of a specific node learned by L layer GNN-based encoder $f_{\mathbf{\Theta}_g}$ and $\alpha = 0$ in Eq.* (8.9)*, the global optimum of Eq.* (8.8)-(8.9) *is equivalent to the one of Eq.* (8.7).

Proof. Based on Proposition 1. in [231] and Proposition 4.1. in [215], the optimal discriminator is $d_{\boldsymbol{\theta}_d^*}(\widetilde{\mathbf{h}}^L) = \frac{P(\widetilde{\mathbf{h}}^L|s=1)}{P(\widetilde{\mathbf{h}}^L|s=1)+P(\widetilde{\mathbf{h}}^L|s=0)}$, which is exactly the probability when discriminator randomly guesses the sensitive features. Then we further substituted it into Eq. (8.9) and the optimal generator is achieved when $d_{\boldsymbol{\theta}_d^*}(\widetilde{\mathbf{h}}^L) = 0.5$, i.e., $P(\widetilde{\mathbf{h}}^L|s = 1) = P(\widetilde{\mathbf{h}}^L|s = 0)$. Then we have:

$$\begin{aligned} P(\hat{y} = 1|s = 1) &= \int_{\widetilde{\mathbf{h}}^L} P(\hat{y} = 1|\widetilde{\mathbf{h}}^L) P(\widetilde{\mathbf{h}}^L|s = 1) d\widetilde{\mathbf{h}}^L \\ &= \int_{\widetilde{\mathbf{h}}^L} P(\hat{y} = 1|\widetilde{\mathbf{h}}^L) P(\widetilde{\mathbf{h}}^L|s = 0) d\widetilde{\mathbf{h}}^L = P(\hat{y} = 1|s = 0), \end{aligned}$$

which is the global minimum of Eq. (8.7). □

Note that node representations $\widetilde{\mathbf{H}}^L$ have already been propagated in GNN-based encoder $f_{\boldsymbol{\theta}_f}$ and therefore, the optimal discriminator $d_{\boldsymbol{\theta}_d^*}$ could identify sensitive-related features after correlation variation. Besides the bi-level optimization training loss to ensure the fairness of the generated view, the classification loss for training the classifier $c_{\boldsymbol{\theta}_c}$ is used to guarantee the model utility:

$$\min_{\boldsymbol{\theta}_c} \mathcal{L}_{\mathrm{c}} = -\mathbb{E}_{\widetilde{\mathbf{X}} \sim \mathbb{P}^{\boldsymbol{\theta}_g}_{(\widetilde{\mathbf{X}}|\mathbf{X})}} \mathbb{E}_{v_i \sim \mathcal{V}} \Big(\mathbf{Y}_i \log \big(c_{\boldsymbol{\theta}_c}(\widetilde{\mathbf{H}}_i^L) \big) + (1 - \mathbf{Y}_i) \log \big(1 - c_{\boldsymbol{\theta}_c}(\widetilde{\mathbf{H}}_i^L) \big) \Big) \tag{8.10}$$

8.5.2 Adaptive Weight Clamping

Although the generator is theoretically guaranteed to achieve its global minimum by applying bi-level optimization, in practice the generated views may still encode sensitive information, and the corresponding classifier may still make discriminatory decisions. This is because of the instability of the training process of bi-level optimization [231] and the entanglement with the training classifier.

To alleviate the above issue, we propose to adaptively clamp weights of the encoder $f_{\boldsymbol{\Theta}_f}$ based on the learned masking probability distribution from the generator $g_{\boldsymbol{\Theta}_g}$. After bi-level optimization, only the sensitive and its highly-correlated features would have a higher probability to be masked and therefore, declining their contributions in $\widetilde{\mathbf{H}}^L$ by clamping their corresponding weights in the encoder would discourage the encoder from capturing these features and hence alleviate the discrimination. Concretely, within each training epoch after the bi-level optimization, we compute the probability of keeping features $\mathbf{p} \in \mathbb{R}^d$ by sampling K masks and calculate their mean $\mathbf{p} = \sum_{k=1}^{K} \mathbf{m}^k \in \mathbb{R}$. Then, assuming the weights of the first layer in the encoder $f_{\boldsymbol{\Theta}_f}$ is $\mathbf{W}^{f,1} \in \mathbb{R}^{d_1 \times d}$, we clamp it by:

$$\mathbf{W}_{ij}^{f,1} = \begin{cases} \mathbf{W}_{ij}^{f,1}, & |\mathbf{W}_{ij}^{f,1}| \le \epsilon * \mathbf{p}_j \\ \mathrm{sign}(\mathbf{W}_{ij}^{f,1}) * \epsilon * \mathbf{p}_j, & |\mathbf{W}_{ij}^{f,1}| > \epsilon * \mathbf{p}_j \end{cases}, \tag{8.11}$$

where $\epsilon \in \mathbb{R}$ is a prefix cutting threshold selected by hyperparameter tuning and $\mathrm{sign} : \mathbb{R} \to \{-1, 0, 1\}$ takes the sign of $\mathbf{W}_{ij}^{f,1}$. Intuitively, feature channels masked with higher probability (remained with lower probability $\mathbf{p}_j$) would have a lower threshold in weight clamping and hence their contributions to the representations $\widetilde{\mathbf{H}}^L$ are weakened.

Next, we theoretically rationalize this adaptive weight clamping by demonstrating its equivalence to minimizing the upper bound of the difference of representations between two sensitive groups:

Theorem 6. *Given a 1-layer GNN encoder $f_{\boldsymbol{\theta}_f}$ with row-normalized adjacency matrix $\mathbf{D}^{-1}\mathbf{A}$ as the PROP and weight matrix $\mathbf{W}^{f,1}$ as TRAN and further assume that features of nodes from two sensitive groups in the network independently and identically follow two different Gaussian distributions, i.e., $\mathbf{X}^{s_1} \sim \mathcal{N}(\boldsymbol{\mu}^{s_1}, \boldsymbol{\Sigma}^{s_1})$, $\mathbf{X}^{s_2} \sim \mathcal{N}(\boldsymbol{\mu}^{s_2}, \boldsymbol{\Sigma}^{s_2})$, then the difference of representations $\mathbf{H}^{s_1} - \mathbf{H}^{s_2}$ also follows a Gaussian with the 2-*

norm of its mean $\boldsymbol{\mu}$ *as:*

$$||\boldsymbol{\mu}||_2 = ||(2\chi - 1)\mathbf{W}^{f,1}\Delta\boldsymbol{\mu}||_2 \leq (2\chi - 1)\big(\sum_{i=1}^{d_1}(\sum_{r\in\mathcal{S}}\epsilon\mathbf{p}_r\Delta\boldsymbol{\mu}_r + \sum_{k\in\mathcal{NS}}\epsilon\mathbf{p}_k\Delta\boldsymbol{\mu}_k)^2\big)^{0.5} \tag{8.12}$$

where $\Delta\boldsymbol{\mu} = \boldsymbol{\mu}^{s_1} - \boldsymbol{\mu}^{s_2} \in \mathbb{R}^d$ and $\mathcal{S}, \mathcal{NS}$ denote the sensitive and non-sensitive features, and χ is the network homophily.

Proof. Substituting the row-normalized adjacency matrix $\mathbf{D}^{-1}(\mathbf{A}+\mathbf{I})$, we have $f_{\boldsymbol{\theta}_f}(\mathbf{X}) = \mathbf{W}^{f,1}\mathbf{D}^{-1}(\mathbf{A}+\mathbf{I})\mathbf{X}$, for any pair of nodes coming from two different sensitive groups $v_i \in \mathcal{V}_0, v_j \in \mathcal{V}_1$, we have:

$$\begin{aligned} f_{\boldsymbol{\theta}_f}(\mathbf{X}_i) - f_{\boldsymbol{\theta}_f}(\mathbf{X}_j) &= \mathbf{W}^{f,1}\big(\mathbf{D}^{-1}(\mathbf{A}+\mathbf{I})\mathbf{X}\big)_i - \mathbf{W}^{f,1}\big(\mathbf{D}^{-1}(\mathbf{A}+\mathbf{I})\mathbf{X}\big)_j \\ =&\mathbf{W}^{f,1}\Big(\frac{1}{d_i+1}\sum_{v_p\in\mathcal{N}_i\cup v_i}\mathbf{X}_p - \frac{1}{d_j+1}\sum_{v_q\in\mathcal{N}_j\cup v_j}\mathbf{X}_q\Big), \end{aligned} \tag{8.13}$$

if the network homophily is χ and further assuming that neighboring nodes strictly obey the network homophily, i.e., among $|\mathcal{N}_i \cup v_i| = d_i + 1$ neighboring nodes of the center node v_i, $\chi(d_i+1)$ of them come from the same feature distribution as v_i while $(1-\chi)(d_i+1)$ of them come from the other feature distribution as v_j, then symmetrically we have:

$$\begin{aligned} \frac{1}{d_i+1}\sum_{v_p\in\mathcal{N}_i\cup v_i}\mathbf{X}_p &\sim \mathcal{N}\big(\chi\boldsymbol{\mu}^{s_1} + (1-\chi)\boldsymbol{\mu}^{s_2}, (d_i+1)^{-1}(\chi\boldsymbol{\Sigma}^{s_1} + (1-\chi)\boldsymbol{\Sigma}^{s_2})\big), \\ \frac{1}{d_j+1}\sum_{v_q\in\mathcal{N}_j\cup v_j}\mathbf{X}_q &\sim \mathcal{N}\big(\chi\boldsymbol{\mu}^{s_2} + (1-\chi)\boldsymbol{\mu}^{s_1}, (d_j+1)^{-1}(\chi\boldsymbol{\Sigma}^{s_2} + (1-\chi)\boldsymbol{\Sigma}^{s_1})\big). \end{aligned} \tag{8.14}$$

Combining Eq. (8.14) and Eq. (8.13), the distribution of their difference would also be a Gaussian $f_{\boldsymbol{\theta}_f}(\mathbf{X}_i) - f_{\boldsymbol{\theta}_f}(\mathbf{X}_j) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where:

$$\boldsymbol{\mu} = \mathbf{W}^{f,1}\big(\chi\boldsymbol{\mu}^{s_1} + (1-\chi)\boldsymbol{\mu}^{s_2} - \chi\boldsymbol{\mu}^{s_2} - (1-\chi)\boldsymbol{\mu}^{s_1}\big) = (2\chi-1)\mathbf{W}^{f,1}\Delta\boldsymbol{\mu} \tag{8.15}$$

$$\boldsymbol{\Sigma} = \mathbf{W}^{f,1}\big((d_i+1)^{-1}(\chi\boldsymbol{\Sigma}^{s_1} + (1-\chi)\boldsymbol{\Sigma}^{s_2}) + (d_j+1)^{-1}(\chi\boldsymbol{\Sigma}^{s_2} + (1-\chi)\boldsymbol{\Sigma}^{s_1})\big)\mathbf{W}^{f,1\top} \tag{8.16}$$

Taking the $2-$norm on the mean $\boldsymbol{\mu}$, splitting channels into sensitive ones $\mathcal{S}$ and non-sensitive ones $\mathcal{NS}$, i.e., $\{1, 2, ..., d\} = \mathcal{S} \cup \mathcal{NS}$ and expanding $\boldsymbol{\mu}$ based on the input channel, we have:

$$||(2\chi-1)\mathbf{W}^{f,1}\Delta\boldsymbol{\mu}||_2 = (2\chi - 1)\big(\sum_{i=1}^{d_1}(\sum_{r\in\mathcal{S}}\mathbf{W}_{ir}^{f,1}\Delta\boldsymbol{\mu}_r + \sum_{k\in\mathcal{NS}}\mathbf{W}_{ik}^{f,1}\Delta\boldsymbol{\mu}_k)^2\big)^{0.5}, \tag{8.17}$$

where $\mathbf{W}_{ir}^{f,1}, \mathbf{W}_{ik}^{f,1}$ represent the weights of the encoder from feature channel $r(k)$ to the hidden neuron i. Since we know that $|\mathbf{W}_{ir}^{f,1}| \leq \epsilon\mathbf{p}_r, |\mathbf{W}_{ik}^{f,1}| \leq \epsilon\mathbf{p}_k, \forall r \in \mathcal{S}, k \in \mathcal{NS}$, we substitute the upper bound here into Eq. (8.17) and finally end up with:

$$||\boldsymbol{\mu}||_2 = ||(2\chi - 1)\mathbf{W}^{f,1}\Delta\boldsymbol{\mu}||_2 \leq (2\chi - 1)\big(\sum_{i=1}^{d_1}(\sum_{r\in\mathcal{S}}\epsilon\mathbf{p}_r\Delta\boldsymbol{\mu}_r + \sum_{k\in\mathcal{NS}}\epsilon\mathbf{p}_k\Delta\boldsymbol{\mu}_k)^2\big)^{0.5}.$$

□

Algorithm 2: The algorithm of FairVGNN

```
Input: an attributed graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{A}, \mathbf{Y})$, Classifier $c_{\Theta_c}$, Encoder $f_{\Theta_f}$, Generator $g_{\Theta_g}$, Discriminator $d_{\Theta_d}$, $K$
Output: Learned fairness attribute $\widetilde{\mathbf{X}}$ and Predictions $\hat{\mathbf{Y}}$
1  while not converged do
2      $\boldsymbol{\pi} \leftarrow \mathbf{W}^{g\Theta_g}$
3      for $k \leftarrow 1$ to $K$ do
4          $\mathbf{m}^k \sim$ Gumbel-softmax($\boldsymbol{\pi}$) , $\widetilde{\mathbf{X}}^k \leftarrow \mathbf{X} \odot \mathbf{m}^k$,            // Section 8.5.1
5          $\widetilde{\mathbf{H}}_i^{L,k} \leftarrow f_{\Theta_f}(\widetilde{\mathbf{X}}^k, \mathbf{A})$, $\widehat{\mathbf{H}}_i^{L,k} \leftarrow$ stopgrad($\widetilde{\mathbf{H}}_i^{L,k}$)            // Section 8.5.1[5]
6      for epoch $\leftarrow 1$ to epoch$_d$ do
7          $\mathcal{L}_{\mathrm{d}} \leftarrow \sum_{k=1}^{K} \sum_{v_i \in \mathcal{V}} [\mathbf{S}_i \log(d_{\Theta_d}(\widehat{\mathbf{H}}_i^{L,k})) + (1 - \mathbf{S}_i) \log(1 - d_{\Theta_d}(\widehat{\mathbf{H}}_i^{L,k}))]$
8          $\Theta_d \leftarrow \Theta_d + \nabla_{\Theta_d}\mathcal{L}_d$, $\Theta_f \leftarrow \Theta_f + \nabla_{\Theta_f}\mathcal{L}_f$            // Section 8.5.1
9      for epoch $\leftarrow 1$ to epoch$_c$ do
10         $\mathcal{L}_{\mathrm{c}} \leftarrow \sum_{k=1}^{K} \sum_{v_i \in \mathcal{V}} [\mathbf{Y}_i \log(c_{\Theta_c}(\widetilde{\mathbf{H}}_i^{L,k})) + (1 - \mathbf{Y}_i) \log(1 - c_{\Theta_c}(\widetilde{\mathbf{H}}_i^{L,k}))]$
11         $\Theta_c \leftarrow \Theta_c - \nabla_{\Theta_c}\mathcal{L}_c$, $\Theta_f \leftarrow \Theta_f - \nabla_{\Theta_f}\mathcal{L}_f$            // Section 8.5.1
12     for epoch $\leftarrow 1$ to epoch$_g$ do
13         $\mathcal{L}_{\mathrm{g}}^k \leftarrow \sum_{k=1}^{K} \sum_{v_i \in \mathcal{V}} ||d_{\Theta_d}(\widetilde{\mathbf{H}}_i^{L,k}) - 0.5||_2^2$,
14         $\Theta_g \leftarrow \Theta_g - \nabla_{\Theta_g}\mathcal{L}_g$, $\Theta_f \leftarrow \Theta_f - \nabla_{\Theta_f}\mathcal{L}_f$            // Section 8.5.1
15     $\Theta_f \leftarrow$ Clamp($\Theta_f$, $\sum_{k=1}^{K} \mathbf{m}^k$)            // Section 8.5.2
16 $\widetilde{\mathbf{X}} = \sum_{k=1}^{K} \widetilde{\mathbf{X}}^k$,  $\hat{\mathbf{Y}} = c_{\Theta_c}(f_{\Theta_f}(\widetilde{\mathbf{X}}, \mathbf{A}))$
17 return $\widetilde{\mathbf{X}}, \widehat{\mathbf{Y}}$
```

The left side of Eq. (8.12) is the difference of representations between two sensitive groups, and if it is large, i.e., $||\boldsymbol{\mu}||_2$ is very large, then the predictions between these two groups would also be very different, which reflects more discrimination in terms of the group fairness. Additionally, Theorem 6 indicates that the upper bound of the group fairness between two sensitive groups depends on the network homophily χ, the initial feature difference $\Delta\boldsymbol{\mu}$ and the masking probability $\mathbf{p}$. As the network homophily χ decreases, more neighboring nodes come from the other sensitive group, and aggregating information of these neighborhoods would smooth node representations between different sensitive groups and reduce the bias. To the best of our knowledge, this is the first work relating fairness with the network homophily. Furthermore, Eq. (8.12) proves that clamping weights of the encoder $\mathbf{W}^{f,1}$ upper bounds the group fairness.

8.5.3 Training Algorithm

Compared to vanilla bi-level optimization, additional computational of FairVGNN comes from generating K different masks. However, since within each training epoch, we can pre-compute the masks as Step 4 before bi-level optimization and the total number of views K becomes constant compared with the whole time used for bi-level optimization as Step 6-14, the time complexity is still linear proportional to the size of the whole graph, i.e., $O(|\mathcal{V}| + |\mathcal{E}|)$. The total model complexity includes parameters of the feature masker $O(2d)$, the discriminator/classifier $O(2d^L)$ and the encoder $O(d\prod_{l=1}^{L} d^l)$, which boils down to $O(\max_{i\in\{0,\ldots,L\}}(d^i)^L)$, the same as any other L-layer GNN backbones.

8.6 Experiment

8.6.1 Experimental Settings

Datasets. We validate the approach on three benchmark datasets [218, 222] with their statistics shown below.

Table 8.2: Basic dataset statistics for bias evaluation in node classification.

Dataset	**German**	**Credit**	**Bail**
# Nodes	1000	30,000	18,876
# Edges	22,242	1,436,858	321,308
# Features	27	13	18
Sens.	Gender	Age	Race
Label	Good/bad Credit	Default/no default Payment	Bail/no bail

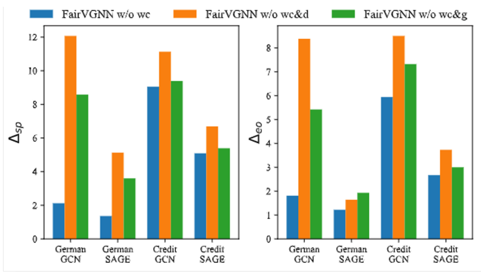
Since different GNN-backbones may cause different levels of sensitive attribute leakage, we consider equipping each of the above three bias-alleviating methods with three GNN-backbones: GCN [15], GIN [18], GraphSAGE [43], e.g., GCN-NIFTY represents the GCN encoder with NIFTY.

Setup. Our proposed FairVGNN is implemented using PyTorch-Geometric [47]. For EDITS[6], NIFTY[7] and FairGNN[8], we use the original code from the authors' GitHub repository. We aim to provide a rigorous and fair comparison between different models on each dataset by individually tuning hyperparameters for all models. The detailed hyperparameter configuration of each baseline is in Appendix 8.8.1. Following [218] and [222], we use 1-layer GCN, GIN convolution and 2-layer GraphSAGE convolution respectively as our encoder f_{Θ_f}, and use 1 linear layer as our classifier c_{Θ_c} and discriminator d_{Θ_d}. The detailed GNN architecture is described in Appendix 8.8.1. We fix the number of hidden units of the encoder f_{Θ_f} as 16, the dropout rate as 0.5, and the number of generated fair feature views during each training epoch $K = 10$. The learning rates and the training epochs of the generator g_{Θ_g}, the discriminator d_{Θ_d}, the classifier c_{Θ_c} and the encoder f_{Θ_f} are searched from $\{0.001, 0.01\}$ and $\{5, 10\}$, the prefix cutting threshold ϵ in Eq. (8.11) is searched from $\{0.01, 0.1, 1\}$, the whole training epochs as $200, 300, 400$, and $\alpha \in \{0, 0.5, 1\}$. We use the default data splitting following [218, 222], and experimental results are averaged over five repeated executions with five different seeds to remove any potential initialization bias.

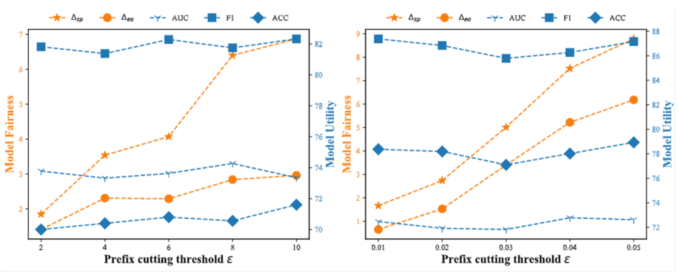
8.6.2 Node Classification

Performance comparison The model utility and fairness of each baseline is shown in Table 8.3. We observe that our FairVGNN consistently performs the best compared with other bias-alleviating methods in terms of the average rank for all datasets and across all evaluation metrics, which indicates the superiority of our model in achieving a better trade-off between model utility and fairness. Since no fairness regularization is

[6]https://github.com/yushundong/edits
[7]https://github.com/chirag126/nifty
[8]https://github.com/EnyanDai/FairGNN



(a) Model bias without the discriminator/generator



(b) Impact of different prefix-cutting thresholds

Figure 8.3: (**a**): Model bias without discriminator/generator. (**b**): Results of prefix cutting threshold.

imposed on GNN encoders equipped with vanilla methods, they generally achieve better model utility. However, for this reason, sensitive-related information is also completely free to be encoded in the learned node representations and hence causes higher bias. To alleviate such discrimination, all other methods propose different regularizations to constrain sensitive-related information in learned node representations, removing some task-related information and hence sacrificing model utility as expected in Table 8.3. However, we do observe that our model can yield lower biased predictions with less utility sacrifice, which is mainly ascribed to two reasons. Firstly, we generate different fair feature views by randomly sampling masks from learned Gumbel-Softmax distribution and make predictions. This can be regarded as a data augmentation technique by adding noise to node features, which decreases the population risk and enhances the model generalibility [193] by creating novel mapping from augmented training points to the label space. Secondly, the weight clamping module clamps weights of encoder based on feature correlations to the sensitive feature channel, which adaptively remove/keep the sensitive/task-relevant information.

Ablation study Next, we conduct the ablation study to fully understand the effect of each component of FairVGNN on alleviating discrimination. Concretely, we denote **FairV w/o fm** as removing the module of generating fair feature views, **FairV w/o wc** as removing the module of adaptive weight clamping, and **FairV w/o fm&wc** as removing both of these two modules. Since computing thresholds in adaptive weight clamping needs the probability of feature masking from fair feature view generation in Eq. (8.11), we instead directly take the prefix value ϵ without $\mathbf{p}_i$ as our cutting threshold in *FairV w/o fm*. The utility and bias of these variants are presented in Table 8.4. We observe that *FairV w/o fm* and *FairV w/o wc* perform worse than *FairV*, which validates the effectiveness of different components in *FairV* for learning fair node representations. Furthermore, the worse performance of *FairV w/o fm&wc* than *FairV w/o fm* and *FairV w/o wc* indicates the proposed two modules alleviate discrimination from two different aspects and their effects could be accumulated together. In most cases, *FairV w/o fm* achieves more bias than *FairV w/o wc*. This is because the original clamping threshold of sensitive feature channels $\epsilon * \mathbf{p}_i$ would be replaced by a higher threshold ϵ, which allows more sensitive information leakage to predictions.

Table 8.3: Model utility and bias of node classification. We compare the proposed FairVGNN (i.e., FairV) against state-of-the-art baselines NIFTY, EDITS, and FairGNN (i.e., Fair) when equipped with various GNN backbones (i.e., GCN, GIN, and SAGE). The best and runner-up results are **bolded** and underlined.

Encoder	Method	German					Credit					Bail					Avg.
		AUC (↑)	F1 (↑)	ACC (↑)	Δ_{sp} (↓)	Δ_{eo} (↓)	AUC (↑)	F1 (↑)	ACC (↑)	Δ_{sp} (↓)	Δ_{eo} (↓)	AUC (↑)	F1 (↑)	ACC (↑)	Δ_{sp} (↓)	Δ_{eo} (↓)	(Rank)
GCN	Vanilla	74.11±0.37	82.46±0.89	73.44±1.09	35.17±7.27	25.17±5.89	73.87±0.02	81.92±0.02	73.67±0.03	12.86±0.09	10.63±0.13	87.08±0.35	79.02±0.74	84.56±0.68	7.35±0.72	4.96±0.62	9.17
	NIFTY	68.78±2.69	81.40±0.54	69.92±1.14	5.73±5.25	5.08±4.29	71.96±0.19	81.72±0.05	73.45±0.06	11.68±0.07	9.39±0.07	78.20±2.78	64.76±3.91	74.19±2.57	2.44±1.29	1.72±1.08	9.69
	EDITS	69.41±2.33	81.55±0.59	71.60±0.89	4.05±4.48	3.89±4.23	73.01±0.11	81.81±0.28	73.51±0.30	10.90±1.22	8.75±1.21	86.44±2.17	75.58±3.77	84.49±2.27	6.64±0.39	7.51±1.20	9.89
	FairGNN	67.35±2.13	82.01±0.26	69.68±0.30	3.49±2.15	3.40±2.15	71.95±1.43	81.84±1.19	73.41±1.24	12.64±2.11	10.41±2.03	87.36±0.90	77.50±1.69	82.94±1.67	6.90±0.17	4.65±0.14	9.17
	FairVGNN	72.41±2.10	82.14±0.42	70.16±0.86	1.71±1.68	0.88±0.58	71.34±0.41	87.08±0.74	78.04±0.33	5.02±5.22	3.60±4.31	85.68±0.37	79.11±0.33	84.73±0.46	6.53±0.67	4.95±1.22	5.67
GIN	Vanilla	72.71±1.44	82.78±0.50	**73.84±0.54**	13.56±5.23	9.47±4.49	74.36±0.21	82.28±0.64	74.02±0.73	14.48±2.44	12.35±2.86	86.14±0.25	76.49±0.57	81.70±0.67	8.55±1.61	6.99±1.51	9.56
	NIFTY	67.61±4.88	80.46±3.06	69.92±3.64	5.26±3.24	5.34±5.67	70.90±0.24	84.05±0.82	75.59±0.66	7.09±4.62	6.22±3.26	82.33±4.61	70.64±6.73	74.46±9.98	5.57±1.11	3.41±1.43	8.56
	EDITS	69.35±1.64	82.80±0.22	72.08±0.66	0.86±0.76	1.72±1.14	72.35±1.11	82.47±0.85	74.07±0.98	14.11±14.45	15.40±15.76	80.19±4.62	68.07±5.30	73.74±5.12	6.71±2.35	5.98±3.66	11.36
	FairGNN	72.95±0.82	**83.16±0.56**	72.24±1.44	6.88±4.42	2.06±1.46	68.66±4.48	79.47±5.29	70.33±5.50	4.67±3.06	3.94±1.49	86.14±0.89	73.67±1.17	77.90±2.21	6.33±1.49	4.74±1.64	7.64
	FairVGNN	71.65±1.90	82.40±0.14	70.16± 0.32	**0.43±0.54**	**0.34±0.41**	71.36±0.72	87.44±0.23	78.18±0.20	**2.85±2.01**	**1.72±1.80**	83.22±1.60	76.36±2.20	83.86±1.57	5.67±0.76	5.77±1.26	5.44
SAGE	Vanilla	**75.74±0.69**	81.25±1.72	72.24±1.61	24.30±6.93	15.55±7.59	**74.58±1.31**	83.38±0.77	75.28±0.83	15.65±1.30	13.34±1.34	90.71±0.69	80.99±0.55	86.72±0.48	2.16±1.53	**0.84±0.55**	7.31
	NIFTY	72.05±2.15	79.20±1.19	69.60±1.50	7.74±7.80	5.17±2.38	72.89±0.44	82.60±1.25	74.39±1.35	10.65±1.65	8.10±1.91	**92.04±0.89**	77.81±6.03	84.11±5.49	5.74±0.38	4.07±1.28	8.06
	EDITS	69.76±5.46	81.04±1.09	71.68±1.25	8.42±7.35	5.69±2.16	75.04±0.12	82.41±0.52	74.13±0.59	11.34±6.36	9.38±5.39	89.07±2.26	77.83±3.79	84.42±2.87	3.74±3.54	4.46±3.50	11.36
	FairGNN	65.85±9.49	82.29±0.32	70.64±0.74	7.65±8.07	4.18±4.86	70.82±0.74	83.97±2.00	75.29±1.62	6.17±5.57	5.06±4.46	91.53±0.38	82.55±0.98	87.68±0.73	1.94±0.82	1.72±0.70	5.83
	FairVGNN	73.84±0.52	81.91±0.63	70.00±0.25	1.36±1.90	1.22±1.49	74.05±0.20	**87.84±0.32**	**79.94±0.30**	4.94±1.10	2.39±0.71	91.56±1.71	**83.58±1.88**	**88.41±1.29**	**1.14±0.67**	1.69±1.13	**2.92**

Table 8.4: Model utility and bias of node classification of different variants of FairVGNN. The best and runner-up results are **bolded** and underlined.

Encoder	Model Variants	German					Credit					Bail				
		AUC (↑)	F1 (↑)	ACC (↑)	Δ_{sp} (↓)	Δ_{eo} (↓)	AUC (↑)	F1 (↑)	ACC (↑)	Δ_{sp} (↓)	Δ_{eo} (↓)	AUC (↑)	F1 (↑)	ACC (↑)	Δ_{sp} (↓)	Δ_{eo} (↓)
GCN	FairV	72.69± 1.67	81.86± 0.49	69.84±0.41	0.77± 0.39	0.46± 0.34	71.34±0.41	87.08±0.74	78.04±0.33	5.02±5.22	3.60±4.31	85.68±0.37	79.11±0.33	84.73±0.46	6.53±0.67	4.95±1.22
	FairV w/o fm	73.63± 1.14	82.28±0.28	70.88±1.09	5.56±3.89	4.41±3.59	72.51±0.32	86.15±2.18	77.83±2.15	6.94±2.86	4.64±2.73	86.98±0.32	78.08±0.53	84.59±0.29	7.24±0.26	5.75±0.68
	FairV w/o wc	72.08± 1.83	82.72± 0.50	71.04± 1.23	3.19± 3.51	0.59± 1.12	71.80±0.47	87.27±0.47	78.47±0.34	9.05±4.55	5.94±3.61	85.93±0.38	79.22±0.29	85.38±0.25	6.61±0.48	5.82±0.66
	FairV w/o fm&wc	74.97±0.94	82.30±0.67	70.8±0.88	7.74±5.05	4.56±4.15	73.09±0.41	84.48±2.14	76.40±2.29	11.91±2.34	9.27±1.98	86.44±0.16	78.75±0.27	84.41±0.28	8.32±0.60	6.34±0.32
GIN	FairV	71.65±1.90	82.40±0.14	70.16±0.32	**0.43±0.54**	**0.34±0.41**	71.36±0.72	87.44±0.23	78.18±0.20	**2.85±2.01**	1.72±1.80	83.22±1.60	76.36±2.20	83.86±1.57	5.67±0.76	5.77±1.26
	FairV w/o fm	73.76±0.77	83.06±0.67	71.68±1.63	2.76±2.64	0.57±0.47	71.15±0.63	87.09±0.7	78.29±0.53	3.36±2.34	1.86±1.19	85.12±0.54	77.06±0.83	83.13±1.19	6.80±0.28	5.97±0.64
	FairV w/o wc	72.65±1.65	82.70±0.30	71.20±1.01	3.44±3.19	0.97±0.9	71.13±0.59	87.96±0.25	80.04±0.22	3.16±1.28	**1.47±0.72**	85.09±2.36	79.07±2.70	85.85±2.13	5.24±1.41	4.33±2.05
	FairV w/o fm&wc	73.41±1.17	**83.20±0.44**	**72.40±1.29**	5.70±4.57	1.01±1	72.73±0.32	86.10±0.59	77.90±0.63	6.66±1.10	3.97±0.41	86.32±1.60	79.28±1.39	**86.02±0.40**	7.48±0.71	7.43±2.38
SAGE	FairV	73.84±0.52	81.91±0.63	70.00±0.25	1.36±1.90	1.22±1.49	74.05±0.20	87.84±0.32	79.94±0.30	4.94±1.10	2.39±0.71	91.56±1.71	83.58±1.88	88.41±1.29	**1.14±0.67**	1.69±1.13
	FairV w/o fm	73.98±1.40	81.36±1.45	70.00±1.50	3.67±2.80	1.55±2.01	73.58±0.68	83.18±2.32	74.97±2.49	7.23±3.91	5.05±3.17	91.96±0.57	84.04±1.01	88.69±0.79	1.51±1.17	1.59±0.35
	FairV w/o wc	73.93±2.16	82.02±0.72	70.16±1.25	2.80±2.79	0.90±1.06	74.05±0.42	**88.10±0.30**	**80.16±0.19**	5.09±1.30	2.67±0.92	92.01±0.74	84.64±0.91	**89.24±0.58**	2.99±0.94	**1.07±1.19**
	FairV w/o fm&wc	73.87±1.62	80.09±1.73	70.08±1.17	6.18±1.31	4.68±2.38	**74.57±0.14**	81.91±0.92	73.61±1.02	7.27±3.22	5.03±3.01	**92.05±0.89**	83.40±1.79	88.44±1.02	3.51±0.87	2.05±1.19

Table 8.5: Comparison with different weight regularization with best results in **bold**.

Dataset (Model)	**Strategy**	AUC (↑)	ACC (↑)	F1 (↑)	Δ_{sp} (↓)	Δ_{eo} (↓)
German (SAGE)	Ad wc	**73.84+0.52**	70.00+0.25	81.91+0.63	**1.36+1.90**	**1.22+1.49**
	Wc	72.43+1.60	**70.48+0.85**	**82.03+0.82**	4.85+4.10	2.50+2.12
	Sn	73.00+1.53	70.00+1.07	81.82+0.59	3.74+3.22	1.89+1.08
Credit (GIN)	Ad wc	**74.05±0.20**	**79.94±0.19**	**87.84±0.32**	4.94±1.10	2.39±0.71
	Wc	73.20±1.20	79.03±1.09	87.23±0.94	7.03±4.58	4.74±3.47
	Sn	71.12±0.55	78.54±2.00	86.53±1.90	**2.60±0.73**	**0.87±0.54**
Bail (GCN)	Ad wc	85.68+0.37	84.73+0.46	79.11+0.33	**6.53+0.67**	**4.95+1.22**
	Wc	85.97+0.45	85.12+0.26	79.08+0.28	6.86+0.47	5.85+0.83
	Sn	**86.10+0.61**	**85.69+0.42**	**79.66+0.63**	7.53+0.17	6.43+0.81

* **Ad wc**: adaptively clamp weights of the encoder; **Wc**: clamp weights of the encoder; **Sn**: spectral normalization of the encoder

8.6.3 Further Probe

Does bi-level optimization work? We first remove the weight clamping to solely study the effect of bi-level optimization training, and then remove the discriminator/generator respectively by setting their corresponding training epochs to be 0 and denote the corresponding models as **FairVGNN w/o wc&d** and **FairVGNN w/o wc&g**. We re-conduct the node classification with five different initializations following the previous setting and report the average bias in Figure 8.3(a)-(b). We can see that after removing the discriminator or generator, the model bias becomes even higher in both situations, which indicates the importance of the competition between the discriminator and the generator in improving the discriminative power of the discriminator to recognize sensitive features and generating power of generator to generate fair feature views. Moreover, since the discriminator in *FairVGNN w/o wc&g* can still recognize the sensitive features and then guide the encoder to extract less sensitive-related information, the bias of *FairVGNN w/o wc&g* is lower than *FairVGNN w/o wc&d* in most cases.

Does adaptive weight clamping work? To demonstrate the advantages of the proposed adaptive weight clamping, we compare it with the non-adaptive weight clamping and spectral normalization, another technique of regularizing weight matrix to enhance the model robustness and counterfactual fairness [218]. The prefix-cutting thresholds in both the adaptive and non-adaptive weight clamping are set to be the same as the best ones tuned in Table 8.3 for SAGE/GCN/GIN to ensure a fair comparison. As shown in Table 8.5, we can see that except for GIN, the adaptive weight clamping consistently achieves lower bias while not hurting so much model utility. This is because for sensitive-related feature channels, multiplying the masking probability by the prefix threshold would lower the threshold and prevent more sensitive information from leaking to prediction through the encoder. We also investigate the influence of prefix cutting threshold ϵ in Eq. (8.11) on the model bias/utility. Higher ϵ indicates less weight clamping on the encoder, and more sensitive-related information is leveraged in predictions, which leads to higher bias.

8.7 Conclusion

In this chapter, we focus on alleviating discrimination in learned node representations and make predictions on graphs from the perspective of sensitive leakage to innocuous features. Specifically, we empirically observe a novel problem: feature propagation could vary feature correlation and cause sensitive leakage to innocuous feature channels, which may exacerbate discrimination in predictions. To tackle this problem, we propose FairVGNN to automatically mask sensitive-correlated feature channels considering the effect of correlation variation after feature propagation and adaptively clamp weights of the encoder to absorb less sensitive information. Experimental results demonstrate the effectiveness of the proposed FairVGNN framework in achieving a better trade-off between utility and fairness than other baselines. Some interesting phenomena are also observed, such as the correlation variation depending on different datasets, and the group fairness is related to the network homophily. Thus, one future direction would be to theoretically analyze the relationships among feature propagation, network homophily, and correlation variation.

8.8 Appendix

8.8.1 Experimental Settings

Detailed Model Architecture A unified template of a graph convolutional layer is formalized as follows:

$$\mathbf{h}_i^l = \text{TRAN}^l(\text{PROP}^l(\mathbf{h}_i^{l-1}, \{\mathbf{h}_j^{l-1} | j \in \mathcal{N}_i\})), \tag{8.18}$$

where $\mathcal{N}_i$ denotes the neighborhood set of node v_i and $\text{PROP}^l, \text{TRAN}^l$ stand for neighborhood propagation and feature transformation at layer l. In neighborhood propagation, neighborhood representations are propagated and further fused with itself to get the intermediate representation $\widehat{\mathbf{h}}_i^l$. Then, the TRAN^l function is applied on $\widehat{\mathbf{h}}_i^l$ to get the final representation $\mathbf{h}_i^l$ of node v_i at layer l. Note that $\mathbf{h}_i^0$ of node v_i is typically initialized as the original node feature $\mathbf{X}_i$. After stacking L graph convolutional layers, every node aggregates their neighborhood information up to L-hops away, and we denote it as $\mathbf{H}^L \in \mathbb{R}^{n \times d^L}$. Many graph convolutions can be obtained under this template by configuring different PROP^l and TRAN^l. In this work, the encoder of FairVGNN is designed following this template.

We use GCN, GIN, and GraphSAGE as our GNN backbones for each bias-alleviating method. The basic graph convolution layer of these three backbones, respectively, are:

$$\mathbf{H}^l = \widetilde{\mathbf{D}}^{-0.5}(\mathbf{A} + \mathbf{I})\widetilde{\mathbf{D}}^{-0.5}\mathbf{H}^{l-1}\mathbf{W}^l, \tag{8.19}$$

$$\mathbf{H}^l = \text{MLP}^l((\mathbf{A} + (1 + \alpha)\mathbf{I})\mathbf{H}^{l-1}), \tag{8.20}$$

$$\mathbf{H}^{l} = \mathbf{W}^{l,1}\mathbf{H}^{l-1} + \mathbf{W}^{l,2}\mathbf{D}^{-1}\mathbf{A}\mathbf{H}^{l-1}, \tag{8.21}$$

where $\widetilde{\mathbf{D}}$ is the degree matrix with added self-loop, $\mathbf{H}^{l-1}$ is the node representation obtained from the previous layer and $\mathbf{H}^{0} = \mathbf{X}$. In this work, we only consider one graph convolution, therefore $l = 1$.

Hyperparameter for Each Baseline As different bias-alleviating methods have different model architectures, their hyperparameters are also different and are presented respectively in the following:

- **NIFTY**: dropout $\{$0.0, 0.5, 0.8$\}$, the number of hidden unit 16, learning rate $\{1e^{-2}, 1e^{-3}, 1e^{-4}\}$, project hidden unit 16, weight decay $\{1e^{-4}, 1e^{-5}\}$, drop edge rate 0.001, drop feature rate 0.1, regularization coefficient $\{0.4, 0.5, 0.6, 0.7, 0.8\}$.

- **EDITS**: initial learning rate 0.003, weight decay $1e^{-7}$, threshold proportions for Credit, German, and Recidivism dataset are 0.02, 0.25, 0.012 respectively.

- **FairGNN**: dropout $\{0.0, 0.5, 0.8\}$, the number of hidden unit 32, learning rate $\{0.0001, 0.001, 0.01\}$, weight decay $1e^{-5}$, regularization coefficients $\alpha = 4, \beta = 0.01$, sensitive and label number $200, 500$.

CHAPTER 9

Bias Issue: Discovering the Degree-related Evaluation Bias in Link Prediction

Link prediction is a fundamental problem for network-structured data and has achieved unprecedented success in many real-world applications. Despite the significant progress being made towards improving its performance by characterizing underlined topological patterns or leveraging representation learning, few works have focused on the imbalanced performance among nodes of different degrees. In this chapter, we propose a novel problem, degree-related bias and evaluation bias, on link prediction with an emphasis on recommender system applications. We first empirically demonstrate the performance difference among nodes with different degrees and then theoretically prove that Recall is an unbiased evaluation metric compared with F1, NDCG and Precision. Furthermore, we show that under the unbiased evaluation metric Recall, low-degree nodes tend to have higher performance than high-degree nodes in link prediction. [1]

9.1 Introduction

Graph-structured data is omnipresent in various fields, such as biology, chemistry, social media, and transportation [188, 232]. Link prediction, as one of the most important graph-related tasks, has become a central problem and finds its applications in predicting drug interactions, recovering knowledge graphs, and recommendations [51, 233].

As well-known in many graphs (e.g. citation graphs and social networks, etc.), node degree usually follows a power-law distribution. While the degree of major nodes is relatively small, few nodes on the long tail have significantly high-degree. Existing works [60, 234, 63] have shown that such power-law distributed node degree may hurt the performance of GNNs in node classification. Specifically, nodes with higher degrees are much more likely to own labeled neighbors compared with lower-degree ones and by message-passing mechanism, these nodes participate more frequently in the optimization and their learned representations are more predictive of their ground-truth labels. However, we argue that this conclusion does not hold in the task of link prediction.

On one hand, in link prediction, we are not given any golden label and hence message-passing may not cause imbalanced training/optimization between nodes of high and low degree. On the other hand, even given golden labels in link prediction, then each unique node would correspond to a unique label (we aim to correctly classify all neighbors of this node to be this unique class). Therefore, the more frequent participation of high-degree nodes in the optimization by message-passing, the more likely their representations would be

[1] https://ieeexplore.ieee.org/abstract/document/10031056

optimized towards pairing with so many unrelated nodes, and hence their performance would decrease. As shown in Figure 4.6, Recall@20 decreases when node degree increases, which aligns with our argument. Note that because Normalized Discounted Cumulative Gain (NDCG), unlike Recall, is a biased evaluation metric (as justified later in Section 9.2.1), we observe that NDCG@20 increases as the node degree increases.

9.2 Analyzing Bias in Link Prediction

Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph, where $\mathcal{V} = \{v_1, v_2, ..., v_n\}$ is the set of nodes with $n = |\mathcal{V}|$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges with $m = |\mathcal{E}|$. Given the historical edges $\bar{\mathcal{E}}$ that we have observed, most link predictors expect to predict the incoming edges $\hat{\mathcal{E}}$ with $\mathcal{E} = \bar{\mathcal{E}} \cup \hat{\mathcal{E}}$ by learning a mapping $\mathcal{V} \times \mathcal{V} \rightarrow \mathbf{S} \in \mathbb{R}^{n \times n}$, where $\mathbf{S}_{ij} \in \mathbb{R}$ represents how likely a link will form between v_i and v_j. The performance of each node v_i is evaluated by comparing the level of the alignment between its ground-truth 1-hop neighbors $\hat{\mathcal{N}}_i^1$ and its predicted 1-hop neighbors $\widetilde{\mathcal{N}}_i^1$. Specifically, for each node v_i, we sort its preference scores over all nodes $\mathbf{S}_i \in \mathbb{R}^n$ and select the top-K items to form its predicted 1-hop neighbors $\widetilde{\mathcal{N}}_i^1 = \{v_{\phi_i^k}\}_{k=1}^K$ where ϕ_i^k denotes v_i's k^{th} preferred item selected according to the rank of $\mathcal{S}_i$. Assuming $K < |\hat{\mathcal{N}}_i^1|$, then we formulate four commonly-used evaluation metrics Recall(R), Precision(P), F1 and NDCG(N) as:

$$\text{R@}K_i = \frac{|\hat{\mathcal{N}}_i^1 \cap \widetilde{\mathcal{N}}_i^1|}{|\hat{\mathcal{N}}_i^1|}, \quad \text{P@}K_i = \frac{|\hat{\mathcal{N}}_i^1 \cap \widetilde{\mathcal{N}}_i^1|}{K} \tag{9.1}$$

$$\text{F1@}K_i = 2\frac{\text{P@}K \cdot \text{R@}K}{\text{R@}K + \text{P@}K} = \frac{2|\hat{\mathcal{N}}_i^1 \cap \widetilde{\mathcal{N}}_i^1|}{K + |\hat{\mathcal{N}}_i^1|} \tag{9.2}$$

$$\text{N@}K_i = \frac{\sum_{k=1}^{K} \frac{\mathbb{1}[v_{\phi_i^k} \in (\hat{\mathcal{N}}_i^1 \cap \widetilde{\mathcal{N}}_i^1)]}{\log_2(k+1)}}{\sum_{k=1}{}^{K} \frac{1}{\log_2(k+1)}} \tag{9.3}$$

9.2.1 Theoretical Analysis

Before analyzing bias in link prediction with the evaluation metrics defined above, we first theoretically prove that Recall is an unbiased evaluation metric while Precision, F1, and NDCG are biased ones. Assuming that $|\hat{\mathcal{N}}_i^1 \cap \widetilde{\mathcal{N}}_i^1|$ follows hyper-geometric distribution for any node v_i and $|\hat{\mathcal{N}}_i^1| = d$, the relationship between the expectation of each evaluation and the node activity d is derived as:

Recall

$$E(\text{R@K}|d) = \frac{K}{n}, \qquad \frac{\partial E(\text{R@K}|d)}{\partial d} = 0, \tag{9.4}$$

Precision

$$E(\text{P@K}|d) = \frac{d}{n}, \qquad \frac{\partial E(\text{P@K}|d)}{\partial d} = \frac{1}{n}, \tag{9.5}$$

F1

$$E(\text{F1@K}|d) = \frac{2K}{n}\frac{d}{K+d}, \quad \frac{\partial E(\text{F1@K}|d)}{\partial d} = \frac{2K^2}{n}\frac{1}{(K+d)^2} \in (0,1), \tag{9.6}$$

NDCG

$$E(\text{N@K}|d) = \frac{d}{n}, \qquad \frac{\partial E(\text{N@K}|d)}{\partial d} = \frac{1}{n}. \tag{9.7}$$

For brevity, we put the detailed derivations for Eq. (9.4)-(9.7) in the Appendix. Precision, F1, and NDCG increase as the node degree d increases and hence lead to bias in evaluating the degree-related bias in link prediction. Note that although the node degree d is defined to be the size of the ground-truth neighborhood, the conclusion still holds since typically, nodes with high degrees in training data would also have high degrees in testing data assuming no degree distribution shift.

9.2.2 Empirical Analysis

We further empirically verify the above observation by leveraging an untrained link predictor to calculate the corresponding evaluation metric. More specifically, for each node v_i, we randomly select K nodes from $\mathcal{V}$ and check whether the selected K nodes come from $\hat{\mathcal{N}_i}$. To approximate the expectation with less error, we average the results over 200 runs. Straightforwardly, any untrained model should output exactly the same performance for each individual. However, it is clearly seen in Figure 9.1 that when evaluating with Precision, F1, and NDCG, the performance still increases as the node degree increases, which is consistent with what we derive in Section 9.2.1 and further demonstrates the evaluation bias embedded in Precision, F1, and NDCG.

9.3 Conclusion

In this chapter, we propose a novel issue, degree-related bias and evaluation bias, in link prediction. We first empirically demonstrate the imbalanced performance of link prediction on nodes with different degrees, which disclose the degree-related bias in link prediction. Then, we theoretically analyze the bias of different evaluation metrics and prove that NDCG, F1 and Precision are all biased towards high-degree nodes while Recall is the only unbiased evaluation metric. When evaluating under the unbiased

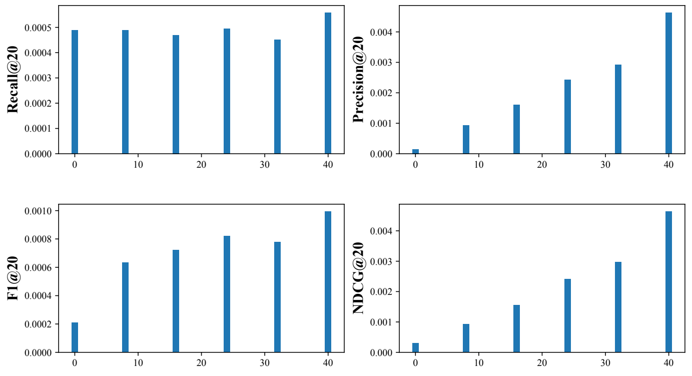


Figure 9.1: Performance under each metric w.r.t. node degrees on Gowalla.

metric Recall, we finally conclude that low-degree nodes tend to have higher performance in link prediction than high-degree nodes.

9.4 Appendix

Degree-related Bias of Evaluation Metrics.

One previous work [4] has empirically shown the degree-related bias of evaluation metrics used in link prediction models. Following that, we go one step further and theoretically derive the concrete format of the evaluation bias in this section. We leverage an untrained link prediction model to study the bias. This avoids any potential supervision signal from training over observed links and enables us to study the evaluation bias exclusively. Since two nodes with the same degree may end up with different performances, i.e., $\text{X@}K_i \neq \text{X@}K_j, d_i = d_j$, we model $\text{X@}K|d$ as a random variable and expect to find the relationship between its expectation and the node degree d, i.e., $f : E(\text{X@}K|d) = f(d)$.

Following many existing ranking works [79, 59], we assume without loss of generalizability that the link predictor $\mathscr{P}$ ranking the predicted neighbors based on their embedding similarity with embeddings noted as $\mathbf{E}$, then we have:

Lemma 2. *For any untrained embedding-based link predictor $\mathscr{P}$, given the existing $k-1$ predicted neighbors for the node $v_i \in \mathcal{V}$, the k^{th} predicted neighbor is generated by randomly selecting a node without replacement from the remaining nodes with equal opportunities, i.e., $P(v_{\phi_i^k} = v|\{v_{\phi_i^1}, v_{\phi_i^2}, ..., v_{\phi_i^{k-1}}\}) = \frac{1}{N-(k-1)}$.*

Without any training, Lemma 2 trivially holds since embeddings of all nodes are the same, which trivially leads to the following theorem:

Theorem 7. *Given the untrained embedding-based link predictor $\mathscr{P}$, the size of the intersection between any node's predicted list $\widetilde{\mathcal{E}}_i$ and its ground-truth list $\widehat{\mathcal{E}}_i$ follows a hypergeometric distribution: $|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i| \sim \mathcal{HG}(|\mathcal{V}|, K, |\widehat{\mathcal{E}}_i|)$ where $|\mathcal{V}|$ is the population size (the whole node space), K is the number of trials and $|\widehat{\mathcal{E}}_i|$ is the number of successful states (the number of node's ground-truth neighbors).*

Proof. Given the ground-truth node neighbors $\widehat{\mathcal{E}}_i$, the predicted neighbors $\widetilde{\mathcal{E}}_i = \{v_{\phi_i^k}\}_{k=1}^K$ is formed by selecting one node at a time without replacement K times from the whole node space $\mathcal{V}$. Since any selected node $v_{\phi_i^k}$ can be classified into one of two mutually exclusive categories $\widehat{\mathcal{E}}_i$ or $\mathcal{V}\backslash\widehat{\mathcal{E}}_i$ and by Lemma 2, we know that for any untrained link predictor, each unselected node has an equal opportunity to be selected in every new trial, we conclude that $|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i| \sim \mathcal{HG}(|\mathcal{V}|, K, |\widehat{\mathcal{E}}_i|)$ and by default $E(|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|) = |\widetilde{\mathcal{E}}_i|\frac{|\widehat{\mathcal{E}}_i|}{|\mathcal{V}|} = K\frac{|\widehat{\mathcal{E}}_i|}{|\mathcal{V}|}$. □

Furthermore, we present Theorem 8 to state the relationships between the LP performance under each evaluation metric and the node degree:

Theorem 8. *Given that* $|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|$ *follows hyper-geometric distribution, we have:*

$$E(\text{R@}K_i|d) = \frac{K}{N}, \qquad \frac{\partial E(\text{R@}K|d)}{\partial d} = 0, \tag{9.8}$$

$$E(\text{P@}K|d_i) = \frac{\alpha d}{N}, \qquad \frac{\partial E(\text{P@}K|d)}{\partial d} = \frac{\alpha}{N}, \tag{9.9}$$

$$E(\text{F1@}K|d) = \frac{2K}{N}\frac{\alpha d}{K+\alpha d}, \qquad \frac{\partial E(\text{F1@}K|d)}{\partial d} = \frac{2\alpha K^2}{N}\frac{1}{(K+\alpha d)^2}, \tag{9.10}$$

$$E(\text{N@}K|d) = \frac{\alpha d}{N}, \qquad \frac{\partial E(\text{N@}K|d)}{\partial d} = \frac{\alpha}{N}. \tag{9.11}$$

Proof.

$$E(\text{R@}K_i|d) = E(\frac{|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|}{|\widehat{\mathcal{E}}_i|}) = \frac{E(|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|)}{|\widehat{\mathcal{E}}_i|} = \frac{\frac{|\widehat{\mathcal{E}}_i|}{|\mathcal{V}|}K}{|\widehat{\mathcal{E}}_i|} = \frac{K}{N} \tag{9.12}$$

$$E(\text{P@}K_i|d) = E(\frac{|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|}{K}) = \frac{E(|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|)}{K} = \frac{\frac{|\widehat{\mathcal{E}}_i|}{|\mathcal{V}|}K}{K} = \frac{\alpha d}{N} \tag{9.13}$$

$$E(\text{F1@}K_i|d) = E(\frac{2|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|}{K+|\widehat{\mathcal{E}}_i|}) = \frac{2E(|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|)}{K+\alpha d} = \frac{2K}{N}\frac{\alpha d}{K+\alpha d} \tag{9.14}$$

$$E(\text{N@}K_i|d) = E(\frac{\sum_{k=1}^{K}\frac{\mathbb{1}[v_{\phi^k}\in(\widetilde{\mathcal{E}}_i\cap\widehat{\mathcal{E}}_i)]}{\log_2(k+1)}}{\sum_{k=1}^{K}\log_2(k+1)}) = \frac{E(\sum_{k=1}^{K}\frac{\mathbb{1}[v_{\phi^k}\in(\widetilde{\mathcal{E}}_i\cap\widehat{\mathcal{E}}_i)]}{\log_2(k+1)})}{\sum_{k=1}^{K}\frac{1}{\log_2(k+1)}} \tag{9.15}$$

To calculate the numerator DCG, i.e., $E(\sum_{k=1}^{K}\frac{\mathbb{1}[v_{\phi^k}\in(\widetilde{\mathcal{E}}_i\cap\widehat{\mathcal{E}}_i)]}{\log_2(k+1)})$ in Eq. (9.15), we model the link prediction procedure as 1) randomly select K nodes from the whole node space $\mathcal{V}$; 2) calculate $|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|$, i.e., how many nodes among the selected nodes $\widetilde{\mathcal{E}}_i$ are in the ground-truth neighborhood list $\widehat{\mathcal{E}}_i$; 3) randomly select $|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|$ slots to position nodes in $\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i$ and calculate DCG. The above steps can be mathematically formulated as:

$$\sum_{i=0}^{K}\frac{C(N-\alpha d, K-i)C(\alpha d, i)}{C(N,K)}\sum_{j=1}^{C(K,i)}p(\mathbf{O}_j^{(K,i)})\sum_{k=1}^{K}\frac{\mathbb{1}[\mathbf{O}_{jk}^{(K,i)}=1]}{\log_2(k+1)}, \tag{9.16}$$

where $\mathbf{O}^{(K,i)} \in \{0,1\}^{C(K,i)\times K}$ represents all $C(K,i)$ possible positional indices of putting i nodes into

K candidate slots. Specifically $\mathbf{O}_j^{(K,i)} \in \{0,1\}^K$ indicates the j^{th} positional configuration of i nodes where $\mathbf{O}_{jk}^{(K,i)} = 1$ if an node is positioned at k^{th} slot and $\mathbf{O}_{jk}^{(K,i)} = 0$ otherwise. Since our link predictor has no bias in positioning nodes in the K slots by Lemma 2, we have $p(\mathbf{O}_j^{(K,i)}) = \frac{1}{C(K,i)}$ and Eq. (9.16) can be transformed as:

$$\sum_{i=0}^{K} \frac{C(N-\alpha d, K-i)C(\alpha d, i)}{C(N,K)} \frac{1}{C(K,i)} \sum_{j=1}^{C(K,i)} \sum_{k=1}^{K} \frac{\mathbb{1}[\mathbf{O}_{jk}^{(K,i)} = 1]}{\log_2(k+1)}. \tag{9.17}$$

We know that only when the k^{th} slot is positioned a node can we have $\mathbf{O}_{jk}^{(K,i)} = 1$ and among the total $C(K,i)$ selections, every candidate slot $k \in \{1, 2, ..., K\}$ would be selected $C(K-1, i-1)$ times to position a node, which hence leads to:

$$\sum_{j=1}^{C(K,i)} \sum_{k=1}^{K} \frac{\mathbb{1}[\mathbf{O}_{jk}^{(K,i)} = 1]}{\log_2(k+1)} = \sum_{k=1}^{K} \frac{C(K-1, i-1)}{\log_2(k+1)}. \tag{9.18}$$

We then substitute Eq. (9.18) into Eq. (9.17) as:

$$\begin{aligned} &\sum_{i=0}^{K} \frac{C(N-\alpha d, K-i)C(\alpha d, i)}{C(N,K)} \frac{1}{C(K,i)} \sum_{k=1}^{K} \frac{C(K-1, i-1)}{\log_2(k+1)} \\ &= \sum_{i=0}^{K} \frac{C(N-\alpha d, K-i)C(\alpha d, i)}{C(N,K)} \frac{C(K-1, i-1)}{C(K,i)} \sum_{k=1}^{K} \frac{1}{\log_2(k+1)}. \end{aligned} \tag{9.19}$$

Further substituting Eq. (9.19) into Eq. (9.15), we finally get:

$$\begin{aligned} E(\mathrm{N@}K|d_i) &= \sum_{i=0}^{K} \frac{C(N-\alpha d, K-i)C(\alpha d, i)}{C(N,K)} \frac{C(K-1, i-1)}{C(K,i)} \\ &= \sum_{i=0}^{K} \frac{C(N-\alpha d, K-i)C(\alpha d, i)}{C(N,K)} \frac{\frac{(K-1)!}{(i-1)!(K-i)!}}{\frac{K!}{i!(K-i)!}} \\ &= \frac{1}{K} \underbrace{\sum_{i=0}^{K} i \frac{C(N-\alpha d, K-i)C(\alpha d, i)}{C(N,K)}}_{E(|\widetilde{\mathcal{E}}_i \cap \widehat{\mathcal{E}}_i|)} = \frac{1}{K}\frac{\alpha d}{N} * K = \frac{\alpha d}{N} \end{aligned} \tag{9.20}$$

□

Based on Theorem 8, Precision, F1, and NDCG increase as node degree increases even when no observed links are used to train the link predictor, which informs the degree-related evaluation bias and causes the illusion that high-degree nodes are more advantageous than low-degree ones observed in some previous works [68, 76].

CHAPTER 10

Bias Issue: Overcoming the Hallucination Bias Issue in Documental Question-answering

The revolutionary success of large language models (LLMs) in numerous real-world applications is often overshadowed by their propensity for generating inaccurate or nonsensical content. This issue, known as hallucination bias, critically undermines its utility in scenarios requiring high-stakes decision-making. To address this, we propose a knowledge graph prompting (KGP) method designed to ensure the generation of accurate content by LLMs, particularly in multi-document question-answering (MD-QA) tasks. Our approach comprises two main components: a graph construction module and a graph traversal module. For graph construction, we create a knowledge graph (KG) over multiple documents with nodes symbolizing passages or document structures (e.g., pages/tables) and edges denoting the semantic/lexical similarity between passages or intra-document structural relations. For graph traversal, we design an LM-guided graph traverser that navigates across nodes and gathers supporting passages assisting LLMs in MD-QA. The constructed graph serves as the global ruler that regulates the transitional space among passages and reduces retrieval latency. Concurrently, the LM-guided traverser acts as a local navigator that gathers pertinent context to progressively approach the question and guarantee retrieval quality. Extensive experiments underscore the efficacy of KGP for MD-QA, signifying the potential of leveraging graphs in enhancing the prompt design for LLMs[1].

10.1 Introduction

Due to the emergence of large language models (LLMs), the "pre-train, prompt, predict" paradigm has revolutionized natural language processing (NLP) in real-world applications, such as open-domain question answering (O-QA), fact-checking (FC), and arithmetic reasoning (AR) [235, 236, 237, 238, 239, 240]. However, no significant efforts have investigated this framework in the scenario of multi-documental question answering (MD-QA), which enjoys practical usage in academic research, customer support, and financial/legal inquiries that require analysis/insights derived from multiple documents [241, 242].

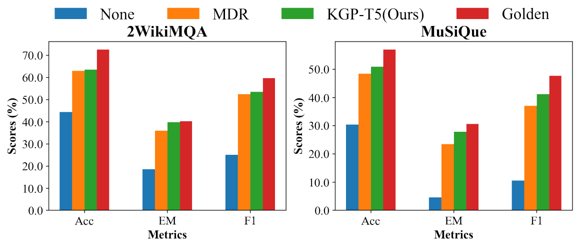


Figure 10.1: MD-QA performance when prompting ChatGPT with the context retrieved using different strategies.

To investigate the capability of LLMs for MD-QA, we randomly sample multi-document questions from the development set of 2WikiMQA [243] and MuSiQue [244], and then prompt LLMs in four different

[1]https://ojs.aaai.org/index.php/AAAI/article/view/29889

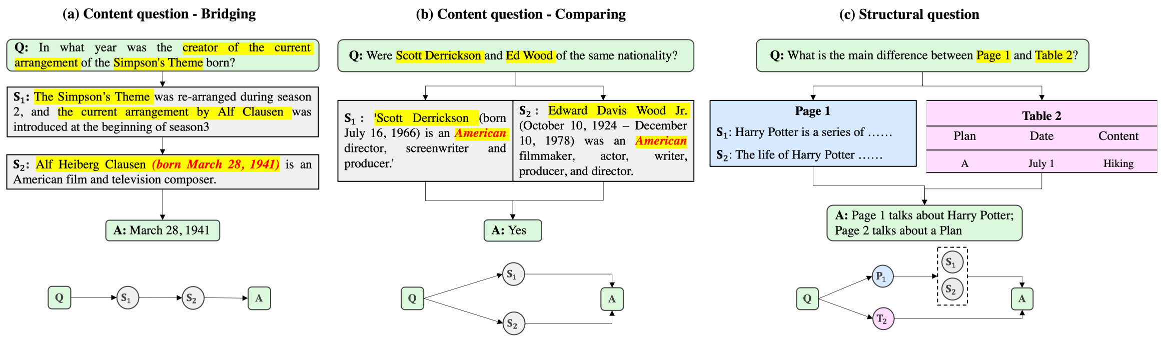

(a) Content question - Bridging
Q: In what year was the creator of the current arrangement of the Simpson's Theme born?
S1: The Simpson's Theme was re-arranged during season 2, and the current arrangement by Alf Clausen was introduced at the beginning of season3
S2: Alf Heiberg Clausen (born March 28, 1941) is an American film and television composer.
A: March 28, 1941
(b) Content question - Comparing
Q: Were Scott Derrickson and Ed Wood of the same nationality?
S1: 'Scott Derrickson (born July 16, 1966) is an American director, screenwriter and producer.'
S2: Edward Davis Wood Jr. (October 10, 1924 – December 10, 1978) was an American filmmaker, actor, writer, producer, and director.
A: Yes
(c) Structural question
Q: What is the main difference between Page 1 and Table 2?
Page 1
S1: Harry Potter is a series of ……
S2: The life of Harry Potter ……
Table 2
Plan Date Content
A July 1 Hiking
A: Page 1 talks about Harry Potter; Page 2 talks about a Plan
Q
S1
S2
P1
T2
A

Figure 10.2: Three popular questions that require reasoning and retrieving over passages/pages/tables from multiple documents. **(a) Bridging questions** rely on sequential reasoning while **(b) Comparing questions** rely on parallel reasoning over different passages. **(c) Structural questions** rely on fetching contents in the corresponding document structures.

strategies for the answer[2]. Successfully answering these questions requires knowledge of multiple Wikipedia documents. As shown in Figure 10.1, on 2WikiMQA and MuSiQue, directly prompting LLMs without providing any context, i.e., None, achieves only 25.07%/10.58% F1 and 18.60%/4.60% EM on 2WikiMQA and MuSiQue, which is far less than 59.69%/47.75% F1 and 40.20%/30.60% EM when prompting with supporting facts[3] provided as contexts, i.e., the Golden one. This demonstrates the limitation of fulfilling MD-QA using solely the knowledge encoded in LLMs. One standard solution to overcome this limitation in conventional O-QA and single document question-answering (D-QA) [245, 246] is to retrieve grounding contexts and derive faithful answers from the contexts, i.e., retrieve-and-read [247, 248]. However, unlike O-QA and D-QA, the primary challenge of MD-QA roots in its demands for alternatively retrieving and reasoning knowledge across different documents [249, 250]. For example, successfully answering questions in Figure 10.2(a)-(b) requires reasoning over distinct passages from two different documents (in these two cases, Wikipedia pages). Moreover, each document is essentially a compilation of multi-modality structured data (e.g., pages, sections, paragraphs, tables, and figures) and some questions may specifically ask for the content in certain structures, which necessitates a comprehensive grasp of these complex document structures. For example, the question in Figure 10.2(c) asks about the difference between Page 1 and Table 2, which is unanswerable if leveraging heuristic methods like BM25 or deep-learning ones like DPR [237]. Building on previous challenges, the advent of LLMs introduces new complexities.

For the challenge of alternatively retrieving and reasoning knowledge across different documents, although previous works train a multi-hop retriever [251, 252] to imitate such process by sequentially fetching the following passage based on the already-retrieved ones, none of them explore the potential of engaging LLMs into this process. More recent works design different prompting strategies such as Chain/Tree/Graph-of-thought [253, 254, 255, 256] to guide LLMs approaching answers progressively. However, prompting

[2]Detailed experimental setting is presented in Section 10.5.
[3]Supporting facts: passages that are assumed to contain the answer to the question.

non-open-sourced LLMs back and forth incurs forbiddable latency as well as unaffordable consumption. In addition, how to integrate different document structures into the prompt design so that LLMs can understand them is still an open-ended question.

In view of the above challenges, we propose a knowledge graph prompting (KGP) method for enhancing LLMs in MD-QA. Specifically, we construct a knowledge graph (KG) over the given documents with nodes symbolizing passages or document structures and edges denoting their lexical/semantic similarity between passages or intra-document structural relations. Then, for the first challenge of alternative retrieving and reasoning knowledge across different documents, we address it by alternatively prompting LMs to generate the subsequent evidence to approach the question, i.e., reasoning and selecting the most promising neighbor to visit next from the constructed KG based on the generated evidence, i.e., retrieval. Moreover, we apply the instruction fine-tuning strategy to augment the reasoning capability of our LMs and hence refrain from repeatedly prompting non-open-sourced LLMs for evidence generation. For the multi-modality challenge, we add different types of nodes to the KG, characterizing different document structures and hence enabling content retrieval within those specific structures. We highlight our contributions as follows:

- **Generally-applicable KG Construction.** We propose three KG construction methods over documents, with passages or document structures as nodes and their lexical/semantical similarity or structural relations as edges. Then, we empirically evaluate the quality of the constructed KGs in MD-QA by checking the level of overlap between the neighborhood and the supporting facts for each question (Figure 10.4).

- **Engaging KG for Prompt Formulation.** We design a Knowledge Graph Prompting (KGP) method, which retrieves the question-relevant contexts by traversing the constructed KG. Meanwhile, we fine-tune LMs that guide the graph traverser to adaptively navigate the most promising neighbors for approaching the question based on the already-visited nodes (retrieved passages).

- **Case Studies Verifying MD-QA Framework.** We provide insightful analysis, including comparing the quality of the constructed KGs in MD-QA and the performance of using different LMs to guide the graph traversal.

10.2 Related Work

Question answering Question Answering (QA) aims to provide answers to users' questions in natural language [248, 257], and most QA systems are composed of information retrieval (IR) and answer extraction (AE) [258, 247]. In IR, the system searches for query-relevant factual passages using heuristic methods (BM25) [259] or neural-ranking ones (DPR) [237]. The final answer is usually extracted as a textual span from related passages in AE. Although this framework has been broadly applied in O-QA [258, 260] and

D-QA [246, 245], no previous work focus on MD-QA, which demands alternatively reasoning and retrieving knowledge from multiple documents. To tackle this issue, we construct the KG to encode the logical associations among different passages across multiple documents and design an LM-guided traverser to generate the reason alternatively and visit the most matching passage node.

Multi-Document Question-Answering We are bombarded with large volumes of information, and studying Multi-Document Question-Answering (MD-QA) enhances our efficiency in digesting this information and taking efficacy action. For example, layers/doctors make decisions based on multiple reports, and companies gain insights into customer satisfaction from various feedbacks. While numerous works focus on question-answering, few [250, 261, 262, 263] explore its applications in multi-document scenarios. We briefly discuss their difference from ours. [250] devises multi-document pre-training objectives by predicting the salient sentence, while ours tackle MD-QA by constructing KGs and performing LM-guided graph traversal over documents. [261] uses LLM to extract document structures and answer questions, while ours models document structures by KG and extracts related contents via LM-guided traversal. Moreover, [250, 262, 263] do not consider the structure-based questions and do not follow the pretrain-prompt-predict framework of LLMs, while only [261] and ours consider structural-based questions and improve the prompt design for LLMs.

Pre-train, Prompt, and Predict with LLMs With the emergence of LLMs, the 'pre-train, prompt, predict' paradigm has gained significant popularity in handling a broad spectrum of tasks [264, 265, 266]. This approach begins with pre-training LLMs by pretext tasks to encode world knowledge into model parameters [267] followed by a prompting function to extract pertinent knowledge for downstream tasks [268]. Recent advancements explore different prompting strategies to enhance LLMs' reasoning capabilities [254, 255]. In contrast, our work offers a novel perspective by transforming the prompt formulation into the KG traversal.

10.3 Knowledge Graph Construction

Following [147], let $G = (\mathcal{V}, \mathcal{E})$ be a knowledge graph constructed from a set of documents $\mathcal{D}$, where the node set $\mathcal{V} = \{v_i\}_{i=1}^{n}$ representing document structures (e.g., passages/pages/tables, etc.) and the edge set $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ representing the connections among different nodes (e.g., semantic/lexical similarity and belonging relations among document structures, etc.). Let $\mathcal{X} = \{\mathcal{X}_i\}_{i}^{n}$ be node features and $\mathcal{X}_i$ corresponds to the feature of node v_i, the form of which could be the text for the passage, the markdown for the table and the page number for the page.

Despite numerous well-established KGs [269, 270], they treat nodes/edges as entities/relations, which necessitates sophisticated relational extraction techniques and thereby limits their applicability in general domains [271]. Additionally, their primary focus on the Wikipedia domain also restricts their usage for

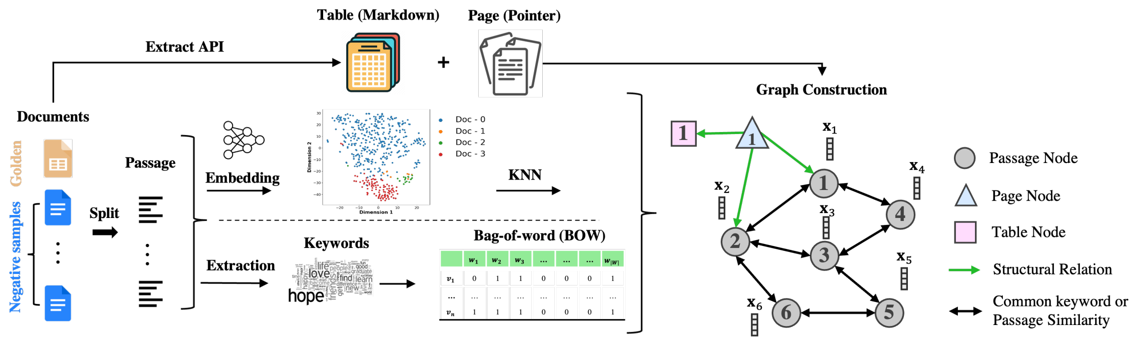


Figure 10.3: Knowledge Graph Construction. We split each document in the document collection into passages. For each passage, we either directly obtain their embeddings via pre-trained encoders or extract their keywords to build bag-of-word (BOW) features. Then, we connect two passages based on their embedding similarity or whether they share common keywords. Additionally, we extract tables/pages via Extract-PDF API and add them as structural nodes to the KG. If pages include passages and tables, we add a directed edge to denote the belonging relations. The table nodes include the markdown formatted content of that table.

answering non-Wikipedia questions such as ones over legal or financial documents. To remedy this issue, we propose three generally applicable KG construction methods.

We first analyze two representative questions in Figure 10.2(a)-(b) to motivate our KG construction. Answering these two questions necessitates the deduction of logical associations among different passages. These associations are encoded either through 1) lexical similarity: common keywords shared among different passages, e.g., ‘Alf Clausen’ bridges passage $\mathbf{S}_1$ and passage $\mathbf{S}_2$ in Figure 10.2(a), or 2) semantic similarity: syntactic elements that convey semantic relations, e.g., “nationality” and “American director” in Figure 10.2(b). This motivates us to construct the graph by modeling passages as nodes and their lexical/semantic similarity as edges. More specifically, in Figure 10.3, we split each document into individual passages, and for each passage $\mathbf{S}_i$, we add a node v_i to the KG with its feature being the text of that passage $\mathcal{X}_i$. Then we add edges by checking the lexical/semantic similarity between pairs of passage nodes.

TF-IDF KG Construction. For adding edges according to lexical similarity, we first apply TF-IDF keyword extraction [272] over each document to filter out meaningless words such as supporting verbs and articles, which reduces the dimension of BOW features, sparsifies the constructed graph and increases the efficiency of the graph traversal. In addition, we add the document title into the extracted keyword set since some questions focus on title entities. We collect the extracted keywords from all documents to form the keyword space $\mathcal{W}$ and then connect two passages if they share any common keyword in $\mathcal{W}$.

KNN-ST/MDR KG Construction. For adding edges according to semantic similarity, we can readily employ pre-existing models such as sentence transformers to generate passage embedding $\mathbf{X}_i$ for each node v_i and subsequently compute pairwise similarity matrix to construct the K-nearest neighbor (KNN) graph. However, these off-the-shelf models, typically trained on tasks not so-related to MD-QA, may not adequately encapsulate necessary logical associations in their embedding similarity demanded by the question. To overcome this problem, we follow the training strategy of MDR [251] and train a sentence encoder by predicting the subsequent supporting facts based on previously supporting facts, thereby endowing the encoder with reasoning capability. Consequently, the embedding similarity and the corresponding constructed KNN graph fundamentally encapsulate the necessary logical associations between different passages.

TAGME. Moreover, we employ TAGME [273] to extract Wikipedia entities from each passage and construct the graph based on whether two passage nodes share common Wikipedia entities.

In addition to passage nodes, we further add structural nodes into the graph by extracting document structures via Extract-PDF [4]. In this chapter, we only consider adding pages and tables but the constructed KG can include more different types of document structures.

To verify the constructed KGs indeed encode the necessary information for MD-QA, we randomly sample questions from HotpotQA and create KGs over the set of documents for each of these questions using our proposed methods. We vary the hyperparameters to control the sparsity of the constructed graph and measure how much of the percentage of the supporting facts are covered by neighbors of the seeding passages initialized by TF-IDF. As shown in Figure 10.4, as the constructed graph becomes denser, the chance that the neighboring node passages hit the supporting facts increases (i.e., SF-EM increases) although the redundant information also increases (i.e., the precision decreases). Given the common keywords shared between one passage and all other passages are typically far less than the total number of passages across all documents, the density of the constructed graph by TF-IDF would be upper-bounded, causing lower SF-EM (evidenced by SF-EM below 0.7 in Figure 10.4 for TF-IDF curve). For TAGME, we empirically find it identifies a larger quantity of entities mentioned in a single passage, which leads to a denser graph and causes the starting SF-EM of TAGME to be already around 0.95. In addition, since KNN-MDR is pre-trained by predicting the next supporting facts [251] on HotpotQA, it achieves better trade-off than KNN-ST where the embeddings are directly obtained from the sentence transformer without dataset-specific pre-training.

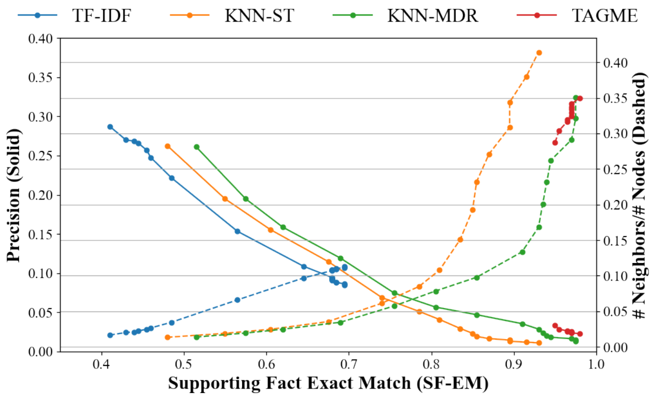
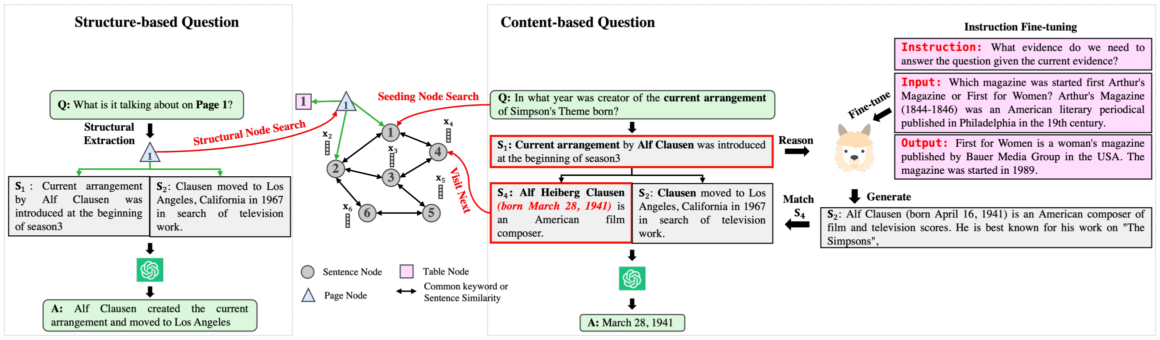


Figure 10.4: Quality of KGs on HotpotQA. For each KG Construction method, as the average number of neighbors increases (KG becomes denser) in the right y-axis, the SF-EM increases while the precision decreases. KNN-MDR achieves a better trade-off than TF-IDF and KNN-ST. KGs constructed by TAGME are denser than others.

[4]https://developer.adobe.com/document-services/docs/overview/pdf-extract-api/



Structure-based Question
Q: What is it talking about on Page 1?
Structural Extraction
S1 : Current arrangement by Alf Clausen was introduced at the beginning of season3
S2: Clausen moved to Los Angeles, California in 1967 in search of television work.
A: Alf Clausen created the current arrangement and moved to Los Angeles
Structural Node Search
Seeding Node Search
Visit Next
x2
x3
x4
x5
x6
Sentence Node
Table Node
Page Node
Common keyword or Sentence Similarity
Content-based Question
Q: In what year was creator of the current arrangement of Simpson's Theme born?
S1: Current arrangement by Alf Clausen was introduced at the beginning of season3
S4: Alf Heiberg Clausen (born March 28, 1941) is an American film composer.
S2: Clausen moved to Los Angeles, California in 1967 in search of television work.
A: March 28, 1941
Reason
Match S4
Instruction Fine-tuning
Instruction: What evidence do we need to answer the question given the current evidence?
Input: Which magazine was started first Arthur's Magazine or First for Women? Arthur's Magazine (1844-1846) was an American literary periodical published in Philadelphia in the 19th century.
Output: First for Women is a woman's magazine published by Bauer Media Group in the USA. The magazine was started in 1989.
Fine-tune
Generate
S2: Alf Clausen (born April 16, 1941) is an American composer of film and television scores. He is best known for his work on "The Simpsons",

Figure 10.5: LM-guided graph traverser for context retrieval. For questions on document structures (left), we employ LM to extract structures and retrieve their corresponding contents (the content of pages are passages belonging to that page, and the content of tables is the markdown-formatted text). For questions on document content, we concatenate it with the currently retrieved context and prompt the LM to generate the next evidence to answer the question. By comparing the similarity between the candidate neighboring sentences and the generated passage, we determine the next passage node to traverse. Correspondingly, the candidate neighbors are updated for the next round of traversal.

To summarize, although high SF-EM indicates that the neighbors of seeding passages fully cover the supporting facts for most questions, low precision signifies that most of these neighboring passages are irrelevant to the question. Therefore, if we blindly perform graph traversal without any question-tailored adaptation, our retrieved contexts would include redundant passages and compromise the capability of LLMs in MD-QA (which is also verified by the low performance of KGP w/o LM in Table 10.3). To remedy this issue, in the next section, we introduce an LM-guided graph traverser to adaptively visit neighboring passages most conducive to answering the given question.

10.4 LM-guided Graph Traverser

A natural solution to enable adaptive graph traversal is to rank the candidate nodes, i.e., the neighbors of the already-visited nodes in our case, thereby determining which ones to visit next. The most straightforward way is to apply heuristic-based fuzzy matching or embedding-based similarity ranking, which cannot capture the intrinsic logic relations between the already traversed paths and the nodes to visit. Instead, we fine-tune a language model (LM) to guide the graph traversal toward the next most promising passages in approaching the question based on the visited passages.

Given a question q asking about the document content, the LM-guided graph traverser reasons over previously visited nodes/retrieved passages $\{s_k\}_{k=0}^{j}$ and then generates the next passage s_{j+1} as follows:

$$s_{j+1} = \operatorname*{arg\,max}_{v \in \mathcal{N}_j} \phi(g(\mathcal{X}_v), f(||_{k=0}^{j}\mathcal{X}_k)), \tag{10.1}$$

where $||_{k=0}^{j}\mathcal{X}_k$ concatenates the textual information of previously retrieved passages/visited nodes. For the choice of f, one way is to employ encoder-only models like Roberta-base [236, 251, 252] and correspondingly g would be another encoder model with $\phi(\cdot)$ being the inner product measuring the embedding similarity. Another way is to employ encoder-decoder models such as T5 [274, 275] and correspondingly g would be an identity function with $\phi(\cdot)$ measuring the textual similarity. To mitigate the hallucination issue [276] and enhance the reasoning capability [254] of LMs, we further apply instruction fine-tuning to f [277] by predicting the next supporting facts based on previous supporting facts, thereby integrating commonsense knowledge encoded originally in their pre-trained parameters with the enhanced reasoning capability inherited from the instruction fine-tuning. After visiting the top-scoring nodes selected from the candidate neighbor queue by Eq (10.1), the candidate neighbor queue is updated by adding neighbors of these newly visited nodes. We iteratively apply this process until we hit the preset budget. Next, we illustrate the above process with an example in Figure 10.5 but leave the comprehensive traversal algorithm in Algorithm 3 in Appendix.

In Figure 10.5, the content-based question asks, 'In what year was the creator of the current arrangement of Simpson's Theme born?'. We use TF-IDF search to initialize our seeding passage Node 1, which reads: "Alf Heiberg Clausen (born March 28, 1941) is an American film composer". Subsequently, we prefix the currently retrieved-context (Node 1) with the question and prompt the LM to generate the subsequent evidence required to approach the question more closely. Because we augment the reasoning capability of the LM by instruction fine-tuning, it is expected to recognize the logical associations between the question and the currently retrieved context. Consequently, it can predict the subsequent passage that *maintains logical coherence, albeit may contain factual mistakes*, i.e., "Alf Clausen (born April 16, 1941) is an American composer of film and television scores." To rectify this potential factual mistake, we select nodes from the candidate neighbors that match the most with the LM-generated passage; in this case, Node 4 "Alf Heiberg Clausen (born March 28, 1941) is an American film composer". Since this passage is sourced directly from documents, it inherently ensures the validity of the information. Then, we prompt LLMs along with the retrieved context Node 1 and 4 for the answer.

Additionally, we extracted the document structure names for questions about document structures and located their corresponding structural nodes in the KG. For the table node, we retrieve its markdown formatted content; for the page node, we traverse its one-hop neighbor and obtain passages belonging to that page.

10.5 Experiment

In this section, we conduct experiments to verify the proposed knowledge graph prompting method (KGP) for MD-QA. In particular, we answer the following questions:

- **Q1 - Section 10.5.2**: How well does KGP perform MD-QA compared with existing baselines?
- **Q2 - Section 10.5.3-10.5.4**: How do the quality of the constructed KG and the LM-guided graph traverser impact the MD-QA performance?

10.5.1 Experimental Setting

Datasets. To explore the uncharted domain of MD-QA, we have created our own datasets to simulate real-world scenarios where users maintain folders containing various documents and pose questions, the answers to which are only from certain parts of these documents. Specifically, we randomly sample questions from the development set of four existing datasets: HotpotQA [278], IIRC [279], 2WikiMQA [243], and MuSiQue [244]. For each question, we source documents from Wikipedia that encompass supporting facts pertaining to the question and combine them with randomly sampled negative documents to form the document collection. In addition to the content-based questions from these four existing datasets, we additionally incorporate the 'Comp' dataset, an internal company collection of real-world document-based questions. During its creation, humans were asked to read documents and pose questions according to document structures. We summarize the statistics of each dataset along with their KGs in Table 10.1.

Table 10.1: Statistics of documents and their KGs constructed by TAGME average across all questions.

Dataset	**# Documents**	**# Questions**	**# Passages**	**# Edges**	**Passage Avg. Length**	**KG Density**
HotpotQA	12	500	715.22	70420.68	37.55	0.23
IIRC	12	477	1120.55	143136.17	37.24	0.20
WikiMHop	12	500	294.19	19235.15	37.24	0.27
MuSiQue	12	500	748.04	97931.28	38.56	0.29

Baselines. We compare KGP with retrieval baselines in three categories. The first category is the heuristic-based retriever including KNN with fuzzy search, TF-IDF [272], and BM25 [259]. The second category is the deep-learning-based retriever including DPR [237] and MDR [251]. The third category is the prompting-based retriever including IRCoT [253]. For KGP, we explore three variants based on their LM-guided graph traverser: KGP-T5, KGP-LLaMA, and KGP-MDR, using T5 (encoder-decoder), LLaMA (decoder only), and MDR (encoder only) respectively as f in Eq (10.1).

Evaluation Criteria. Following [280], we compute F1 and EM to compare the LLM's answer and the ground-truth one. As the predicted answer may not overlap with the ground-truth one, we additionally check the correctness of the answer following [281, 282, 283] by prompting the LLM. Moreover, for evaluating the quality of KGs in Figure 10.4, we adopt SF-EM (Supporting Fact Exact Matching) and precision from [251]. Given the subjective nature of the questions in Comp, we devise the metric Structure Exact Matching (Struct-EM) to assess if retrieved contexts include the document structures mentioned in the question.

Table 10.2: MD-QA Performance (%) of different baselines. The best and runner-up are in **bold** and underlined. None: no passages, but only the question is provided. Golden: supporting facts are provided along with the question. Therefore, None and Golden routine the lower/upper bound of MD-QA.

Method	**HotpotQA**			**IIRC**			**2WikiMQA**			**MuSiQue**			**Comp**
	Acc	EM	F1	Acc	EM	F1	Acc	EM	F1	Acc	EM	F1	Struct-EM
None	41.80	19.00	30.50	19.50	8.60	13.17	44.40	18.60	25.07	30.40	4.60	10.58	0.00
KNN	71.57	40.73	57.97	43.82	25.15	37.24	52.40	31.20	42.13	44.70	18.86	30.04	–
TF-IDF	**76.64**	45.97	64.64	47.47	27.22	40.80	58.40	34.60	44.50	44.40	21.59	32.50	–
BM25	71.95	41.46	59.73	41.93	23.48	35.55	55.80	30.80	40.55	44.47	21.11	31.15	–
DPR	73.43	43.61	62.11	48.11	26.89	41.85	62.40	35.60	51.10	44.27	20.32	31.64	–
MDR	75.30	45.55	65.16	**50.84**	27.52	**43.47**	63.00	36.00	52.44	48.39	23.49	37.03	–
IRCoT	74.36	45.29	64.12	49.78	27.73	41.65	61.81	37.75	50.17	45.14	22.46	34.21	–
KGP-T5	76.53	**46.51**	**66.77**	48.28	26.94	41.54	**63.50**	**39.80**	**53.50**	**50.92**	**27.90**	**41.19**	
KGP-LM	75.66	46.22	66.31	49.57	28.09	42.56	62.45	37.55	52.45	50.81	26.72	40.01	**67.00**
KGP-MDR	75.72	46.09	65.77	49.58	**29.32**	43.21	60.94	37.22	51.29	**51.22**	27.76	41.11	
Golden	82.19	50.20	71.06	62.68	35.64	54.76	72.60	40.20	59.69	57.00	30.60	47.75	100.00

10.5.2 Performance Comparison on MD-QA

We compare the MD-QA performance of the proposed KGP-T5 and other baselines in Table 10.2. Firstly, the baseline "None" and "Golden" achieve the worst and the best performance because one provides no context and the other provides the golden context. All other baselines achieve the performance in-between because the retrieved context only covers the partial of the supporting facts. Our proposed methods KGP-T5 rank at the Top-1 except for the Golden baseline. The 2$^{\text{nd}}$-performing baseline MDR fine-tunes a RoBERTa-base encoder by predicting the next supporting fact based on the question and the already retrieved contexts [251]. This next-passage prediction pretext task equips the model with the reasoning capability of the knowledge across different passages and hence increases the quality of the retrieved contexts. The other deep-learning-based retriever DPR achieves much worse performance than MDR because it only fine-tunes the encoder by maximizing the similarity between the query and its supporting facts regardless of their sequential order, demonstrating the importance of understanding the logical order of different knowledge when solving MD-QA [251]. By comparing the MD-QA performance across different datasets, we find that all baselines perform better on HotpotQA than on IIRC. This is because questions in HotpotQA are generally simpler than in IIRC. Existing works [284] have shown that some questions can be easily answered by following shortcuts, while questions in IIRC sometimes necessitate arithmetic skills to derive the numerical answers, e.g., 'How many years did the event last when Wingfield lost much of his fortune?'.

Moreover, without any particular design for document structures, no existing baselines can handle structural questions in Comp, e.g., "What is the difference between Page 1 and Page 2?" or "In Table 3, which station has the highest average flow rate?". Fortunately, with the constructed KG incorporating the structural nodes and our designed traversal algorithm retrieving structural contexts, our proposed method achieves 67% Struct-EM.

10.5.3 Impact of the Constructed Graph

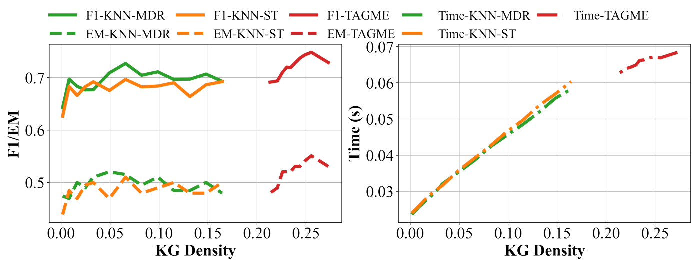


Figure 10.6: The performance/latency increases as the KG density increases. The results are averaged across 100 randomly sampled questions on HotpotQA.

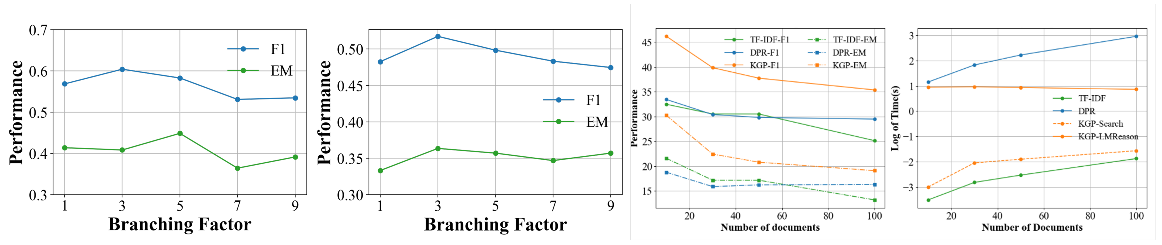
Here, we construct KGs with varying densities by changing the hyperparameters of TF-IDF/KNN-ST/KNN-MDR/TAGME and studying its impact on the performance and the neighbor matching time of MD-QA using KGP-T5. Since the LM-guided graph traverser selects the next node to visit from neighbors of already visited nodes, the chance that it hits the supporting facts increases as the number of neighbors increases. In contrast, the neighborhood matching efficiency decreases as the candidate pool, i.e., $\mathcal{N}_j$ in Eq (10.1), becomes larger. As evidenced in Figure 10.6, we observe a similar trend, i.e., as the KG density increases, the F1/EM increases and then stays stable while the latency for selecting the most promising neighbors to visit next also increases. KNN-MDR performs better than KNN-ST when the density of the two constructed KGs is the same. This is because the encoder in KNN-ST is pre-trained on wide-spectrum datasets. In contrast, the encoder in MDR is specifically pre-trained on the HotpotQA dataset by the pretext task of predicting the following supporting facts. Therefore, the embedding similarity and the corresponding neighbor relations better reflect the logical associations among different passages, which aligns with the better constructed KG by KNN-MDR than the KG by KNN-ST in Figure 10.4. Compared with KNN-MDR/ST, TAGME delivers superior performance at the cost of increasing latency since the KG generated by TAGME is denser than the KGs generated by KNN-ST/MDR.

Table 10.3: Comparing ChatGPT equipped with few-shot demonstration with Fine-tuned LLaMA/T5

Dataset		**Hotpot-QA**			**IIRC**			**2WikiMQA**			**MuSiQue**		
Metric		Acc	EM	F1	Acc	EM	F1	Acc	EM	F1	Acc	EM	F1
TAGME	w/o LM	73.52	43.79	63.14	46.30	27.70	41.43	58.12	35.07	45.95	44.67	21.93	32.90
	ChatGPT	**77.80**	46.03	66.57	46.27	26.01	39.35	61.62	36.16	49.39	50.61	26.92	38.66
	LLaMA	75.66	46.22	66.31	**49.57**	**28.09**	**42.56**	62.45	37.55	52.45	50.81	26.72	40.01
	T5	76.53	**46.51**	**66.77**	48.28	26.94	41.54	**63.50**	**39.80**	**53.50**	**50.92**	**27.90**	**41.19**

10.5.4 Impact of the LM-guided Graph Traverser

Here, we study the influence of using different LMs in guiding graph traversers over TAGME-constructed KG on MD-QA performance. Specifically, we compare the guidance by no LM (w/o LM), LLaMA, T5, and MDR in Table 10.3. Because TAGME w/o LM only blindly traverses in the KG without any guidance from LM, it unavoidably collects irrelevant passages and hence achieves the worst performance than others with LM guidance. This aligns with our previous observation on the generally low precision in Figure 10.4



(a) 2WikiMQA (b) MuSiQue (c) Performance (d) Efficiency

Figure 10.7: **(a)-(b)**: The performance first increases and then decreases as the branching factor increases. The results are averaged across 100 sampled questions on 2WikiMQA and MuSiQue. **(c)-(d)**: KGP achieves higher performance/efficiency than DPR when QA over different numbers of documents.

and further demonstrates the necessity of using LMs to guide the graph traversal. Interestingly, we find that KGP-T5 performs better than LLaMA even though the parameters of LLaMA (7B) are more than the ones with T5 (0.7B). We will investigate this in future work.

10.5.5 Sensitivity Analysis

Here, we perform the sensitivity analysis of the branching factor (the number of nodes selected from candidate neighbors to visit next). In Figure 10.7(a)-(b), the performance first increases as the branching factor increases because more passage nodes selected from the candidate neighbors lead to more reasoning paths to reach the final answer. However, as we fix the context budget to ensure fair comparison (i.e., the total number of passages we are allowed to retrieve for each question is the same across all baselines), the performance declines as the branching factor increases because the number of initial seeding nodes diminishes, leading to reduced coverage of the KG.

10.6 Conclusion

Answering multi-document questions demands knowledge reasoning and retrieving from different documents across various modalities, presenting challenges for applying the 'pre-train, prompt and predict' paradigm with LLMs. Recognizing that the logical associations among passages and structural relations within the documents can be unified into a graphical representation, we propose a Knowledge Graph Prompting method (KGP) for aiding LLMs in MD-QA. The KGP constructs KGs from documents with nodes depicting sentences or document structures and edges denoting their lexical/semantic similarity or structural relations. Since the constructed KGs may contain irrelevant neighbor information, we further design an LM-guided graph traverser that selectively visits the most promising node in approaching the question. In the future, we plan to investigate the capability of LLMs to understand graph topology and explore the potential of fine-tuning/prompting LLMs to encode complex topological signals hidden in the graph.

10.7 Appendix

10.7.1 Dataset Collection

This section introduces the collection of datasets used for the experiments conducted in this chapter.

Document Set Collection and Procession. As no previous works focus on MD-QA, we create our own datasets to simulate real-world scenarios where users maintain folders containing various documents and pose questions, the answers to which are only from certain parts of these documents. To imitate this scenario, we randomly sample questions from the development set of existing datasets: HotpotQA/IIRC/2WikiMQA/MuSiQue, and then for each specific question, we fetch documents from Wikipedia that encompass supporting facts of the question [5] and term these documents as golden documents. Then, we randomly sample negative documents from Wikipedia and pair them with golden documents to constitute the document collection. For each document in the collected document set, we split it into multiple passages, with the default passage length being 250, as it empirically yields superior performance. As questions from these existing datasets are only focused on document contents, we additionally incorporate the 'Comp' dataset, an internal company collection of real-world questions focusing on document structures.

Knowledge Graph Construction. We construct a knowledge graph for each question and its corresponding collection of documents. For datasets where the questions are from Wikipedia: HotpotQA, IIRC, WikiMHop, and Musique, we only have passage nodes since answering questions in these datasets does not require information about document structures. In addition to passage nodes, we apply ExtractAPI for the Comp dataset to obtain the page and table information so that the constructed KG also has pages/tables as nodes. We add edges for all these datasets following Section 10.3. For Comp, due to privacy concerns, we omit the data statistics but only provide some question examples, e.g., 'How many more classical students in Table 2 had the mixed teaching style versus the classical teaching style?' or 'Can you give me a simple summary about page 5?'.

Sequential Data Collection. Training MDR [251] requires rearranging supporting facts into the sequential order that progressively approaches the answer. To fulfill this requirement, we directly follow MDR and use the pre-processed HotpotQA data from the GitHub Repository[6] to train the encoder and apply it to other datasets that do not provide the sequential order of supporting facts. For instruction fine-tuning LLaMA, we still use the above HotpotQA data and rearrange it into the instruction-input-output format and use the instruction 'What evidence do we need to answer the question given the current evidence'. For T5-large, we use the same input-output but prefix the reasoning instruction to the input following the original T5 input format [285].

[5]The HotpotQA/IIRC/2WikiMQA/Musique datasets already have the supporting facts for each question.

[6]https://github.com/facebookresearch/multihop_dense_retrieval/tree/main

Algorithm 3: Knowledge Graph Prompting Method for Questions on Document Contents

```
Input: A question $q$ over a set of documents $\mathcal{D}$, the constructed knowledge Graph $G = \{\mathcal{V}, \mathcal{E}, \mathcal{X}\}$ over $\mathcal{D}$, the
       fine-tuned LLM-guided graph traversal $f_{\text{GT}}$, the preset context budget $K$, the initial TF-IDF search
       function $g$.
1  Initialize seed passages $\mathcal{V}^s = g(\mathcal{V}, \mathcal{X}, q)$
2  Initialize the retrieved passage queue $\mathcal{P} = [\{v_i\}|v_i \in \mathcal{V}^s]$
3  Initialize the candidate neighbor queue $\mathcal{C} = [\mathcal{N}_i|v_i \in \mathcal{V}^s]$
4  Initialize the retrieved passage counter $k = \sum_{\mathcal{P}_i \in \mathcal{P}} |\mathcal{P}_i|$
5  while queue $\mathcal{P}$ and queue $\mathcal{C}$ are not empty do
6      $\mathcal{P}_i \leftarrow \mathcal{P}$.dequeue(), $\mathcal{C}_i \leftarrow \mathcal{C}$.dequeue()
7      $\mathcal{V}_i' =$ Graph Traversal($\{q\} \cup \mathcal{P}_i, \mathcal{C}_i, k$) by Eq (10.1)
8      for $v \in \mathcal{V}_i'$ do
9          $\mathcal{P}$.enqueue($\mathcal{P}_i \cup \{v\}$)
10         $\mathcal{C}$.enqueue($\mathcal{N}_v$)
11         $k \leftarrow k + 1$
12         if $k > K$ then
13             Terminate
14 return Retrieved Passage Queue $\mathcal{P}$
```

10.7.2 Algorithm for KGP

Here we present the algorithm for our proposed knowledge graph prompting (KGP) method for MD-QA. Given a question, we first apply LLM to classify whether the question is asking about the document structure or document content. If the question focuses on the document structure, we extract the structural keywords such as Page or Table, and retrieve the content in the corresponding structural nodes in KG. If the question focuses on the document content, we follow the step according to Algorithm 3. Specifically, we first initialize seeding passages $\mathcal{V}^s$ and the reasoning path queue $\mathcal{P}$ by TF-IDF search. Then for each seeding passage $v_i \in \mathcal{V}^s$, we add its neighboring passage nodes $\mathcal{N}_i$ into the candidate neighbor queue $\mathcal{C}$. (lines 1-4) After that, we iteratively pop out the leftmost reasoning path/candidate neighborhood $\mathcal{P}_i/\mathcal{C}_i$ from $\mathcal{P}/\mathcal{C}$ and employ the fine-tuned LM-guided graph traverser to rank the popped out neighbors in $\mathcal{C}_i$ by Eq. (10.1) (lines 5-7). Last, we select top-k passage nodes $\mathcal{V}_i'$ from $\mathcal{C}_i$ to visit next based on their rank and correspondingly update the candidate neighbor queue/reasoning path queue (lines 8-13). The above process terminates when either the candidate neighbor queue becomes empty, or the prefixed budget K for the retrieved passages is met.

CHAPTER 11

Limited Data Issue: Overcoming Limited Data Issue in Graph Generation

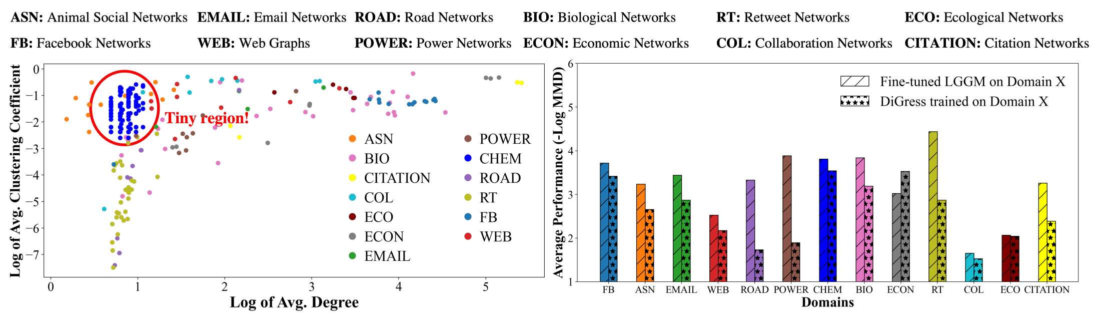
Large Generative Models (LGMs) such as GPT, Stable Diffusion, Sora, and Suno are trained on a huge amount of language corpus, images, videos, and audio that are extremely diverse from numerous domains. This training paradigm over diverse well-curated data lies at the heart of generating creative and sensible content. However, all previous graph generative models (e.g., GraphRNN, MDVAE, MoFlow, GDSS, and DiGress) have been trained only on one dataset each time, which cannot replicate the revolutionary success achieved by LGMs in other fields. To remedy this crucial gap, we propose a new class of graph generative model called LARGE GRAPH GENERATIVE MODEL (LGGM) that is trained on a large corpus of graphs (over 5000 graphs) from 13 different domains. We empirically demonstrate that the pre-trained LGGM has superior zero-shot generative capability to existing graph generative models. Furthermore, our pre-trained LGGM can be easily fine-tuned with graphs from target domains and demonstrate even better performance than those directly trained from scratch, behaving as a solid starting point for real-world customization. Inspired by Stable Diffusion, we further equip LGGM with the capability to generate graphs given text prompts (Text-to-Graph), such as the description of the network name and domain (i.e., ”The power-1138-bus graph represents a network of buses in a power distribution system.”), and network statistics (i.e., ”The graph has a low average degree, suitable for modeling social media interactions.”). This Text-to-Graph capability integrates the world knowledge in the underlying language model, offering users fine-grained control of the generated graphs.[1].

11.1 Introduction

Recently, Large Generative Models (LGMs) such as GPT, Stable Diffusion, Sora, and Suno [286, 287, 288, 275] have achieved revolutionary success in generating creative and sensible content, which significantly increases the productivity of real-world applications [289, 290, 291]. Unlike previous models such as Bert/Bart [292, 293] in Natural Language Processing (NLP) and Unet [294] in Image Segmentation that are trained only on small-scale datasets from specific domains over narrow tasks, the key to the success of these LGMs lies in their large training paradigm over the well-curated training data from a wide variety of domains [295, 292, 296, 275]. Graph, as a data modality distinct from image, text, and audio, is ubiquitous across numerous fields and presents a new frontier for applications of generative models such as drug discovery [297, 298], material design [299, 300] and cyber-security [301, 302]. Given the unprecedented success achieved by LGMs in other domains, we naturally ask:

Can we propose large generative models for graph-structured data?

[1]https://lggm-lg.github.io/



(a) Statistics of graphs in the whole graph universe

(b) Average performance across 4 evaluation metrics

Figure 11.1: (a): Average degree and clustering coefficient of graphs from 13 domains. The graph universe consists of graphs from distinct domains (e.g., the tiny region of Chemical Graphs), yet there are some common transferrable patterns. (b): Our pre-trained LGGM after fine-tuning on each domain achieves better generative performance than DiGress trained on that same domain.

Although graph generative models have been the long-standing focus of generative-based research [303, 304, 305], previous ones have been trained on graphs from only one domain each time. For example, both the representative auto-regressive-based GraphRNN [306], VAE-based GraphVAE [307] and diffusion-based DiGress [308] have been trained only on synthetic (Ego, Community, Grid) or chemistry graphs (Enzymes, QM9), the statistics of which only counts a tiny region of the whole graph universe and is different from graphs in other domains, as shown in Figure 11.1(a). **Road Networks** possess lower average clustering coefficients than **Facebook Networks (FB)**. This is because **Road Networks**, by design, have square intersections, whereas social relationships in **Facebook Networks (FB)** naturally form triangular connections [309]. Moreover, Tortoise **Animal Social Networks (ASN)**[2] have a lower average degree than **Power Networks** because tortoises, as solitary creatures, would not share the same burrow [311]. As a result, graph generative models trained on one domain are hardly generalizable to unseen graphs, as shown by the worse zero-shot generation performance of DiGress in Table 11.2. More critically, without training on numerous graphs covering the whole graph universe, these small models can never replicate the revolutionary success by LGMs in other fields.

Recognizing the significant gap in developing LGMs for graph-structured data and their potential revolutionary impact similar to LGMs in other fields, we develop the very first LARGE GRAPH GENERATIVE MODEL (LGGM) that is pre-trained over 5000 graphs from 13 domains sourcing from the Network Repository - the interactive data repository collecting graphs from 30 domains [312, 313]. After this pre-training, our LGGM learns some fundamental structural patterns that are transferrable across different domains [314] and henceforth demonstrates significantly better zero-shot generative capability on graphs from unseen domains shown in Table 11.2. Moreover, the pre-trained LGGM are highly adaptable for fine-tuning on a specific domain, achieving an overall performance increase of 29.59% compared to the smaller DiGress model trained

[2]Nodes represent tortoises and an edge is set up between two tortoises if they share the same burrow [310].

on the same domain, as depicted in Figure 11.1(b). This improvement is even more significant when only limited graphs are available shown by Figure 11.5, proving particular advantages in semi-supervised generative settings [299, 315, 316]. More importantly, our LGGMs support Text-to-Graph generation, which allows finer-level control of the generated graphs (e.g., their domains/names in Table 11.3 and clustering coefficient/average degree in Figure 11.4. **Our contributions are as follows:**

- **Large Graph Generative Model:** We propose a pioneering Large Graph Generative Model (LGGM), trained on thousands of graphs arising from 13 distinct domains. To the best of our knowledge, this work is the very first one exploring the potential of LGMs on graph-structured data. We hope others expand this collection and leverage our work to develop future LGGMs that could eventually replicate the success of Stable Diffusion [288] but in the graph modality.
- **Superior Zero-shot and Fine-tuning Generative Capability:** Our pre-trained LGGM delivers exceptional zero-shot generative performance on unseen graphs in Table 11.2 of Section 11.6.2 and shows great adaptability for fine-tuning in Figure 11.3 of Section 11.6.3. Remarkably, the fine-tuned LGGM outperforms DiGress trained from scratch on the same graphs especially under limited data scenarios, behaving as a better starting point for real-world development.
- **Text-to-Graph Generation:** We equip the LGGM with the capability to generate graphs given user-specified text prompts, allowing finer-level control of the generated graphs in terms of their domains/names and network statistics.

11.2 Related work

11.2.1 Large Generative Models (LGMs)

Recent years have witnessed unprecedented success achieved by LGMs in generating creative and sensible content for a variety of downstream tasks across multiple modalities [286, 287, 317, 275, 318]. For instance, in Natural Language Processing (NLP), large language models trained on next-token prediction can effectively produce human-readable texts for completing question-answering, translation, and more tasks [240, 319, 11]. Furthermore, the advancement of multi-modal generative models now supports cross-modality generation, such as converting text into images with Stable Diffusion or vice versa with GIT [288, 283, 320]. The key to their success lies in their ability to effectively utilize the world knowledge obtained during the pre-training stage over a large amount of well-curated data. This world knowledge has been demonstrated to be positively transferrable across numerous domains, delivering promising efficacy with few-shot task demonstrations. Compared with the recent large generative models in NLP/CV such as LLaMA3, Falcon, Stable Diffusion, LLaVA [321, 322, 288, 275], we alternatively focus on developing large generative models for graphs with the expectation to realize a similar set of advantages achieved by LGMs in other fields, including enhanced zero-shot generalizability, improved fine-tuning performance and cross-modality generation.

11.2.2 Graph Generative Models

Given the ubiquity of graphs in modeling relational information of real-world objects across many domains [336, 312, 309, 11], graph generative models have been developed to generate realistic graphs for advancing numerous applications [297, 337, 299, 316], such as generating molecular graphs with high drug-likeness and designing imperceptible adversarial attacks. Graph generative models can generally be divided into two categories: statistic-based ones [338, 339] and deep learning-based ones [304, 305]. Statistic-based generative models such as Stochastic Block Models [340] and Small World Models [341] assume that the real-world graph formation adheres to specific statistical rules, and define various sampling strategies to simulate networks with prescribed properties. However, this approach oversimplifies the complex distribution of real-world graphs and struggles to generalize to those deviating from established norms. This limitation has spurred recent research into deep-learning-based generative models that automatically capture intricate statistics by learning to recover graphs [307, 308, 306, 332]. Despite their effectiveness, they all focus on a narrow range of domains and are trained solely on a single domain each time. Next, we briefly review representative deep graph generative models in Table 11.1.

Table 11.1: Our LGGM is trained across 13 domains on thousands of graphs with the support for Text-to-Graph (T2G) generation, controlling the domain/property of the generated graphs.

Type	Model	# Domains	Multi-Domain Training	T2G Domain	T2G Property
Auto-Regressive	GraphRNN [306]	2	✘	✘	✘
	EdgeRNN [323]	3	✘	✘	✘
	MolRNN [324]	2	✘	✘	✘
VAE	MDVAE [325]	1	✘	✘	✘
	PCVAE [326, 327]	3	✘	✘	✘
	(DE)CO-VAE [328]	1	✘	✘	✘
	GraphVAE [307]	1	✘	✘	✘
GAN	Mol-CycleGAN [329]	1	✘	✘	✘
	LGGAN [330]	2	✘	✘	✘
Flow	GraphNVP [331]	1	✘	✘	✘
	MoFlow [332]	1	✘	✘	✘
	GraphDF [333]	2	✘	✘	✘
Diffusion	GDSS [334]	3	✘	✘	✘
	DiGress [308]	2	✘	✘	✘
	GraphEBM [335]	1	✘	✘	✘
LGGM - Ours		**13**	✔	✔	✔

GraphRNN [306], a pioneering model in autoregressive generation, employs breadth-first search to establish node ordering and sequentially generates nodes and edges. In addition, variational autoencoders [342, 325, 326, 327, 307] were adopted to enable flexible graph generation tailored to specific properties by regularizing the latent variables. Following that, normalizing flows were used to learn invertible mappings between molecular graphs and latent representations (e.g., GraphNVP [331] and MoFlow [332]). More recently following the success of diffusion-based models in images, graph diffusion-based models like GDSS [334] and DiGress [308] have emerged, allowing gradually generating graphs from the noise either in the continuous or the discrete space. However, these deep graph generative models have primarily focused on limited domains such as chemistry, social networks, and synthetic graphs, neglecting a vast array of unexplored graphs in other fields. More critically, these models have been historically trained on a single domain each time, mirroring earlier approaches like Unet [294] and Bert/Bart [292, 293] in CV and NLP, thus limiting their ability to learn fundamental knowledge transferrable across different domains. In contrast, our work focuses on learning a large generative graph model that is pre-trained on thousands of graphs from 13 domains and we demonstrate that, through extensive experiments, the proposed LGGM successfully replicates the revolutionary achievements gained by the recent LGMs in other fields [286, 287, 288, 275].

11.3 Large Graph Generative Models

11.3.1 Notation

Let $\mathbb{G}$ be a random variable of universal graphs, governed by its underlying distribution $P(\mathbb{G})$. Given that real-world graphs originate from various domains, we introduce $\mathbb{G}^c$ to represent a random variable for graphs from domain c, with its distribution as $P(\mathbb{G}^c)$. Assuming the universal graph space encompasses $\mathcal{C}$ distinct domains, i.e., $\mathcal{G} = \cup_{c\in\mathcal{C}}\mathcal{G}^c$ with each set of graphs from domain c as $\mathcal{G}^c$, then $P(\mathbb{G}^c)/P(\mathbb{G})$ is domain-specific/agnostic distribution. To ease the introduction of training and evaluation setting in Section 11.6, we further divide each domain-specific set of graphs $\mathcal{G}^c$ into training, validation and testing subsets, notated as $\mathcal{G}^c = \mathcal{G}^{\text{Train},c} \cup \mathcal{G}^{\text{Val},c} \cup \mathcal{G}^{\text{Test},c}$. We represent each graph $G = (\mathbf{X}^G, \mathbf{E}^G)$ with $\mathbf{X}^G \in \mathbb{R}^{n_G\times d_X}/\mathbf{E}^G \in \mathbb{R}^{n_G\times n_G\times d_E}$ as the one-hot encoding matrix representing node/edge categories with n_G being the number of nodes in graph G and d_X/d_E being the number of node/edge categories, considering the edge existence as a particular edge category. In Text-to-Graph generation, each graph G is paired with a textual description S from the textual distribution $P(\mathbb{S})$ and their joint distribution is $P(\mathbb{G}, \mathbb{S})$.

11.3.2 Large Graph Corpus

Training LGGM requires a substantial, well-curated collection of graphs from multiple domains. We select graphs from the Network Repository across 13 distinct yet representative domains covering a wide variety of real-world scenarios, including Facebook (FB), Animal Social (ASN), Email, Web, Road, Power, Chemical (CHEM), Biological (BIO), Economic (ECON), Retweet (RT), Collaboration (COL), Ecological (ECO), Citation, as shown in Figure 11.1(a). Given that many real-world graphs (e.g., social networks and road networks) comprise thousands or even millions of nodes and edges, and that state-of-the-art diffusion models, e.g., DiGress and GDSS, are limited to handling networks with only hundreds of nodes, we further sample subgraphs for certain domains to address scalability challenges. Specifically, we generate 2/3-hop ego subgraphs centered on multiple randomly chosen nodes followed by taking their induced subgraphs. We apply this strategy iteratively across all the initially collected graphs until hitting the preset budget. Appendix 11.9.2 presents the graph statistics.

11.3.3 Pre-Training and Graph Generation of LGGM

Our LGGM is designed based on discrete denoising diffusion [343, 344, 308], which is composed of a diffusion forward process based on a transition matrix and a reverse prediction process based on minimizing the cross-entropy loss between the ground-truth graphs and the predicted clean graphs.

During the forward process, for each graph G sampled from the joint distribution $P(\mathbb{G})$, we obtain its noisy version $G^t = (\mathbf{X}^t, \mathbf{E}^t)$ at step t by sampling from the conditional categorical distribution:

$$q(\mathbb{G}^t|\mathbb{G}^{t-1}) = (\mathbf{X}^{t-1}\mathbf{Q}^t_X, \mathbf{E}^{t-1}\mathbf{Q}^t_E) \quad \text{and} \quad q(\mathbb{G}^t|\mathbb{G}^0) = (\mathbf{X}\bar{\mathbf{Q}}^t_X, \mathbf{E}\bar{\mathbf{Q}}^t_E), \tag{11.1}$$

where $\mathbf{Q}^t_X \in \mathbb{R}^{d_X \times d_X}$ and $\mathbf{Q}^t_E \in \mathbb{R}^{d_E \times d_E}$ are node/edge transition matrices and $\mathbb{G}^0 = \mathbb{G}$ is the original data distribution of graphs. Depending on whether our generative downstream tasks require generalization to unseen domains or not, we can either use different transition matrices for graphs from different domains, i.e., domain-specific transition matrix $\mathbf{Q}^{t,c}_X = \alpha^t\mathbf{I} + (1-\alpha^t)\mathbf{1}\mathbf{m}^c_X, \mathbf{m}^c_X = \frac{1}{|\mathcal{G}^{\text{Train},c}|}\sum_{G\in\mathcal{G}^{\text{Train},c}}\mathbf{X}^G, \forall c \in C$ or unify transition matrices across different domains. For the unified transition matrices, we can trivially use the uniform transition matrix, i.e. $\mathbf{Q}^{t,c}_X = \alpha^t\mathbf{I} + (1-\alpha^t)(\mathbf{1}_{d_\mathbf{X}}\mathbf{1}^\top_{d_\mathbf{X}})/d_\mathbf{X}$, or compute the marginal transition matrix across all graphs from all domains $\mathbf{Q}^t_X = \alpha^t\mathbf{I} + (1-\alpha^t)\mathbf{1}\mathbf{m}_X, \mathbf{m}_X = \frac{1}{|\mathcal{G}^{\text{Train}}|}\sum_{G\in\mathcal{G}^{\text{Train}}}\mathbf{X}^G$. And $\mathbf{Q}^t_E$ can be computed similarly.

In the reverse process, a parametrized neural network is trained to predict the clean graph given the noisy graph sampled following Eq. (11.1) by optimizing the following loss:

$$\mathbf{\Theta}^\star = \arg\min_{\mathbf{\Theta}} \mathcal{L} = \mathbb{E}_{G\sim P(\mathbb{G})}\mathbb{E}_{t\sim\mathcal{T}}\mathbb{E}_{G^t\sim q(\mathbb{G}^t|\mathbb{G})}(-\log p_{\mathbf{\Theta}}(G|G^t)). \tag{11.2}$$

Following [308], we combine the learned $P_{\mathbf{\Theta}^\star}(\mathbb{G}|\mathbb{G}^t)$ and the closed-form posterior $P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{G})$ to perform backward generation by sampling from the following distribution:

$$P(\mathbb{G}^{t-1}|\mathbb{G}^t) \propto \sum_{\mathbb{G}} P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{G})P_{\mathbf{\Theta}^*}(\mathbb{G}|\mathbb{G}^t). \tag{11.3}$$

11.4 Fine-tuning LGGM

In many real-world applications, the graphs of interest $\widetilde{\mathcal{G}}$ may highly likely come from completely unseen domains, i.e., $\widetilde{\mathcal{G}} \cap \mathcal{G} = \emptyset$, and their corresponding distribution may also be significantly different from the pre-trained one, i.e., $P(\widetilde{\mathbb{G}}) \neq P(\mathbb{G})$ as shown by comparing CHEM and FB Networks in Figure 11.1(a). In this case, we further fine-tune our pre-trained LGGM based on the observed graphs $\widetilde{\mathcal{G}}$ from the unseen

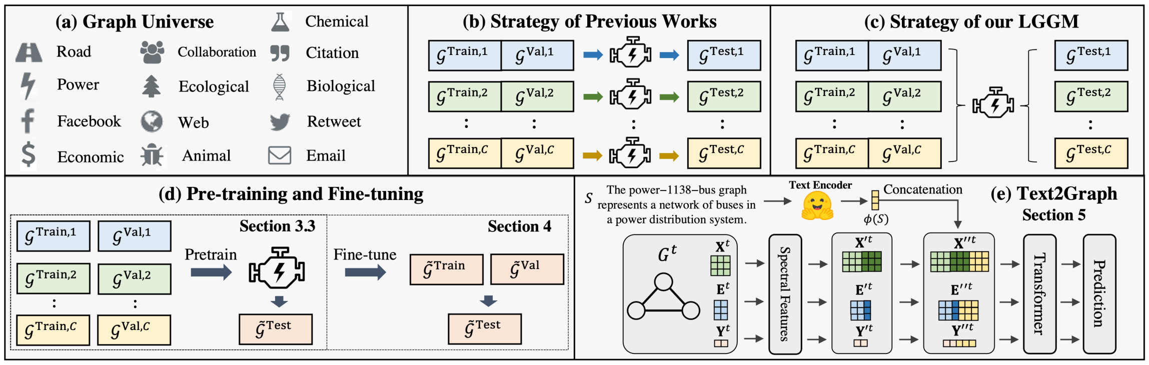


Figure 11.2: The overview of LGGM framework and experimental settings. (a): Graph universe including our collected 13 distinct yet representative domains. (b)-(c): Compared with all previous graph generative models that have been trained only on one domain each time, our LGGM is trained on thousands of graphs from 13 domains. (d): We pre-train/fine-tune LGGM in Section 11.3.3/11.4. (e): Given the text prompt S and the current generated graph at t, we concatenate its textual embedding obtained from a pre-trained language model with the node/edge/graph embeddings after spectral feature extraction and forward them through the Graph Transformer to predict the clean graph.

domains:

$$\mathbf{\Theta}^{\star\star} = \arg\min_{\mathbf{\Theta}} \mathcal{L} = \mathbb{E}_{\widetilde{G}\sim P(\widetilde{\mathbb{G}})}\mathbb{E}_{t\sim\mathcal{T}}\mathbb{E}_{\widetilde{G}^t\sim q(\widetilde{\mathbb{G}}^t|\widetilde{\mathbb{G}})}(-\log p_{\mathbf{\Theta}}(\widetilde{G}|\widetilde{G}^t)), \tag{11.4}$$

where $\mathbf{\Theta}^{\star\star}$ is initialized as $\mathbf{\Theta}^{\star}$ from the pretaining phase in Eq (11.2). After fine-tuning, our LGGM can effectively adapt to unseen distributions by using both the prior knowledge from the pre-training stage and the specific knowledge of new graphs from the unseen domains, as verified in Figure 11.3.

Despite the superior capabilities of LGGM in generating graphs after both pre-training and fine-tuning processes, they essentially mimic the random sampling from the learned distribution $P(\mathbb{G})$ that is prescribed by the training data without any fine-level customization. To control the characteristics of the generated graphs, we further propose the very first Text-to-Graph LGGM to generate graphs based on textual description. In this way, users could specify their desired properties of the graphs through natural language description, thereby guiding the graph generation in a more tailored manner.

11.5 Text-to-Graph LGGM

Given the textual description S about the network to be generated, our goal here is to learn $P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{S})$, which is further decomposed as:

$$P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{S}) \propto \sum_{\mathbb{G}} P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{G}, \mathbb{S})P(\mathbb{G}|\mathbb{G}^t, \mathbb{S}). \tag{11.5}$$

Theorem 9 proves that if the transition matrices $\mathbf{Q}^t_X, \mathbf{Q}^t_E$ in Eq. (11.1) are independent of the textual description S, the first term $P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{G}, \mathbb{S})$ can then be simplified as $P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{G})$ with the analytical form computation [308]. For the second term, we approximate it by a neural network, i.e., $P(\mathbb{G}|\mathbb{G}^t, \mathbb{S}) =$

$P_{\mathbf{\Theta}^\blacktriangle}(\mathbb{G}|\mathbb{G}^t, \mathbb{S})$ with $\mathbf{\Theta}^\blacktriangle$ being optimized by:

$$\mathbf{\Theta}^\blacktriangle = \arg\min_{\mathbf{\Theta}} \mathcal{L} = \mathbb{E}_{(G,S)\sim P(\mathbb{G},\mathbb{S})}\mathbb{E}_{t\sim\mathcal{T}}\mathbb{E}_{G^t\sim q(\mathbb{G}^t|\mathbb{G})}(-\log p_{\mathbf{\Theta}}(G|G^t, \phi(S))), \tag{11.6}$$

where ϕ is a pre-trained textual encoder. Figure 11.2(e) shows the architecture of LGGM-Text2Graph, which firstly integrates the textual embedding $\phi(S)$ into the node/edge/graph-level latent embeddings after spectral feature extraction of the current generated graph and further predicts the clean graph. Theorem 10 proves that modeling $P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{S})$ with $P_{\mathbf{\Theta}^\blacktriangle}(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{S})$ leads to higher evidence lower bound of the likelihood $\log P(\mathbb{G}^0, \mathbb{S})$.

Training $p_{\mathbf{\Theta}}(G|G^t, \phi(S))$ in Eq (11.6) requires the joint distribution between graphs and their corresponding textual descriptions, i.e., $P(\mathbb{G}, \mathbb{S})$. Given users' specific interests in the graphs to generate, we explore two main categories of textual prompts to guide graph generation: domain/name (e.g., Power Network, power-1138-bus) and structural characteristics (e.g., average degree, clustering coefficient). For example, zoologists interested in the dynamics of tortoise interactions might seek to generate Animal Social Networks [345], and social scientists studying social anomalies might prioritize generating social interactions with dense and unexpected connections [346]. Since this work is a pioneering effort in Text-to-Graph generation and no prior collection of user prompts for this purpose exists, following previous works, e.g., LLaVA [321, 347], we ask GPT3.5/4 to emulate the human drafting of prompts to obtain pairs of (user prompt, graph). For preparing the graphs with user prompts about their domains/names, we obtain the domain/name information of each graph directly from the Network Repository [312] and prompt GPT3.5 to generate the human-readable description paired with the corresponding graph. See more details in Appendix 11.9.3. For preparing the graphs with user prompts about their average clustering coefficient/degree, instead of using graphs from Network Repository that only count partially of the entire graph universe (i.e., no existing graphs there cover the area with high average degree and low average clustering coefficient in Figure 11.1(a)), we use the Watts–Strogatz small-world graph model [348] to synthesize graphs covering the full spectrum of the graph universe. After that, we calculate the average degree and clustering coefficient for each graph and prompt GPT4 to generate textual descriptions about these networks using their statistics. See more details in Appendix 11.9.4. We also employ t-SNE visualization [349] to analyze the generated textual descriptions, as shown in Figure 11.6. This visualization indicates that texts describing graphs from various domains or with distinct statistics tend to form separate clusters, a necessary condition for the successful control of the generated graphs.

11.6 Experiments

11.6.1 Experimental Setup

In this section, we conduct four experiments over the graphs collected from 13 domains to demonstrate the effectiveness of LGGMs in four different aspects, the details of which are summarized as follows:

- **Pre-training Evaluation in Table 11.2 in Section 11.6.2:** To demonstrate the superior zero-shot performance of LGGM in generating unseen graphs compared to conventional graph generative models, we adopt the out-of-distribution evaluation where we iteratively treat each domain X as the unseen one and train the LGGM using training graphs from all other domains, and evaluate its performance on the testing graphs from the unseen domain X. The variant of LGGM in this experiment is called LGGM-X where X represents the unseen domain.

- **Fine-tuning Evaluation in Figure 11.3 in Section 11.6.3:** To demonstrate the high adaptability for fine-tuning LGGM, we further fine-tune the above pre-trained LGGM. Specifically, we take LGGM-X pre-trained on graphs from all other domains but domain X, and then fine-tune it on the training graphs from domain X. After that we evaluate it on the testing graphs from domain X. The variant of LGGM in this experiment is called Fine-tuned LGGM on X.

- **Text-to-Graph Generation in Table 11.3 and Figure 11.4 in Section 11.6.4:** To control the graph generation, we consider two types of user prompt information: the domain/name and the graph properties, i.e., we train LGGM on training graphs from all domains with user prompts either describing the graph domains/names or graph statistics. We call these two variants of LGGM as LGGM-T2G$^{\text{D}}$ and LGGM-T2G$^{\text{UP}}$, respectively.

- **Fine-tuned LGGM compared with DiGress trained directly on X in Figure 11.1(b)/11.5 in Section 11.6.5:** When having access to graphs of domain X, users could directly train existing graph generative models and generate graphs for the domain X. To demonstrate the practical usage of LGGMs, we further compare the fine-tuned LGGM on X with DiGress directly trained on X. In addition, we also compare their performance under limited data scenarios [299, 301].

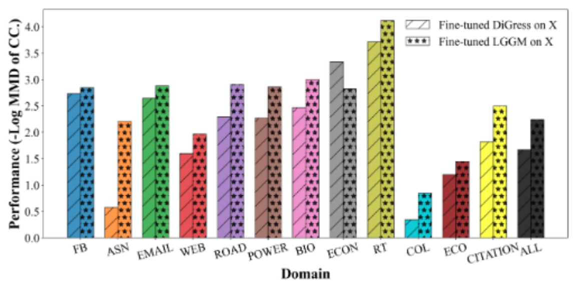
Figure 11.7 in Appendix 11.9.7 comprehensively illustrates each of the above training paradigms. Due to the page limitation, we present the evaluation metrics and model hyperparameters in Appendix 11.9.6.

11.6.2 Pre-training Evaluation

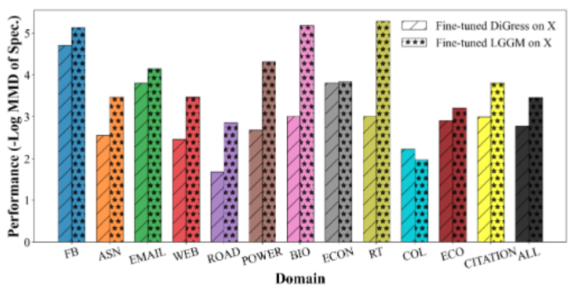
Table 11.2 compares the performance of our model, LGGM-X, pre-trained on all graph domains except the held-out domain X, with DiGress trained on the QM9 dataset. Both of them are evaluated over graphs from

Table 11.2: Comparing Zero-shot Generative Performance on unseen Graphs in held-out domain X between DiGress trained on QM9 and LGGM-X trained on all except the held-out domain X. Result ”ALL” is computed by averaging across 12 domains and the best result for each domain is in **bold**.

Domain	Method	DEG	CC	Spec	Orb	Domain	Method	DEG	CC	Spec	Orb
FB	DiGress	**0.3376**	**0.6298**	**0.0797**	**0.3593**	BIO	DiGress	0.2712	0.5202	0.1127	0.3188
	LGGM-X	0.4723	0.6843	0.2924	0.7555		LGGM-X	**0.1081**	**0.2696**	**0.0900**	**0.2053**
ASN	DiGress	0.1496	0.3258	0.1506	0.4420	ECON	DiGress	0.2987	0.4841	0.2162	0.3834
	LGGM-X	**0.0281**	**0.2440**	**0.0830**	**0.0618**		LGGM-X	**0.1213**	**0.0920**	**0.1120**	**0.1086**
EMAIL	DiGress	0.2192	0.6012	**0.0702**	0.3416	RT	DiGress	0.4164	**0.1327**	0.4147	0.5957
	LGGM-X	**0.0751**	**0.2364**	0.0768	**0.3089**		LGGM-X	**0.0525**	0.1429	**0.1330**	**0.2219**
WEB	DiGress	0.2556	0.6186	0.1877	0.6045	COL	DiGress	0.2473	0.5826	0.2314	0.7679
	LGGM-X	**0.0648**	**0.3961**	**0.0549**	**0.1127**		LGGM-X	**0.0736**	**0.5769**	**0.0895**	**0.0988**
ROAD	DiGress	0.3705	0.8226	0.2801	0.7198	ECO	DiGress	0.5431	0.7915	**0.2338**	0.6045
	LGGM-X	**0.0713**	**0.2193**	**0.0987**	**0.2986**		LGGM-X	**0.4753**	**0.3904**	0.3194	**0.3934**
POWER	DiGress	0.3726	0.4582	0.3270	1.4732	CITATION	DiGress	0.2527	0.7790	0.1315	**0.4966**
	LGGM-X	**0.0119**	**0.1293**	**0.0373**	**0.0754**		LGGM-X	**0.1348**	**0.7257**	**0.1160**	0.4981
ALL	DiGress	0.3112	0.5622	0.2030	0.5923						
	LGGM-X	**0.1408**	**0.3422**	**0.1253**	**0.2616**						



(a) Performance of MMD of CC.



(b) Performance of MMD of Spec.

Figure 11.3: Performance comparison between Fine-tuned LGGM and Fine-tuned DiGress.

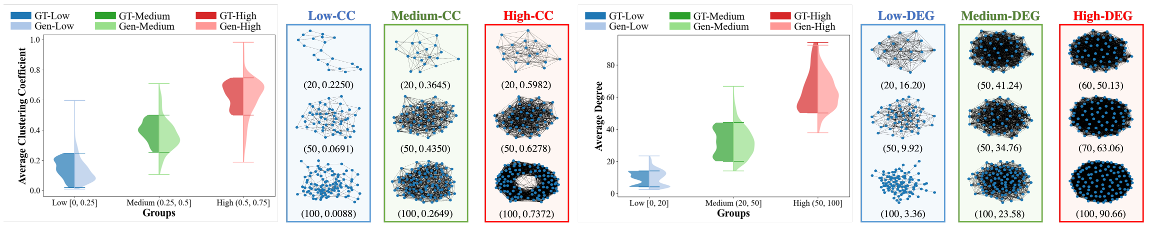
the unseen domain X. Overall, LGGM-X outperforms DiGress across all evaluation metrics shown by the ”ALL” result. This superiority suggests that training on graphs from diverse domains captures transferable structural patterns and enhances the generalization of the model to unseen domains. The only exception from this trend occurs with Facebook Networks (FB) where our LGGM-X performs uniformly worse than DiGress across all evaluation metrics. This is because Facebook Networks (FB) only count a tiny region among the whole graph universe. As illustrated in Figure 11.1(a), the average clustering coefficient of FB graphs ranges from 0.301 to 0.407, a narrow segment within the broader global graph spectrum spanning from 0 to 1. This narrow range poses a challenge for the generalized LGGM-X to specialize in learning the graph data distribution specific to the FB domain.

11.6.3 Fine-tuning Evaluation

In addition to the superior zero-shot generative performance of pre-trained LGGM-X, many real-world applications already possess exemplary graphs that can be leveraged, e.g., different types of anomaly behaviors in social networks/e-commerce platforms, and molecules with predefined chemical structures in drug dis-

Table 11.3: Comparing the Graph Generative Performance of LGGM with/without Text Conditions. Best and runner-up results are **bolded** and underlined.

Domain	Method	DEG	CC	Spec	Orb	Domain	Method	DEG	CC	Spec	Orb
FB	LGGM	0.0321	0.4994	0.0763	0.3117	BIO	LGGM	0.2661	0.3120	0.1135	0.3835
	LGGM-T2G^{D}	0.1561	0.1639	0.0924	0.0417		LGGM-T2G^{D}	0.0099	0.1286	0.0303	0.1366
	LGGM-T2G^{UP}	**0.0050**	**0.0545**	**0.0070**	**0.0251**		LGGM-T2G^{UP}	**0.0028**	**0.0287**	**0.0236**	**0.0174**
ASN	LGGM	0.1511	0.4325	0.1875	0.3896	ECON	LGGM	0.3828	0.1533	0.2039	0.2583
	LGGM-T2G^{D}	0.0318	0.2821	0.0606	0.0631		LGGM-T2G^{D}	0.0666	0.0594	0.0650	0.0586
	LGGM-T2G^{UP}	**0.0211**	**0.1191**	**0.0462**	**0.0195**		LGGM-T2G^{UP}	**0.0132**	**0.0257**	**0.0053**	**0.0191**
EMAIL	LGGM	0.2156	0.2450	0.0666	0.2757	RT	LGGM	0.4395	0.2225	0.4337	0.6641
	LGGM-T2G^{D}	0.0469	0.0982	0.0484	0.0505		LGGM-T2G^{D}	0.0468	0.0955	0.0729	0.0393
	LGGM-T2G^{UP}	**0.0073**	**0.0379**	**0.0127**	**0.0437**		LGGM-T2G^{UP}	**0.0286**	**0.0933**	**0.0400**	**0.0312**
WEB	LGGM	0.2725	0.2672	0.1900	0.4368	COL	LGGM	0.3565	0.3554	0.2451	0.7874
	LGGM-T2G^{D}	0.0255	**0.0737**	0.0354	0.1856		LGGM-T2G^{D}	0.0395	0.3110	0.1146	0.1823
	LGGM-T2G^{UP}	**0.0105**	0.0941	**0.0206**	**0.0451**		LGGM-T2G^{UP}	**0.0265**	**0.2813**	**0.0895**	**0.0899**
ROAD	LGGM	0.4825	0.5373	0.3398	0.7542	ECO	LGGM	0.5466	0.6003	0.2257	0.7089
	LGGM-T2G^{D}	**0.0088**	0.1225	0.0399	0.0155		LGGM-T2G^{D}	0.2160	0.2917	0.1203	0.2569
	LGGM-T2G^{UP}	0.0177	**0.0437**	**0.0336**	**0.0086**		LGGM-T2G^{UP}	**0.0293**	**0.2885**	**0.0416**	**0.2556**
POWER	LGGM	0.4394	0.4646	0.3473	1.3186	CITATION	LGGM	0.2624	0.5374	0.1295	0.3419
	LGGM-T2G^{D}	0.0162	0.1131	0.0479	0.1786		LGGM-T2G^{D}	0.0101	0.1025	0.0315	0.0651
	LGGM-T2G^{UP}	**0.0062**	**0.0570**	**0.0111**	**0.0084**		LGGM-T2G^{UP}	**0.0072**	**0.0849**	**0.0115**	**0.0287**
ALL	LGGM	0.3206	0.3856	0.2132	0.5526						
	LGGM-T2G^{D}	0.0562	0.1535	0.0633	0.1061						
	LGGM-T2G^{UP}	**0.0146**	**0.1007**	**0.0286**	**0.0494**						



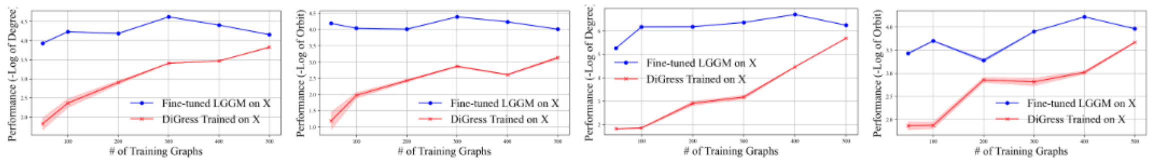
(a) Average Clustering Coefficient (b) Average Degree

Figure 11.4: Text-to-Graph Generation with Prescribed Graph Properties. (a) Controlling Average Clustering Coefficient; (b) Controlling Average Degree. GT-Ground Truth Graphs and Gen-Generated Graphs. Below each graph, the number of nodes and key statistical measures are displayed.

covery. In these scenarios, users can fine-tune LGGM-X with these domain-specific graphs, adapting the broadly trained model to specialize in generating graphs tailored to target domains. Figure 11.3 compares the generative performance of fine-tuned DiGress on X that is originally pre-trained on QM9 and fine-tuned LGGM-X on X that is originally pre-trained on all but domain X. We can see that LGGM-X consistently outperforms DiGress for graphs from most of the domains, which further validates the adaptability of LGGM after fine-tuning on a specific domain.

11.6.4 Text-to-Graph Generation

Here we integrate Text-to-Graph (T2G) generation into LGGMs. We introduce two variants: LGGM-T2G^{D}, which utilizes domain labels such as ”Power Networks” as textual descriptions, and LGGM-T2G^{UP}, which utilizes user prompts from GPT3.5, like ”The power-1138-bus graph represents a network of buses in a power distribution system”. Table 11.3 compares the basic LGGM trained without text conditions, against



(a) Road-DEG. (b) Road-Orbit. (c) Retweet-DEG. (d) Retweet-Orbit.

Figure 11.5: With fewer training graphs, Fine-tuned LGGM becomes more advantageous than DiGress.

LGGM-T2G^{D} and LGGM-T2G^{UP}. Firstly, we observe a significant performance improvement from LGGM to LGGM-T2G^{D}/LGGM-T2G^{UP}. The inclusion of text descriptions acts as a unique identifier that enables LGGM-T2G to specialize in generating graphs aligning with corresponding domains. Moreover, the network-level user prompts in LGGM-T2G^{UP} provide a finer-level control compared to the domain-level descriptions in LGGM-T2G^{D}, further boosting the performance.

LGGM-T2G can also control the properties of the generated graphs. Here we first synthesize ground-truth graphs with clustering coefficients between [0, 0.75] and average degrees between [0, 100]. We divide these ground-truth graphs into three groups, low/medium/high, and prompt GPT4 to generate user instructions describing these two graph properties (Appendix 11.9.4). Then we combine these three groups of graphs with their instructions to train LGGM-T2G and evaluate whether the properties of the generated graphs align with the instructions. In Figure 11.4(a)/(b), we can see a clear alignment between the statistical properties of the ground-truth graphs and those of the generated graphs, both in terms of the average CC and DEG. Furthermore, we visualize the generated graphs for these three groups in Figure 11.4(a)/(b). We can see graphs in low-CC groups possess many squares while the ones in high-CC groups contain many triangles, aligning with the intuition of CC.

11.6.5 Practical Usage of Fine-tuned LGGM

To demonstrate the practical usage of LGGM in generating graphs for real-world deployment, we further compare the fine-tuned LGGM with DiGress trained directly on each domain in Figure 11.1(b). We can see that even using the same graphs for training, due to the additional knowledge incorporated during the pre-training phase of LGGM, it exhibits significantly better generative performance for most domains. Moreover, this advantage becomes even more pronounced when fewer graphs are available. Figure 11.5 illustrates the enhanced performance of fine-tuned LGGM versus DiGress trained on X, with a widening margin as the number of training graphs in X decreases. This is particularly useful since many graph generative applications involve semi-supervised settings, e.g., generating anomaly software and design of drugs, the amount of which only count 0.05%-0.5% [350] and 0.01% [351] among the whole potential candidates, respectively.

11.7 Research Problems Enabled by LGGMs

As the proposed LGGMs are the first to explore the potential of large generative models in graphs, it will spark numerous transformative research opportunities [352], which are summarized below:

- **Simulations, Extrapolation, Anonymization**: Since our LGGM-T2G can generate graphs with pre-defined properties, we can simulate graphs with various properties and extrapolate new insights from these simulated graphs, e.g., evaluate conventional and newly designed graph algorithms/ models [81]. Moreover, for sensitive real-world graphs, we can maintain confidentiality by sharing only the model, which can then simulate similar graphs without disclosing private information.
- **Data Augmentation**: LGGMs can be used for data augmentation when only limited graphs are available for applications like graph anomaly detection and molecular tasks [299, 353].
- **Graph Compression**: LGGM allows for the compression of graphs across multiple domains by merely storing model parameters instead of the original graphs.

11.8 Limitations, Future Directions, and Conclusion

Limitations and Future Directions: Like LGMs in other fields [354, 355], our LGGMs are not specialized in generating graphs for specific domains. One future direction could be exploring strategies such as Retrieval-Augmented Generation to enhance domain proficiency [11]. Additionally, our evaluation of LGGMs has focused solely on their generative capabilities, without examining their potential usage in downstream tasks. A promising future direction is to assess their practical utility in application-oriented manners, e.g., higher quality of generated graphs for better data augmentation.

Conclusion: Motivated by the recent successes of Large Generative Models (LGMs) across fields of Vision, Language, Video, and Audio, and recognizing the promising practical usage of graph generative models, we introduce, for the very first time, Large Graph Generative Models (LGGMs). These models are trained on over 5,000 graphs sourced from 13 distinct domains from the well-known Network Repository. We empirically verify the superiority of our LGGMs in three aspects. Firstly, our pre-trained LGGM-X models demonstrate exceptional zero-shot generative capabilities. Secondly, LGGMs show remarkable adaptability for fine-tuning, and the fine-tuned LGGM is even more powerful than previous graph generative models trained from scratch. Lastly, our models facilitate Text-to-Graph generation, enabling users to specify domain/network names/statistics through prompts to control the generated graphs. Looking ahead, we identify several potential transformative research problems in Section 11.7. To foster further innovation and community collaboration, we release the complete resources of LGGMs, including code, data, and model checkpoints. We invite the community to use these tools to explore new possibilities in graph generation and beyond.

11.9 Appendix

11.9.1 Proof of Theorems

Theorem 9. *If the transition matrices $\mathbf{Q}_X^t, \mathbf{Q}_E^t$ in Eq.* (11.1) *are independent of the textual description $\mathbb{S}$, then we have $P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{G}, \mathbb{S}) \propto P(\mathbb{G}^t|\mathbb{G}^{t-1})P(\mathbb{G}^{t-1}|\mathbb{G})$ and correspondingly, we have the analytical formed solution, i.e., $P(\mathbf{X}^{t-1}|\mathbf{X}^t, \mathbf{X}, S) \propto \mathbf{X}^t(\mathbf{Q}_X^t)^\top \odot \mathbf{X}\bar{\mathbf{Q}}_X^{t-1}$, $P(\mathbf{E}^{t-1}|\mathbf{E}^t, \mathbf{E}, S) \propto \mathbf{E}^t(\mathbf{Q}_E^t)^\top \odot \mathbf{E}\bar{\mathbf{Q}}_E^{t-1}$ following [308].*

Proof. Applying the Bayes rule, we have:

$$P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{G}, \mathbb{S}) \propto P(\mathbb{G}^{t-1}, \mathbb{G}^t, \mathbb{G}, \mathbb{S}) \propto P(\mathbb{G}^t|\mathbb{G}^{t-1}, \mathbb{G}, \mathbb{S})P(\mathbb{G}^{t-1}, \mathbb{G}, \mathbb{S}) \tag{11.7}$$

$$\propto P(\mathbb{G}^t|\mathbb{G}^{t-1}, \mathbb{G}, \mathbb{S})P(\mathbb{G}^{t-1}|\mathbb{G}, \mathbb{S})P(\mathbb{G}, \mathbb{S}). \tag{11.8}$$

Given the independence of the transition matrix on the textual description S and also the noise is Markovian [308], we have $P(\mathbb{G}^t|\mathbb{G}^{t-1}, \mathbb{G}, \mathbb{S}) = P(\mathbb{G}^t|\mathbb{G}^{t-1})$, $P(\mathbb{G}^{t-1}|\mathbb{G}, \mathbb{S}) = P(\mathbb{G}^{t-1}|\mathbb{G})$, and also the irrelevance of $P(\mathbb{G}, \mathbb{S})$ to $P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{G}, \mathbb{S})$, we then end up with:

$$P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{G}, \mathbb{S}) \propto P(\mathbb{G}^t|\mathbb{G}^{t-1})P(\mathbb{G}^{t-1}|\mathbb{G}). \tag{11.9}$$

Since the distribution of graphs can be decomposed into the distribution of node and edge categories, following [308], we similarly have:

$$P(\mathbf{X}^{t-1}|\mathbf{X}^t, \mathbf{X}, S) \propto P(\mathbf{X}^t|\mathbf{X}^{t-1})P(\mathbf{X}^{t-1}|\mathbf{X}) = \mathbf{X}^t(\mathbf{Q}_X^t)^\top \odot \mathbf{X}\bar{\mathbf{Q}}_X^{t-1}, \tag{11.10}$$

$$P(\mathbf{E}^{t-1}|\mathbf{E}^t, \mathbf{E}, S) \propto P(\mathbf{E}^t|\mathbf{E}^{t-1})P(\mathbf{E}^{t-1}|\mathbf{E}) = \mathbf{E}^t(\mathbf{Q}_E^t)^\top \odot \mathbf{E}\bar{\mathbf{Q}}_E^{t-1}. \tag{11.11}$$

□

Theorem 10. *Given the decomposition in Eq.* (11.5) *that $P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{S}) \propto \sum_{\mathbb{G}} P(\mathbb{G}^{t-1}|\mathbb{G}^t, \mathbb{G}, \mathbb{S})P(\mathbb{G}|\mathbb{G}^t, \mathbb{S})$, optimizing $\mathbf{\Theta}$ according to Eq.* (11.6) *essentially optimizes the variational lower bound of the log-likelihood $P_{\mathbf{\Theta}}(\mathbb{G}^0, \mathbb{S})$.*

Proof. We start directly from the log-likelihood of the joint distribution of $P_\Theta(\mathbb{G}^0, \mathbb{S})$:

$$\begin{aligned}
\log P_{\boldsymbol{\Theta}}(\mathbb{G}^0, \mathbb{S}) &= \log \int P_{\boldsymbol{\Theta}}(\mathbb{G}^0, \mathbb{S}, \mathbb{G}^1, ..., \mathbb{G}^T) d(\mathbb{G}^1, \mathbb{G}^2, ..., \mathbb{G}^T) && (11.12)\\
&= \log \int \frac{P_{\boldsymbol{\Theta}}(\mathbb{G}^0, \mathbb{S}, \mathbb{G}^1, ..., \mathbb{G}^T)}{q(\mathbb{G}^1, \mathbb{G}^2, ..., \mathbb{G}^T)} q(\mathbb{G}^1, \mathbb{G}^2, ..., \mathbb{G}^T) d(\mathbb{G}^1, \mathbb{G}^2, ..., \mathbb{G}^T) && (11.13)\\
&= \log \mathbb{E}_{q(\mathbb{G}^1, \mathbb{G}^2, ..., \mathbb{G}^T)} \frac{P_{\boldsymbol{\Theta}}(\mathbb{G}^0, \mathbb{S}, \mathbb{G}^1, ..., \mathbb{G}^T)}{q(\mathbb{G}^1, \mathbb{G}^2, ..., \mathbb{G}^T)} && (11.14)\\
&\geq \mathbb{E}_{q(\mathbb{G}^1, \mathbb{G}^2, ..., \mathbb{G}^T)} \log \frac{P_{\boldsymbol{\Theta}}(\mathbb{G}^0, \mathbb{S}, \mathbb{G}^1, ..., \mathbb{G}^T)}{q(\mathbb{G}^1, \mathbb{G}^2, ..., \mathbb{G}^T)} \quad \text{by Jensen's inequality} && (11.15)\\
&= \mathbb{E}_{q(\mathbb{G}^1, \mathbb{G}^1, ..., \mathbb{G}^T)} \log \frac{P(\mathbb{G}^T, \mathbb{S}) \prod_{t=1}^{T} P_{\boldsymbol{\Theta}}(\mathbb{G}^{t-1} | \mathbb{G}^t, \mathbb{S})}{q(\mathbb{G}^1) \prod_{t=2}^{T} q(\mathbb{G}^t | \mathbb{G}^{t-1})} \quad \text{by Markovian} && (11.16)\\
&= \mathbb{E}_{q(\mathbb{G}^0, \mathbb{G}^1, ..., \mathbb{G}^T)} [\log P(\mathbb{G}^T, \mathbb{S}) + \sum_{t=1}^{T} \log \frac{P_{\boldsymbol{\Theta}}(\mathbb{G}^{t-1} | \mathbb{G}^t, \mathbb{S})}{q(\mathbb{G}^t | \mathbb{G}^{t-1})}] + \text{const.} && (11.17)
\end{aligned}$$

According to the decomposition in Eq (11.3), optimizing $\boldsymbol{\Theta}$ according to Eq. (11.6) leads to optimizing $P_{\boldsymbol{\Theta}}(\mathbb{G}^{t-1} | \mathbb{G}^t, \mathbb{S})$, which corresponds to the second term in Eq. (11.17) and subsequently optimizes the variational lower bound of the log-likelihood $P_{\boldsymbol{\Theta}}(\mathbb{G}^0, \mathbb{S})$ according to the derivation from Eq. (11.12) to Eq. (11.17). Therefore, training Text-to-Graph LGGM according to Eq. (11.6) enables the model to generate graphs such that the pairs of texts and graphs end up with higher likelihoods.

□

11.9.2 Pre-processed Graphs for Training LGGMs

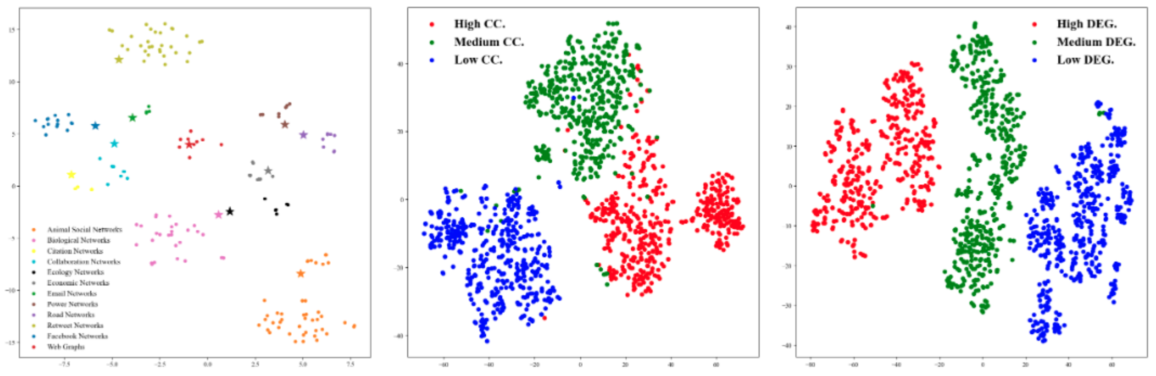
We select graphs from the Network Repository across 13 distinct yet representative domains covering a wide variety of real-world scenarios, including Facebook (FB), Animal Social (ASN), Email, Web, Road, Power, Chemical (CHEM), Biological (BIO), Economic (ECON), Retweet (RT), Collaboration (COL), Ecological (ECO), Citation. Due to the scalability issue with diffusion-based graph generative models, we further sample subgraphs for certain domains, and Table 11.4 presents the comprehensive statistics of the sampled subgraphs, which are used for training LGGMs. We can see that graphs from different domains are statistically different.

11.9.3 Preparation of Graphs and Textual Description About Their Domains/Names

Here we thoroughly discuss the process of obtaining graphs and their corresponding text prompts describing their domains/names. As given by the Network Repository, we directly download graphs along with their domains/names. We then prompt GPT3.5 to generate user prompts describing the graph given its domain/-name. Moreover, we apply the sentence transformer to obtain text embeddings of the generated prompts for each network and perform t-SNE visualization. As shown in Figure 11.6(a), we see prompts for graphs from different domains from different clusters. More importantly, textual similarity can somewhat reflect their

Table 11.4: Summary of Graph Statistics. Facebook (FB), Animal Social (ASN), Email, Web, Road, Power, Chemical (CHEM), Biological (BIO), Economic (ECON), Retweet (RT), Collaboration (COL), Ecological (ECO), Citation.

Category	Num Nodes	Num Edges	Avg Degree	Avg Clustering	Max Nodes	Min Nodes	Max Edges	Min Edges	Num Graphs
ASN	52.47 ± 40.13	77.59 ± 80.95	2.62 ± 1.52	0.395 ± 0.178	283	3	515	2	267
BIO	191.14 ± 43.47	965.71 ± 878.35	9.16 ± 7.69	0.276 ± 0.199	258	109	4392	96	504
CHEM	36.46 ± 20.49	64.61 ± 26.23	3.75 ± 0.63	0.421 ± 0.223	125	2	149	1	646
Citation	235.91 ± 27.25	1287.16 ± 1087.00	10.17 ± 8.14	0.369 ± 0.224	270	175	4474	188	504
COL	174.26 ± 53.82	312.56 ± 176.33	3.41 ± 1.24	0.497 ± 0.203	247	52	996	68	504
ECO	100.67 ± 30.10	1490.00 ± 673.87	27.72 ± 7.00	0.406 ± 0.082	128	54	2106	353	6
ECON	144.18 ± 35.82	3258.76 ± 3540.28	39.76 ± 37.80	0.419 ± 0.296	219	90	11142	188	504
Email	146.67 ± 35.86	681.55 ± 500.28	9.79 ± 7.26	0.389 ± 0.211	213	82	2909	216	504
Power	132.22 ± 20.29	289.32 ± 183.02	4.35 ± 2.31	0.161 ± 0.164	187	81	1332	133	512
Road	265.25 ± 94.31	276.46 ± 79.61	2.70 ± 2.08	0.078 ± 0.134	411	32	456	137	504
RT	104.11 ± 35.23	110.99 ± 46.44	2.11 ± 0.37	0.028 ± 0.038	175	35	295	34	558
FB	219.45 ± 47.05	1863.44 ± 701.53	16.36 ± 6.17	0.315 ± 0.083	259	48	3898	46	504
Web	173.32 ± 24.86	462.21 ± 336.46	5.09 ± 3.06	0.404 ± 0.196	231	119	1607	149	504



(a) Domain and Name (b) Average Clustering Coefficient (c) Average Degree

Figure 11.6: t-SNE visualization of textual description about network (a) domain/name (b) average clustering coefficient (c) average degree.

network similarity. For example, prompts for road and power networks are very close, and they both belong to infrastructure. Moreover, Facebook Networks, Email Networks, Collaboration Networks, Web Graphs are very close since all these four belong to some sub-variants of social networks. *This inherent relationship between the textual similarity and structural similarity between two graphs demonstrates that the world knowledge encoded in the text could somehow provide useful preference for the graphs to be generated.*

11.9.4 Preparing Graphs and Their Textual Description about Graph Property

Here we thoroughly discuss the process of obtaining graphs and their corresponding text prompts describing their properties. Our goal is to demonstrate that Text2Graph LGGM can control the statistics of the generated graphs in the full spectrum. However, the graphs obtained directly from the Network Repository do not cover the whole topological space (e.g., Figure 11.1(a) shows that no networks have a higher average degree while low clustering coefficient). Therefore, we plan to synthesize graphs covering the whole space by Watts-Strogatz Small-world Graph Model. We vary the number of nodes between [10, 110], the number of initial neighbors between [5, number of nodes], and also the probability of rewiring each edge between [0, 1] to ensure the generated graphs span across the full spectrum. After that, we group the generated graphs into low, medium, and high groups in terms of their clustering coefficient and average degree. We implement this using NetworkX. After we synthesize graphs and divide them into three groups, we generate user prompts paired with these graphs next. To ensure the compatibility between the synthesis graphs and the generated user prompts. We further replace the number output by GPT4 describing the network property with the real statistic calculated from each network.

11.9.5 Evaluation Metrics

Following [306, 356], we evaluate the graph generation performance by the standard Maximum Mean Discrepancy (MMD) between generated and reference graphs $\mathcal{G}_g, \mathcal{G}_r$:

$$\text{MMD}(\mathcal{G}_g, \mathcal{G}_r) = \frac{1}{m^2}\sum_{i,j=1}^{m} k(\mathbf{x}_i^r, \mathbf{x}_j^r) + \frac{1}{n^2}\sum_{i,j=1}^{n} k(\mathbf{x}_i^g, \mathbf{x}_j^g) - \frac{2}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} k(\mathbf{x}_i^g, \mathbf{x}_j^r), \tag{11.18}$$

where $k(\cdot,\cdot)$ is a general kernel function and specifically we use RBF kernel following [306]:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-d(\mathbf{x}_i, \mathbf{x}_j)/2\sigma^2), \tag{11.19}$$

where $d(\cdot,\cdot)$ computes pairwise distance following [308] and MMD is evaluated over the distributions of degree (DEG), clustering coefficients (CC), eigenvalues of normalized Laplacian matrix (Spec) and orbits counts representing the distribution of all substructures of size 4 (Orb).

11.9.6 Hyperparameter Details

For all experiments, we select the best configuration according to the generation performance on validation graphs and report the final performance on generating testing graphs. We adopt the default hyperparameter settings from DiGress [308] with the following exceptions: we generate 100 graphs per domain for each evaluation and set the training epochs at 300 to ensure convergence. Additionally, we implement gradient accumulation, using a mini-batch size of 12 across 4 accumulations, resulting in an effective batch size of 48. For Text-to-Graph Generation, the textual encoder used to obtain textual description embeddings is ”all-MiniLM-L6-v2”. All experiments are performed on machines with A100-80G GPU RAM and 128GB RAM.

11.9.7 Paradigm Setup

Figure 11.7 comprehensively visualizes the training/evaluation paradigms of the four experiments, the details of which are discussed in Section 11.6.1.

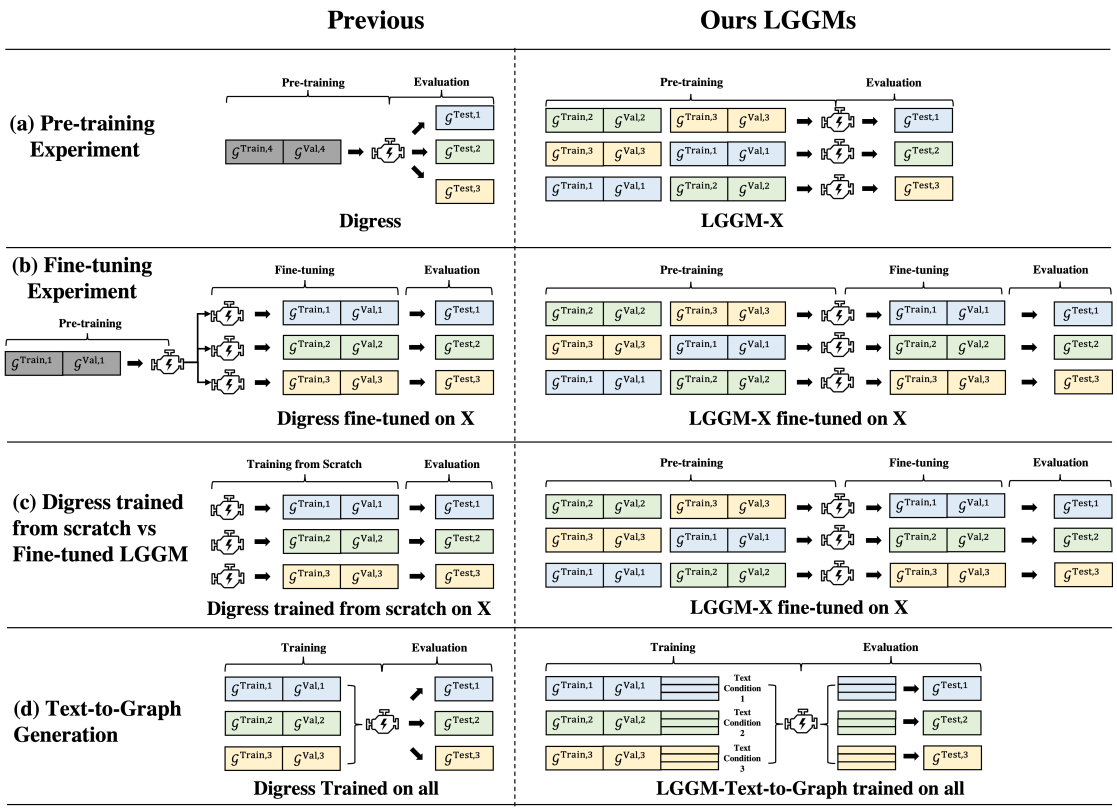


Figure 11.7: Comprehensive Overview of the Experimental Setup for our LGGMs.

CHAPTER 12

Conclusion and Future Work

In this chapter, I summarize the contributions of Data Quality-Aware Graph Machine Learning, which includes systematics study over four different data quality issues on graph-structed data. Later, I briefly introduce some future directions of graph machine learning.

12.1 Conclusion

The focus of artificial intelligence (AI) has recently shifted from a model-centric approach, which primarily emphasizes the performance of AI models, to a data-centric perspective that prioritizes the quality and quantity of data. This shift is critical in the domain of Graph Machine Learning (GML), particularly with techniques like Graph Neural Networks (GNNs) that integrate topological and feature information. However, the dependency on node features and graph topology in GML also makes it susceptible to various data quality issues, which can significantly impair its performance. Recognizing the profound impact of these data quality issues on GML, my research has been dedicated to developing a Data Quality-Aware Graph Machine Trained Learning framework. This framework aims to identify and diagnose data quality issues within graph-structured data and address these through innovative solutions that span both model and data-centric strategies. Concretely, the investigated data quality issues include topology issues, imbalance issues, bias issues and limited data issues.

12.2 Future Work

Marrying Power of AI and Network Science. As the power of any graph machine learning task heavily relies on its underlying network structure, delving deeper into the sophisticated realms of NS would catalyze the evolution of avant-garde GML techniques. Previously, I had applied my NS knowledge in designing model architectures/deriving novel insights in handling/understanding graph data-quality issues, e.g., designing a Breadth-First-Search-based tree decomposition algorithm to enhance node classification on heterophily networks or devising a topological concentration metric to better characterize the node LP performance. Following this research principle, I hope to continuously bridge the profound knowledge of NS into tailoring state-of-the-art machine learning techniques. Concretely, I plan to (1) equip Artificial Intelligence Generated Content (AIGC) with NS/GT by fusing topology-based regularization constraining the generation process of existing graph diffusion methods (e.g., I deeply collaborate with my labmate in molecular ML and plan to follow-up work on enhancing imbalance drug discovery by diffusion-based molecular generation); (2)

design novel topological encodings to make large-language models(LLMs) fully aware of the complex network structure (e.g., I am currently collaborating with Adobe researchers in designing position/role-based topological encoding techniques to augmenting the LLMs' capability for the textual generation.); (3) investigate the applications of network dynamics in designing lifelong GML (e.g., my work identifies a topological distribution shift that newly-joined neighbors become less connective with existing neighbors of a node);

Harmonizing Knowledge Graph (KG) and Large Language Models (LLMs). Large language models (LLMs), such as LLaMA2 and GPT4, are making new waves in natural language processing and artificial intelligence due to their human-like capability and domain-agnostic generalizability. However, LLMs are black-box models, falling short of capturing and accessing factual knowledge. In contrast, Knowledge Graphs (KGs), such as Wikipedia Knowledge Base, are structured databases explicitly storing interpretable factual knowledge. KGs can enhance LLMs by providing external knowledge for grounding, rationalizing, and interpreting. However, KGs are hard to construct, usually domain-specific, and consistently evolve by nature, which limits their long-term benefits to broad real-world applications. Therefore, it is complementary to bridge LLMs and KGs together and simultaneously harness the strength of both. My forward-looking roadmap for this research field is bipartite: (1) LLMs-augmented KG: leverage LLMs to improve/create novel KG signals for enhancing/completing KG-based tasks such as knowledge graph completion/question-answering. (2) KG-augmented LLMs: incorporate KG during the training/inference phase of LLMs to ground/constrain the generated information from LLMs. One golden example is my previous intern project, which leverages the document KGs to mitigate the hallucination of LLMs (KG-augmented LLMs) and leverage LLMs to guide the graph traversal (LLM-augmented KGs).

Data-centric AI for Social-Good Applications The artificial intelligence (AI) community has traditionally taken a model-centric perspective and primarily focuses on developing models for refreshing state-of-the-art performance while keeping the datasets untouched. However, many of these improvements are narrowly domain-specific and have shown the power exclusively on benchmark datasets, which overlooks potential data quality issues such as missing values/anomalies/imbalance/bias/incorrect annotations and behave disastrously in real-world scenarios outside the training domains. Furthermore, much of model-centric AI has been driven by a leaderboard mentality associated with these benchmark datasets, resulting in part of the research community being biased towards more and more complex models that achieve more excellent performance yet more unrealistic utility. This has led to the recent rise in data-centric AI, which emphasizes curating and refining data used within AI models. In my future research endeavors, I am committed to advancing this data-centric AI direction, as already evidenced by my previous research, to curate the data from the wild and derive the most appropriate signals for downstream applications. Beyond research on data-centric AI, I am keen on translating my findings into tangible real-world applications for social good.

References

[1] Yu Wang, Yuying Zhao, Yi Zhang, and Tyler Derr. Collaboration-aware graph convolutional network for recommender systems. In *Proceedings of the ACM Web Conference 2023*, pages 91–101, 2023.

[2] Yu Wang and Tyler Derr. Tree decomposed graph neural network. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 2040–2049, 2021.

[3] Yu Wang, Tong Zhao, Yuying Zhao, Yunchao Liu, Xueqi Cheng, Neil Shah, and Tyler Derr. A topological perspective on demystifying GNN-based link prediction performance. In *The Twelfth International Conference on Learning Representations*, 2024.

[4] Yu Wang and Tyler Derr. Degree-related bias in link prediction. In *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 757–758. IEEE, 2022.

[5] Yu Wang, Amin Javari, Janani Balaji, Walid Shalaby, Tyler Derr, and Xiquan Cui. Knowledge graph-based session recommendation with session-adaptive propagation. In *Companion Proceedings of the ACM on Web Conference 2024*, pages 264–273, 2024.

[6] Yu Wang. Fair graph representation learning with imbalanced and biased data. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, 2022.

[7] Yu Wang, Yuying Zhao, Neil Shah, and Tyler Derr. Imbalanced graph classification via graph-of-graph neural networks. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 2067–2076, 2022.

[8] Yu Wang, Charu Aggarwal, and Tyler Derr. Distance-wise prototypical graph neural network in node imbalance classification. *arXiv preprint arXiv:2110.12035*, 2021.

[9] Yu Wang, Yuying Zhao, Yushun Dong, Huiyuan Chen, Jundong Li, and Tyler Derr. Improving fairness in graph neural networks via mitigating sensitive attribute leakage. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1938–1948, 2022.

[10] Yu Wang, Nedim Lipka, Ruiyi Zhang, Alexa Siu, Yuying Zhao, Bo Ni, Xin Wang, Ryan Rossi, and Tyler Derr. Augmenting textual generation via topology aware retrieval. *arXiv preprint arXiv:2405.17602*, 2024.

[11] Yu Wang, Nedim Lipka, Ryan A Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. Knowledge graph prompting for multi-document question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19206–19214, 2024.

[12] Namyong Park Huiyuan Chen Nesreen K. Ahmed Puja Trivedi Franck Dernoncourt Danai Koutra Tyler Derr Yu Wang, Ryan A. Rossi. Large graph generative models. In *In submission*.

[13] Tyler Derr, Yao Ma, Wenqi Fan, Xiaorui Liu, Charu Aggarwal, and Jiliang Tang. Epidemic graph convolutional network. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 160–168, 2020.

[14] William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

[15] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR*, 2017.

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

[17] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[18] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[19] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2018.

[20] Lei Cai and Shuiwang Ji. A multi-scale approach for graph link prediction. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, 2020.

[21] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018.

[22] Yu Rong, Tingyang Xu, Junzhou Huang, Wenbing Huang, Hong Cheng, Yao Ma, Yiqi Wang, Tyler Derr, Lingfei Wu, and Tengfei Ma. Deep graph learning: Foundations, advances and applications. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3555–3556, 2020.

[23] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, 2019.

[24] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, 2020.

[25] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[26] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2020.

[27] Carey E. Priebe, Cencheng Shen, Ningyuan Huang, and Tianyi Chen. A simple spectral failure mode for graph convolutional networks. *CoRR*, abs/2010.13152, 2020.

[28] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 2019.

[29] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *7th International Conference on Learning Representations, ICLR*, 2019.

[30] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[31] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[32] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33, 2020.

[33] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018.

[34] Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra. Graph neural networks with heterophily. *arXiv preprint arXiv:2009.13566*, 2020.

[35] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *Proceedings of the 16th international conference on World Wide Web*, pages 201–210, 2007.

[36] Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. *arXiv preprint arXiv:2102.06462*, 2021.

[37] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. Direct multi-hop attention based graph neural network. *arXiv preprint arXiv:2009.14332*, 2020.

[38] Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.

[39] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 1–6, 1987.

[40] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Transactions On Algorithms (TALG)*, 1(1):2–13, 2005.

[41] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. Deep graph neural networks with shallow subgraph samplers. *arXiv preprint arXiv:2012.01380*, 2020.

[42] Stanley Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967.

[43] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1024–1034, 2017.

[44] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[45] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[46] Fionn Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5-6):183–197, 1991.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems (NeurIPS) 2019*, pages 8024–8035, 2019.

[48] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[49] Haoyu Han, Xiaorui Liu, Feng Shi, MohamadAli Torkamani, Charu C Aggarwal, and Jiliang Tang. Towards label position bias in graph neural networks. *arXiv preprint arXiv:2305.15822*, 2023.

[50] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD*, pages 974–983, 2018.

[51] Benedek Rozemberczki, Charles Tapley Hoyt, Anna Gogleva, Piotr Grabowski, Klas Karis, Andrej Lamov, Andriy Nikolov, Sebastian Nilsson, Michael Ughetto, Yu Wang, et al. Chemicalx: A deep learning library for drug pair scoring. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3819–3828, 2022.

[52] Juanhui Li, Harry Shomer, Jiayuan Ding, Yiqi Wang, Yao Ma, Neil Shah, Jiliang Tang, and Dawei Yin. Are message passing neural networks really helpful for knowledge graph completion? In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10696–10711, 2023.

[53] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[54] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 556–559, 2003.

[55] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[56] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[57] Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Hammerla, Michael M Bronstein, and Max Hansmire. Graph neural networks for link prediction with subgraph sketching. *arXiv preprint arXiv:2209.15486*, 2022.

[58] Tong Zhao, Gang Liu, Daheng Wang, Wenhao Yu, and Meng Jiang. Learning from counterfactual links for link prediction. In *International Conference on Machine Learning*, pages 26911–26926. PMLR, 2022.

[59] Huiyuan Chen, Lan Wang, Yusan Lin, Chin-Chia Michael Yeh, Fei Wang, and Hao Yang. Structured graph convolutional networks with stochastic masks for recommender systems. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 614–623, 2021.

[60] Xianfeng Tang, Huaxiu Yao, Yiwei Sun, Yiqi Wang, Jiliang Tang, Charu Aggarwal, Prasenjit Mitra, and Suhang Wang. Investigating and mitigating degree-related biases in graph convoltuional networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1435–1444, 2020.

[61] Haitao Mao, Zhikai Chen, Wei Jin, Haoyu Han, Yao Ma, Tong Zhao, Neil Shah, and Jiliang Tang. Demystifying structural disparity in graph neural networks: Can one size fit all? *arXiv preprint arXiv:2306.01323*, 2023.

[62] Hibiki Taguchi, Xin Liu, and Tsuyoshi Murata. Graph convolutional networks for graphs containing missing features. *Future Generation Computer Systems*, 117:155–168, 2021.

[63] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In *Proceedings of the 14th ACM international conference on web search and data mining*, pages 833–841, 2021.

[64] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert systems with applications*, 41(4):2065–2073, 2014.

[65] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. In *Proceedings of the aaai conference on artificial intelligence*, volume 35, pages 11015–11023, 2021.

[66] Zemin Liu, Trung-Kien Nguyen, and Yuan Fang. Tail-gnn: Tail-node graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1109–1119, 2021.

[67] Bowen Hao, Jing Zhang, Hongzhi Yin, Cuiping Li, and Hong Chen. Pre-training graph neural networks for cold-start users and items representation. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 265–273, 2021.

[68] Yunqi Li, Hanxiong Chen, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. User-oriented fairness in recommendation. In *Proceedings of the Web Conference 2021*, pages 624–632, 2021.

[69] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. Mixup for node and graph classification. In *Proceedings of the Web Conference 2021*, pages 3663–3674, 2021.

[70] Joonhyung Park, Jaeyun Song, and Eunho Yang. Graphens: Neighbor-aware ego network synthesis for class-imbalanced node classification. In *International Conference on Learning Representations*, 2021.

[71] Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra. Graph neural networks with heterophily. In *Proceedings of the AAAI conference on artificial intelligence*, pages 11168–11176, 2021.

[72] Haoteng Yin, Muhan Zhang, Yanbang Wang, Jianguo Wang, and Pan Li. Algorithm and system co-design for efficient subgraph-based graph representation learning. *arXiv preprint arXiv:2202.13538*, 2022.

[73] Seongjun Yun, Seoyoon Kim, Junhyun Lee, Jaewoo Kang, and Hyunwoo J Kim. Neo-gnns: Neighborhood overlap-aware graph neural networks for link prediction. *Advances in Neural Information Processing Systems*, 34:13683–13694, 2021.

[74] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The world wide web conference*, pages 417–426, 2019.

[75] Wei Jin, Haitao Mao, Zheng Li, Haoming Jiang, Chen Luo, Hongzhi Wen, Haoyu Han, Hanqing Lu, Zhengyang Wang, Ruirui Li, et al. Amazon-m2: A multilingual multi-locale shopping session dataset for recommendation and text generation. *arXiv preprint arXiv:2307.09688*, 2023.

[76] Hossein A Rahmani, Mohammadmehdi Naghiaei, Mahdi Dehghan, and Mohammad Aliannejadi. Experiments on generalizability of user-oriented fairness in recommender systems. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2755–2764, 2022.

[77] Hao-Ming Fu, Patrick Poirson, Kwot Sin Lee, and Chen Wang. Revisiting neighborhood-based link prediction for collaborative filtering. *arXiv preprint arXiv:2203.15789*, 2022.

[78] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[79] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020.

[80] Liming Pan, Cheng Shi, and Ivan Dokmanić. Neural link prediction with walk pooling. In *International Conference on Learning Representations*, 2022.

[81] John Palowitch, Anton Tsitsulin, Brandon Mayer, and Bryan Perozzi. Graphworld: Fake graphs bring real insights for gnns. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3691–3701, 2022.

[82] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[83] Xiangyu Zhao, Haochen Liu, Wenqi Fan, Hui Liu, Jiliang Tang, Chong Wang, Ming Chen, Xudong Zheng, Xiaobing Liu, and Xiwang Yang. Autoemb: Automated embedding dimensionality search in streaming recommendations. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 896–905. IEEE, 2021.

[84] Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. Streaming graph neural networks. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pages 719–728, 2020.

[85] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

[86] Xiyuan Wang, Haotong Yang, and Muhan Zhang. Neural common neighbor with completion for link prediction. *arXiv preprint arXiv:2302.00890*, 2023.

[87] Haochen Chen, Syed Fahad Sultan, Yingtao Tian, Muhao Chen, and Steven Skiena. Fast and accurate network embeddings via very sparse random projection. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 399–408, 2019.

[88] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. *Advances in neural information processing systems*, 32, 2019.

[89] Juanhui Li, Harry Shomer, Haitao Mao, Shenglai Zeng, Yao Ma, Neil Shah, Jiliang Tang, and Dawei Yin. Evaluating graph neural networks for link prediction: Current pitfalls and new benchmarking. *arXiv preprint arXiv:2306.10453*, 2023.

[90] Sang Gyu Kwak and Jong Hae Kim. Central limit theorem: the cornerstone of modern statistics. *Korean journal of anesthesiology*, 70(2):144–156, 2017.

[91] MN Sanders. Characteristic function of the central chi-squared distribution, 2009.

[92] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[93] Walid Shalaby, Sejoon Oh, Amir Afsharinejad, Srijan Kumar, and Xiquan Cui. M2trec: Metadata-aware multi-task transformer for large-scale and cold-start free session-based recommendations. In *Proceedings of the 16th ACM Conference on Recommender Systems*, pages 573–578, 2022.

[94] Travis Ebesu, Bin Shen, and Yi Fang. Collaborative memory network for recommendation systems. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pages 515–524, 2018.

[95] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[96] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.

[97] Xin Li and Hsinchun Chen. Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach. *Decision Support Systems*, 54(2):880–890, 2013.

[98] Yue Hu, Ao Qu, and Dan Work. Detecting extreme traffic events via a context augmented graph autoencoder. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(6):1–23, 2022.

[99] Zixu Zhuang, Sheng Wang, Liping Si, Kai Xuan, Zhong Xue, Dinggang Shen, Lichi Zhang, Weiwu Yao, and Qian Wang. Local graph fusion of multi-view mr images for knee osteoarthritis diagnosis. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 554–563. Springer, 2022.

[100] Abduallah Mohamed, Kun Qian, Mohamed Elhoseiny, and Christian Claudel. Social-stgcnn: A social spatio-temporal graph convolutional neural network for human trajectory prediction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14424–14432, 2020.

[101] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. Self-supervised graph learning for recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 726–735, 2021.

[102] Yu Wang, Wei Jin, and Tyler Derr. Graph neural networks: Self-supervised learning. In *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 391–420. Springer, 2022.

[103] Wenqi Fan, Xiaorui Liu, Wei Jin, Xiangyu Zhao, Jiliang Tang, and Qing Li. Graph trend networks for recommendations. *arXiv preprint arXiv:2108.05552*, 2021.

[104] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[105] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 452–461, 2009.

[106] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[107] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[108] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Latent relational metric learning via memory-based attention for collaborative ranking. In *WWW*, pages 729–739, 2018.

[109] Marco Gori, Augusto Pucci, V Roma, and I Siena. Itemrank: A random-walk based scoring algorithm for recommender engines. In *IJCAI*, volume 7, pages 2766–2771, 2007.

[110] Xiangnan He, Ming Gao, Min-Yen Kan, and Dingxian Wang. Birank: Towards ranking on bipartite graphs. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):57–71, 2016.

[111] Jheng-Hong Yang, Chih-Ming Chen, Chuan-Ju Wang, and Ming-Feng Tsai. Hop-rec: high-order proximity for implicit recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 140–144, 2018.

[112] Yu Wang and Tyler Derr. Tree decomposed graph neural network. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2040–2049, 2021.

[113] Changxin Tian, Yuexiang Xie, Yaliang Li, Nan Yang, and Wayne Xin Zhao. Learning to denoise unreliable interactions for graph collaborative filtering. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 122–132, 2022.

[114] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

[115] Elizabeth A Leicht, Petter Holme, and Mark EJ Newman. Vertex similarity in networks. *Physical Review E*, 73(2):026120, 2006.

[116] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, 2009.

[117] Mark EJ Newman. Clustering and preferential attachment in growing networks. *Physical review E*, 64(2):025102, 2001.

[118] Muhan Zhang and Yixin Chen. Weisfeiler-lehman neural machine for link prediction. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 575–583, 2017.

[119] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any gnn with local structure awareness. In *International Conference on Learning Representations*, 2021.

[120] Asiri Wijesinghe and Qing Wang. A new perspective on" how graph neural networks go beyond weisfeiler-lehman?". In *ICLR*, 2021.

[121] Meng Liu, Haiyang Yu, and Shuiwang Ji. Your neighbors are communicating: Towards powerful and scalable graph neural networks. *arXiv preprint arXiv:2206.02059*, 2022.

[122] Kelong Mao, Jieming Zhu, Xi Xiao, Biao Lu, Zhaowei Wang, and Xiuqiang He. Ultragcn: Ultra simplification of graph convolutional networks for recommendation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1253–1262, 2021.

[123] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 2011.

[124] Gerard Salton. Automatic text processing: The transformation, analysis, and retrieval of. *Reading: Addison-Wesley*, 169, 1989.

[125] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[126] Junshan Wang, Guojie Song, Yi Wu, and Liang Wang. Streaming graph neural networks via continual learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1515–1524, 2020.

[127] Chen Wang, Yuheng Qiu, Dasong Gao, and Sebastian Scherer. Lifelong graph learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13719–13728, 2022.

[128] Dongmin Park, Hwanjun Song, Minseok Kim, and Jae-Gil Lee. Trap: Two-level regularized autoencoder-based embedding for power-law distributed data. In *Proceedings of The Web Conference 2020*, pages 1615–1624, 2020.

[129] Ruining He and Julian McAuley. Vbpr: visual bayesian personalized ranking from implicit feedback. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

[130] William Webber, Alistair Moffat, and Justin Zobel. A similarity measure for indefinite rankings. *ACM Transactions on Information Systems (TOIS)*, 28(4):1–38, 2010.

[131] Andrew T Stephen and Olivier Toubia. Explaining the power-law degree distribution in a social commerce network. *Social Networks*, 31(4):262–270, 2009.

[132] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *ICLR*, 2016.

[133] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Jiajie Xu, Victor S Sheng S. Sheng, Zhiming Cui, Xiaofang Zhou, and Hui Xiong. Recurrent convolutional neural network for sequential recommendation. In *The world wide web conference*, pages 3398–3404, 2019.

[134] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1419–1428, 2017.

[135] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*, pages 197–206. IEEE, 2018.

[136] Gabriel de Souza Pereira Moreira, Sara Rabhi, Jeong Min Lee, Ronay Ak, and Even Oldridge. Transformers4rec: Bridging the gap between nlp and sequential/session-based recommendation. In *Fifteenth ACM Conference on Recommender Systems*, pages 143–153, 2021.

[137] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, pages 346–353, 2019.

[138] Wei Zhang, Zeyuan Chen, Hongyuan Zha, and Jianyong Wang. Learning from substitutable and complementary relations for graph-based sequential product recommendation. *ACM Transactions on Information Systems (TOIS)*, 40(2):1–28, 2021.

[139] Chao Huang, Jiahui Chen, Lianghao Xia, Yong Xu, Peng Dai, Yanqing Chen, Liefeng Bo, Jiashu Zhao, and Jimmy Xiangji Huang. Graph-enhanced multi-task learning of multi-level transition dynamics for session-based recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4123–4130, 2021.

[140] Ruihong Qiu, Zi Huang, Jingjing Li, and Hongzhi Yin. Exploiting cross-session information for session-based recommendation with graph neural networks. *ACM Transactions on Information Systems (TOIS)*, 38(3):1–23, 2020.

[141] Qi Shen, Lingfei Wu, Yitong Pang, Yiming Zhang, Zhihua Wei, Fangli Xu, and Bo Long. Multi-behavior graph contextual aware network for session-based recommendation. *arXiv preprint arXiv:2109.11903*, 2021.

[142] Lianghao Xia, Chao Huang, Yong Xu, Peng Dai, Xiyue Zhang, Hongsheng Yang, Jian Pei, and Liefeng Bo. Knowledge-enhanced hierarchical graph transformer network for multi-behavior recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4486–4493, 2021.

[143] Zhiqiang Pan, Fei Cai, Wanyu Chen, and Honghui Chen. Graph co-attentive session-based recommendation. *ACM Transactions on Information Systems (TOIS)*, 40(4):1–31, 2021.

[144] Ziyang Wang, Wei Wei, Gao Cong, Xiao-Li Li, Xian-Ling Mao, and Minghui Qiu. Global context enhanced graph neural networks for session-based recommendation. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pages 169–178, 2020.

[145] Zihan Wang, Ziheng Jiang, Zhaochun Ren, Jiliang Tang, and Dawei Yin. A path-constrained framework for discriminating substitutable and complementary products in e-commerce. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 619–627, 2018.

[146] Julian McAuley, Rahul Pandey, and Jure Leskovec. Inferring networks of substitutable and complementary products. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794, 2015.

[147] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*, pages 2704–2710, 2020.

[148] Rongzhi Zhang, Yulong Gu, Xiaoyu Shen, and Hui Su. Knowledge-enhanced session-based recommendation with temporal transformer. *arXiv preprint arXiv:2112.08745*, 2021.

[149] Priyanka Gupta, Diksha Garg, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. Niser: Normalized item and session representations to handle popularity bias. *arXiv preprint arXiv:1909.04276*, 2019.

[150] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR, 2020.

[151] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1419–1428, 2017.

[152] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[153] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[154] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. Meta-gnn: On few-shot node classification in graph meta-learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2357–2360, 2019.

[155] Kaize Ding, Jianling Wang, Jundong Li, Kai Shu, Chenghao Liu, and Huan Liu. Graph prototypical networks for few-shot learning on attributed networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 295–304, 2020.

[156] Min Shi, Yufei Tang, Xingquan Zhu, David Wilson, and Jianxun Liu. Multi-class imbalanced graph convolutional network learning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, 2020.

[157] Justin M Johnson and Taghi M Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):1–54, 2019.

[158] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems 29 (NeurIPS)*, pages 3630–3638, 2016.

[159] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[160] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, 2020.

[161] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Special issue on learning from imbalanced data sets. *ACM SIGKDD explorations newsletter*, 6(1):1–6, 2004.

[162] Nguyen Thai-Nghe, Zeno Gantner, and Lars Schmidt-Thieme. Cost-sensitive learning methods for imbalanced data. In *The 2010 International joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2010.

[163] Jinhao Dong and Tong Lin. Margingan: Adversarial training in semi-supervised learning. 2019.

[164] Zheng Wang, Xiaojun Ye, Chaokun Wang, Jian Cui, and Philip Yu. Network embedding with completely-imbalanced labels. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[165] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[166] Haibo He and Yunqian Ma. Imbalanced learning: foundations, algorithms, and applications. 2013.

[167] Bartosz Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232, 2016.

[168] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems 30 (NeurIPS)*, pages 4077–4087, 2017.

[169] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

[170] Kelsey Allen, Evan Shelhamer, Hanul Shin, and Joshua Tenenbaum. Infinite mixture prototypes for few-shot learning. In *International Conference on Machine Learning*, pages 232–241. PMLR, 2019.

[171] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer, 2001.

[172] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

[173] Deli Chen, Yanyai Lin, Lei Li, Xuancheng Ren Li, Jie Zhou, Xu Sun, et al. Distance-wise graph contrastive learning. *arXiv preprint arXiv:2012.07437*, 2020.

[174] Yaochen Xie, Zhao Xu, Jingtun Zhang, Zhengyang Wang, and Shuiwang Ji. Self-supervised learning of graph neural networks: A unified review. *arXiv preprint arXiv:2102.10757*, 2021.

[175] Yixin Liu, Shirui Pan, Ming Jin, Chuan Zhou, Feng Xia, and Philip S Yu. Graph self-supervised learning: A survey. *arXiv preprint arXiv:2103.00111*, 2021.

[176] Wei Liu, Junfeng He, and Shih-Fu Chang. Large graph construction for scalable semi-supervised learning. In *ICML*, 2010.

[177] Lingfei Wu, Ian En-Hsu Yen, Zhen Zhang, Kun Xu, Liang Zhao, Xi Peng, Yinglong Xia, and Charu Aggarwal. Scalable global alignment graph kernel using random features: From node embedding to graph embedding. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1418–1428, 2019.

[178] Yu Chen, Lingfei Wu, and Mohammed Zaki. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *Advances in Neural Information Processing Systems*, 33, 2020.

[179] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[180] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

[181] Bo Yuan and Xiaoli Ma. Sampling+ reweighting: Boosting the performance of adaboost on imbalanced datasets. In *The 2012 international joint conference on neural networks (IJCNN)*, pages 1–6. IEEE, 2012.

[182] Yang Gao, Yi-Fan Li, Yu Lin, Charu Aggarwal, and Latifur Khan. Setconv: A new approach for learning from imbalanced data. *arXiv preprint arXiv:2104.06313*, 2021.

[183] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *ICLR*, 2020.

[184] Zixing Song, Xiangli Yang, Zenglin Xu, and Irwin King. Graph-based semi-supervised learning: A comprehensive review. *arXiv:2102.13303*, 2021.

[185] Joffrey L Leevy, Taghi M Khoshgoftaar, Richard A Bauder, and Naeem Seliya. A survey on addressing high-class imbalance in big data. *Journal of Big Data*, 5(1):1–30, 2018.

[186] Jin-Zhu Yu, Mincheng Wu, Gisela Bichler, Felipe Aros-Vera, and Jianxi Gao. Reconstructing sparse illicit supply networks: A case study of multiplex drug trafficking networks. *arXiv preprint arXiv:2208.01739*, 2022.

[187] Gabriel Idakwo, Sundar Thangapandian, Joseph Luttrell, Yan Li, Nan Wang, Zhaoxian Zhou, Huixiao Hong, Bei Yang, Chaoyang Zhang, and Ping Gong. Structure–activity relationship-based chemical classification of highly imbalanced tox21 datasets. *Journal of cheminformatics*, 12(1):1–19, 2020.

[188] Yunchao Liu, Yu Wang, Oanh T Vu, Rocco Moretti, Bobby Bodenheimer, Jens Meiler, and Tyler Derr. Interpretable chirality-aware graph neural network for quantitative structure activity relationship modeling in drug discovery. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

[189] Tong Zhao, Tianwen Jiang, Neil Shah, and Meng Jiang. A synergistic approach for graph anomaly detection with pattern mining and feature learning. *IEEE TNNLS*, 2021.

[190] Jason Van Hulse, Taghi M Khoshgoftaar, and Amri Napolitano. Experimental perspectives on learning from imbalanced data. In *ICML*, pages 935–942, 2007.

[191] Liang Qu, Huaisheng Zhu, Ruiqi Zheng, Yuhui Shi, and Hongzhi Yin. Imgagn: Imbalanced network embedding via generative adversarial graph networks. *arXiv:2106.02817*, 2021.

[192] Shirui Pan and Xingquan Zhu. Graph classification with imbalanced class distributions and noise. In *IJCAI*, pages 1586–1592, 2013.

[193] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[194] Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. A survey of data augmentation approaches for nlp. *arXiv:2105.03075*, 2021.

[195] Tong Zhao, Gang Liu, Stephan Günnemann, and Meng Jiang. Graph data augmentation for graph machine learning: A survey. *arXiv preprint arXiv:2202.08871*, 2022.

[196] Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. Data augmentation for deep graph learning: A survey. *arXiv preprint arXiv:2202.08235*, 2022.

[197] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *NeurIPS*, 2020.

[198] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. *arXiv preprint arXiv:1905.02249*, 2019.

[199] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[200] Shuangjia Zheng, Zengrong Lei, Haitao Ai, Hongming Chen, Daiguo Deng, and Yuedong Yang. Deep scaffold hopping with multimodal transformer neural networks. *Journal of cheminformatics*, 13(1):1–15, 2021.

[201] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

[202] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *KDD*, pages 1365–1374, 2015.

[203] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *ICLR 2020*.

[204] Hongwei Wang and Jure Leskovec. Unifying graph convolutional neural networks and label propagation. *arXiv preprint arXiv:2002.06755*, 2020.

[205] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[206] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. *Advances in neural information processing systems*, 17, 2004.

[207] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *ICDM*, 2005.

[208] Jonatan Kilhamn. Fast shortest-path kernel computations using aproximate methods. Master's thesis, 2015.

[209] Hannu Toivonen, Ashwin Srinivasan, Ross D King, Stefan Kramer, and Christoph Helma. Statistical evaluation of the predictive toxicology challenge 2000–2001. *Bioinformatics*, 19(10):1183–1193, 2003.

[210] Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *KAIS*, 14(3):347–375, 2008.

[211] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *J. Med. Chem.*, 34(2):786–797, 1991.

[212] Jeffrey J Sutherland, Lee A O'brien, and Donald F Weaver. Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *J Chem Inform Comput Sci*, 43(6), 2003.

[213] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *Icml*. Citeseer, 1997.

[214] Asiri Wijesinghe and Qing Wang. A new perspective on "how graph neural networks go beyond weisfeiler-lehman?". In *ICLR*, 2022.

[215] Enyan Dai and Suhang Wang. Say no to the discrimination: Learning fair graph neural networks with limited sensitive attribute information. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021.

[216] Valentina Shumovskaia, Kirill Fedyanin, Ivan Sukharev, and Dmitry Berestnev. Linking bank clients using graph neural networks powered by rich transactional data. *International Journal of Data Science and Analytics*, 2021.

[217] Bingbing Xu, Huawei Shen, Bingjie Sun, Rong An, Qi Cao, and Xueqi Cheng. Towards consumer loan fraud detection: Graph neural networks with role-constrained conditional random field. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[218] Chirag Agarwal, Himabindu Lakkaraju, and Marinka Zitnik. Towards a unified framework for fair and stable graph representation learning. In *Uncertainty in Artificial Intelligence*, pages 2114–2124. PMLR, 2021.

[219] Avishek Bose and William Hamilton. Compositional fairness constraints for graph embeddings. In *International Conference on Machine Learning*, pages 715–724, 2019.

[220] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54(6):1–35, 2021.

[221] Mengnan Du, Fan Yang, Na Zou, and Xia Hu. Fairness in deep learning: A computational perspective. *IEEE Intelligent Systems*, 2020.

[222] Yushun Dong, Ninghao Liu, Brian Jalaian, and Jundong Li. Edits: Modeling and mitigating data bias for graph neural networks. In *Proceedings of the ACM Web Conference 2022*, pages 1259–1269, 2022.

[223] Öykü Deniz Köse and Yanning Shen. Fairness-aware node representation learning. *arXiv preprint arXiv:2106.05391*, 2021.

[224] Yushun Dong, Song Wang, Yu Wang, Tyler Derr, and Jundong Li. On structural explanation of bias in graph neural networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2022.

[225] Wei Jin, Xiaorui Liu, Yao Ma, Charu Aggarwal, and Jiliang Tang. Feature overcorrelation in deep graph neural networks: A new perspective. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2022.

[226] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. Graph condensation for graph neural networks. In *7th International Conference on Learning Representations, ICLR*, 2021.

[227] Yushun Dong, Jian Kang, Hanghang Tong, and Jundong Li. Individual fairness for graph neural networks: A ranking based approach. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021.

[228] Tianxiang Zhao, Enyan Dai, Kai Shu, and Suhang Wang. Towards fair classifiers without sensitive attributes: Exploring biases in related features. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, 2022.

[229] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

[230] Toshihiro Kamishima, Shotaro Akaho, and Jun Sakuma. Fairness-aware learning through regularization approach. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 643–650. IEEE, 2011.

[231] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[232] Xiaoyang Wang, Yao Ma, Yiqi Wang, Wei Jin, Xin Wang, Jiliang Tang, Caiyan Jia, and Jian Yu. Traffic flow prediction via spatial temporal graph neural network. In *Proceedings of The Web Conference 2020*, pages 1082–1092, 2020.

[233] Yu Wang, Yuying Zhao, Yi Zhang, and Tyler Derr. Collaboration-aware graph convolutional networks for recommendation systems. *arXiv preprint arXiv:2207.06221*, 2022.

[234] Yushun Dong, Jing Ma, Chen Chen, and Jundong Li. Fairness in graph mining: A survey. *arXiv preprint arXiv:2204.09888*, 2022.

[235] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

[236] Akari Asai, Kazuma Hashimoto, Hannaneh Hajishirzi, Richard Socher, and Caiming Xiong. Learning to retrieve reasoning paths over wikipedia graph for question answering. *arXiv preprint arXiv:1911.10470*, 2019.

[237] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

[238] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*, 2018.

[239] Rami Aly, Zhijiang Guo, Michael Schlichtkrull, James Thorne, Andreas Vlachos, Christos Christodoulopoulos, Oana Cocarascu, and Arpit Mittal. Feverous: Fact extraction and verification over unstructured and structured information. *arXiv preprint arXiv:2106.05707*, 2021.

[240] Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. Is chatgpt a general-purpose natural language processing task solver? *arXiv preprint arXiv:2302.06476*, 2023.

[241] Girolamo Tessuto. Legal problem question answer genre across jurisdictions and cultures. *English for Specific Purposes*, 30(4):298–309, 2011.

[242] Mark Bolino, David Long, and William Turnley. Impression management in organizations: Critical questions, answers, and areas for future research. *Annual Review of Organizational Psychology and Organizational Behavior*, 3:377–406, 2016.

[243] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*, 2020.

[244] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

[245] Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. Docvqa: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 2200–2209, 2021.

[246] Yang Xu, Yiheng Xu, Tengchao Lv, Lei Cui, Furu Wei, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, Wanxiang Che, et al. Layoutlmv2: Multi-modal pre-training for visually-rich document understanding. *arXiv preprint arXiv:2012.14740*, 2020.

[247] Mingxuan Ju, Wenhao Yu, Tong Zhao, Chuxu Zhang, and Yanfang Ye. Grape: Knowledge graph enhanced passage reader for open-domain question answering. *arXiv preprint arXiv:2210.02933*, 2022.

[248] Fengbin Zhu, Wenqiang Lei, Chao Wang, Jianming Zheng, Soujanya Poria, and Tat-Seng Chua. Retrieving and reading: A comprehensive survey on open-domain question answering. *arXiv preprint arXiv:2101.00774*, 2021.

[249] Jayr Pereira, Robson Fidalgo, Roberto Lotufo, and Rodrigo Nogueira. Visconde: Multi-document qa with gpt-3 and neural reranking. In *European Conference on Information Retrieval*, pages 534–543. Springer, 2023.

[250] Avi Caciularu, Matthew E Peters, Jacob Goldberger, Ido Dagan, and Arman Cohan. Peek across: Improving multi-document modeling via cross-document question-answering. *arXiv preprint arXiv:2305.15387*, 2023.

[251] Wenhan Xiong, Xiang Lorraine Li, Srini Iyer, Jingfei Du, Patrick Lewis, William Yang Wang, Yashar Mehdad, Wen-tau Yih, Sebastian Riedel, Douwe Kiela, et al. Answering complex open-domain questions with multi-hop dense retrieval. *arXiv preprint arXiv:2009.12756*, 2020.

[252] Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, Nitish Shirish Keskar, and Caiming Xiong. Modeling multi-hop question answering as single sequence prediction. *arXiv preprint arXiv:2205.09226*, 2022.

[253] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509*, 2022.

[254] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[255] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

[256] Yao Yao, Zuchao Li, and Hai Zhao. Beyond chain-of-thought, effective graph-of-thought reasoning in large language models. *arXiv preprint arXiv:2305.16582*, 2023.

[257] Hariom A Pandya and Brijesh S Bhatt. Question answering survey: Directions, challenges, datasets, evaluation matrices. *arXiv preprint arXiv:2112.03572*, 2021.

[258] Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. Rider: Reader-guided passage reranking for open-domain question answering. *arXiv preprint arXiv:2101.00294*, 2021.

[259] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[260] Kyosuke Nishida, Itsumi Saito, Atsushi Otsuka, Hisako Asano, and Junji Tomita. Retrieve-and-read: Multi-task learning of information retrieval and reading comprehension. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 647–656, 2018.

[261] Jon Saad-Falcon, Joe Barrow, Alexa Siu, Ani Nenkova, Ryan A Rossi, and Franck Dernoncourt. Pdftriage: Question answering over long, structured documents. *arXiv preprint arXiv:2309.08872*, 2023.

[262] Devendra Singh, Siva Reddy, Will Hamilton, Chris Dyer, and Dani Yogatama. End-to-end training of multi-document reader and retriever for open-domain question answering. *Advances in Neural Information Processing Systems*, 34:25968–25981, 2021.

[263] Valeriia Bolotova-Baranova, Vladislav Blinov, Sofya Filippova, Falk Scholer, and Mark Sanderson. Wikihowqa: A comprehensive benchmark for multi-document non-factoid question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5291–5314, 2023.

[264] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.

[265] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964*, 2020.

[266] Yue Yu, Yuchen Zhuang, Jieyu Zhang, Yu Meng, Alexander Ratner, Ranjay Krishna, Jiaming Shen, and Chao Zhang. Large language model as attributed training data generator: A tale of diversity and bias. *arXiv preprint arXiv:2306.15895*, 2023.

[267] Xuansheng Wu, Kaixiong Zhou, Mingchen Sun, Xin Wang, and Ninghao Liu. A survey of graph prompting methods: techniques, applications, and challenges. *arXiv preprint arXiv:2303.07275*, 2023.

[268] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *arXiv preprint arXiv:2304.13712*, 2023.

[269] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, 2015.

[270] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial intelligence*, 194:28–61, 2013.

[271] Xiaofeng Huang, Jixin Zhang, Zisang Xu, Lu Ou, and Jianbin Tong. A knowledge graph based question answering method for medical domain. *PeerJ Computer Science*, 7:e667, 2021.

[272] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, pages 29–48. Citeseer, 2003.

[273] Sewon Min, Danqi Chen, Luke Zettlemoyer, and Hannaneh Hajishirzi. Knowledge guided text retrieval and reading for open domain question answering. *arXiv preprint arXiv:1911.03868*, 2019.

[274] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[275] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[276] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.

[277] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

[278] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

[279] James Ferguson, Matt Gardner, Hannaneh Hajishirzi, Tushar Khot, and Pradeep Dasigi. Iirc: A dataset of incomplete information reading comprehension questions. *arXiv preprint arXiv:2011.07127*, 2020.

[280] Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. Generate rather than retrieve: Large language models are strong context generators. *arXiv preprint arXiv:2209.10063*, 2022.

[281] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. Gpteval: Nlg evaluation using gpt-4 with better human alignment. *arXiv preprint arXiv:2303.16634*, 2023.

[282] Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. *arXiv preprint arXiv:2305.14387*, 2023.

[283] Yanzhe Zhang, Ruiyi Zhang, Jiuxiang Gu, Yufan Zhou, Nedim Lipka, Diyi Yang, and Tong Sun. Llavar: Enhanced visual instruction tuning for text-rich image understanding. *arXiv preprint arXiv:2306.17107*, 2023.

[284] Yichen Jiang and Mohit Bansal. Avoiding reasoning shortcuts: Adversarial evaluation, training, and model development for multi-hop qa. *arXiv preprint arXiv:1906.07132*, 2019.

[285] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[286] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[287] Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Ng, Ricky Wang, and Aditya Ramesh. Video generation models as world simulators. 2024.

[288] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[289] Daniel Beard. *Firefly: web-based interactive tool for the visualization and validation of image processing algorithms*. PhD thesis, University of Missouri–Columbia, 2009.

[290] Gowthami Somepalli, Vasu Singla, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Diffusion art or digital forgery? investigating data replication in diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6048–6058, 2023.

[291] Zhanjie Zhang, Quanwei Zhang, Wei Xing, Guangyuan Li, Lei Zhao, Jiakai Sun, Zehua Lan, Junsheng Luan, Yiling Huang, and Huaizhong Lin. Artbank: Artistic style transfer with pre-trained diffusion model and implicit style prompt bank. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 7396–7404, 2024.

[292] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[293] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[294] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[295] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[296] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[297] Emiel Hoogeboom, Vıctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *International conference on machine learning*, pages 8867–8887. PMLR, 2022.

[298] Ilia Igashov, Hannes Stärk, Clément Vignac, Arne Schneuing, Victor Garcia Satorras, Pascal Frossard, Max Welling, Michael Bronstein, and Bruno Correia. Equivariant 3d-conditional diffusion model for molecular linker design. *Nature Machine Intelligence*, pages 1–11, 2024.

[299] Gang Liu, Eric Inae, Tong Zhao, Jiaxin Xu, Tengfei Luo, and Meng Jiang. Data-centric learning from unlabeled graphs with diffusion model. *Advances in neural information processing systems*, 36, 2024.

[300] Dhruv Menon and Raghavan Ranganathan. A generative approach to materials discovery, design, and optimization. *ACS omega*, 7(30):25958–25973, 2022.

[301] Dmitrii Gavrilev and Evgeny Burnaev. Anomaly detection in networks via score-based generative models. *arXiv preprint arXiv:2306.15324*, 2023.

[302] Kay Liu, Huijun Lona Yu, Yao Yan, Ziqing Hu, Pankaj Rajak, Amila Weerasinghe, Olcay Boz, Deepayan Chakrabarti, and Fei Wang. Graph diffusion models for anomaly detection. 2024.

[303] Faezeh Faez, Yassaman Ommi, Mahdieh Soleymani Baghshah, and Hamid R Rabiee. Deep graph generators: A survey. *IEEE Access*, 9:106675–106702, 2021.

[304] Xiaojie Guo and Liang Zhao. A systematic survey on deep generative models for graph generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):5370–5390, 2022.

[305] Yanqiao Zhu, Yuanqi Du, Yinkai Wang, Yichen Xu, Jieyu Zhang, Qiang Liu, and Shu Wu. A survey on deep graph generation: Methods and applications. In *Learning on Graphs Conference*, pages 47–1. PMLR, 2022.

[306] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pages 5708–5717. PMLR, 2018.

[307] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*, pages 412–422. Springer, 2018.

[308] Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. In *The Eleventh International Conference on Learning Representations*, 2023.

[309] Ryan A Rossi and Nesreen K Ahmed. Complex networks are structurally distinguishable by domain. *Social Network Analysis and Mining*, 9:1–13, 2019.

[310] Pratha Sah, José David Méndez, and Shweta Bansal. A multi-species repository of social networks. *Scientific data*, 6(1):44, 2019.

[311] Pratha Sah, Kenneth E Nussear, Todd C Esque, Christina M Aiello, Peter J Hudson, and Shweta Bansal. Inferring social structure and its drivers from refuge use in the desert tortoise, a relatively solitary species. *Behavioral Ecology and Sociobiology*, 70:1277–1289, 2016.

[312] Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

[313] Ryan A Rossi and Nesreen K Ahmed. An interactive data repository with visual analytics. *ACM SIGKDD Explorations Newsletter*, 17(2):37–41, 2016.

[314] Haitao Mao, Zhikai Chen, Wenzhuo Tang, Jianan Zhao, Yao Ma, Tong Zhao, Neil Shah, Michael Galkin, and Jiliang Tang. Graph foundation models. *arXiv preprint arXiv:2402.02216*, 2024.

[315] Xuelong Dai, Kaisheng Liang, and Bin Xiao. Advdiff: Generating unrestricted adversarial examples using diffusion models. *arXiv preprint arXiv:2307.12499*, 2023.

[316] Victor Livernoche, Vineet Jain, Yashar Hezaveh, and Siamak Ravanbakhsh. On diffusion modeling for anomaly detection. *arXiv preprint arXiv:2305.18593*, 2023.

[317] Hanqun Cao, Cheng Tan, Zhangyang Gao, Yilun Xu, Guangyong Chen, Pheng-Ann Heng, and Stan Z Li. A survey on generative diffusion models. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[318] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, et al. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *arXiv preprint arXiv:2302.09419*, 2023.

[319] Yijun Tian, Huan Song, Zichen Wang, Haozhu Wang, Ziqing Hu, Fang Wang, Nitesh V Chawla, and Panpan Xu. Graph neural prompting with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19080–19088, 2024.

[320] Jianfeng Wang, Zhengyuan Yang, Xiaowei Hu, Linjie Li, Kevin Lin, Zhe Gan, Zicheng Liu, Ce Liu, and Lijuan Wang. Git: A generative image-to-text transformer for vision and language. *arXiv preprint arXiv:2205.14100*, 2022.

[321] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024.

[322] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023.

[323] Davide Bacciu, Alessio Micheli, and Marco Podda. Edge-based sequential graph generation with recurrent neural networks. *Neurocomputing*, 416:177–189, 2020.

[324] Mariya Popova, Mykhailo Shvets, Junier Oliva, and Olexandr Isayev. Molecularrnn: Generating realistic molecular graphs with optimized properties. *arXiv preprint arXiv:1905.13372*, 2019.

[325] Yuanqi Du, Xiaojie Guo, Amarda Shehu, and Liang Zhao. Interpretable molecular graph generation via monotonic constraints. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pages 73–81. SIAM, 2022.

[326] Yuanqi Du, Yinkai Wang, Fardina Alam, Yuanjie Lu, Xiaojie Guo, Liang Zhao, and Amarda Shehu. Deep latent-variable models for controllable molecule generation. In *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 372–375. IEEE, 2021.

[327] Xiaojie Guo, Yuanqi Du, and Liang Zhao. Property controllable variational autoencoder via invertible mutual dependence. In *International Conference on Learning Representations*, 2020.

[328] Xiaojie Guo, Yuanqi Du, Sivani Tadepalli, Liang Zhao, and Amarda Shehu. Generating tertiary protein structures via interpretable graph variational autoencoders. *Bioinformatics Advances*, 1(1):vbab036, 2021.

[329] Łukasz Maziarka, Agnieszka Pocha, Jan Kaczmarczyk, Krzysztof Rataj, Tomasz Danel, and Michał Warchoł. Mol-cyclegan: a generative model for molecular optimization. *Journal of Cheminformatics*, 12(1):2, 2020.

[330] Shuangfei Fan and Bert Huang. Conditional labeled graph generation with gans. In *Proc. ICLR Workshop Represent. Learn. Graphs Manifolds*, 2019.

[331] Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. Graphnvp: An invertible flow model for generating molecular graphs. *arXiv preprint arXiv:1905.11600*, 2019.

[332] Chengxi Zang and Fei Wang. Moflow: an invertible flow model for generating molecular graphs. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 617–626, 2020.

[333] Youzhi Luo, Keqiang Yan, and Shuiwang Ji. Graphdf: A discrete flow model for molecular graph generation. In *International conference on machine learning*, pages 7192–7203. PMLR, 2021.

[334] Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *International Conference on Machine Learning*, pages 10362–10383. PMLR, 2022.

[335] Meng Liu, Keqiang Yan, Bora Oztekin, and Shuiwang Ji. Graphebm: Molecular graph generation with energy-based models. *arXiv preprint arXiv:2102.00546*, 2021.

[336] Zewen Liu, Guancheng Wan, B Aditya Prakash, Max SY Lau, and Wei Jin. A review of graph neural networks in epidemic modeling. *arXiv preprint arXiv:2403.19852*, 2024.

[337] Mintong Kang, Dawn Song, and Bo Li. Diffattack: Evasion attacks against diffusion-based adversarial purification. *Advances in Neural Information Processing Systems*, 36, 2024.

[338] Anna Goldenberg, Alice X Zheng, Stephen E Fienberg, Edoardo M Airoldi, et al. A survey of statistical network models. *Foundations and Trends® in Machine Learning*, 2(2):129–233, 2010.

[339] Eric D Kolaczyk and Gábor Csárdi. *Statistical analysis of network data with R*, volume 65. Springer, 2014.

[340] Clement Lee and Darren J Wilkinson. A review of stochastic block models and extensions for graph clustering. *Applied Network Science*, 4(1):1–50, 2019.

[341] Mark EJ Newman. Models of the small world. *Journal of Statistical Physics*, 101:819–841, 2000.

[342] Yuanqi Du, Xiaojie Guo, Amarda Shehu, and Liang Zhao. Interpretable molecule generation via disentanglement learning. In *Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 1–8, 2020.

[343] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021.

[344] Xiaohui Chen, Jiaxing He, Xu Han, and Li-Ping Liu. Efficient and degree-guided graph generation via discrete diffusion modeling. *arXiv preprint arXiv:2305.04111*, 2023.

[345] Sebastian Sosa, David Jacoby, Mathieu Lihoreau, and Cédric Sueur. Animal social networks: Towards an integrative framework embedding social interactions, space and time. *Methods in Ecology and Evolution*, 12(1):4–9, 2021.

[346] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z Sheng, Hui Xiong, and Leman Akoglu. A comprehensive survey on graph anomaly detection with deep learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12012–12038, 2021.

[347] Yanzhe Zhang, Ruiyi Zhang, Jiuxiang Gu, Yufan Zhou, Nedim Lipka, Diyi Yang, and Tong Sun. Enhanced visual instruction tuning for text-rich image understanding. 2023.

[348] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

[349] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[350] Jürgen Bajorath. Integration of virtual and high-throughput screening. *Nature Reviews Drug Discovery*, 1(11):882–894, 2002.

[351] Rajvardhan Oak, Min Du, David Yan, Harshvardhan Takawale, and Idan Amit. Malware detection on highly imbalanced data through sequence modeling. In *Proceedings of the 12th ACM Workshop on artificial intelligence and security*, pages 37–48, 2019.

[352] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: an approach to modeling networks. *Journal of Machine Learning Research*, 11(2), 2010.

[353] Yiming Qin, Huangjie Zheng, Jiangchao Yao, Mingyuan Zhou, and Ya Zhang. Class-balancing diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18434–18443, 2023.

[354] Cunxiang Wang, Xiaoze Liu, Yuanhao Yue, Xiangru Tang, Tianhang Zhang, Cheng Jiayang, Yunzhi Yao, Wenyang Gao, Xuming Hu, Zehan Qi, et al. Survey on factuality in large language models: Knowledge, retrieval and domain-specificity. *arXiv preprint arXiv:2310.07521*, 2023.

[355] Xujiang Zhao, Jiaying Lu, Chengyuan Deng, Can Zheng, Junxiang Wang, Tanmoy Chowdhury, Li Yun, Hejie Cui, Zhang Xuchao, Tianjiao Zhao, et al. Domain specialization as the key to make large language models disruptive: A comprehensive survey. *arXiv preprint arXiv:2305.18703*, 2023.

[356] Rylee Thompson, Boris Knyazev, Elahe Ghalebi, Jungtaek Kim, and Graham W Taylor. On evaluation metrics for graph generative models. *arXiv preprint arXiv:2201.09871*, 2022.