MODELS AND REPRESENTATION LEARNING MECHANISMS FOR GRAPH DATA

by

Susheel Suresh

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Computer Science

West Lafayette, Indiana

December 2022

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Jennifer Neville, Chair

Department of Computer Science

Dr. Bruno Ribeiro

Department of Computer Science

Dr. Pan Li

Department of Computer Science

Dr. Tiark Rompf

Department of Computer Science

Approved by:

Dr. Kihong Park

॥ ಹರಿ ಸರ್ವೋತ್ತಮ ವಾಯು ಜೀವೋತ್ತಮ ॥

To my dear parents, for their constant support
and countless sacrifices.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my advisor Professor Jennifer Neville. Jen patiently listened to all of my naive questions and always provided insightful feedback to all of my half-baked ideas. Jen gave me the opportunity to explore multiple research ideas and was supportive of me to work on problems that I thoroughly enjoyed. During my time in grad school, she imparted several invaluable lessons to remember—right from practical instructions on the scientific method, teaching me the nitty-gritty of conducting research, helping me frame the problem and design apt experiments, teaching me the art of paper writing, making presentations, posters and giving talks. She has also helped me navigate grad school— suggesting courses to take, internships to choose and providing advice and opinion on various academic, industry and life matters. Jen has moulded me to become an independent researcher and I am forever grateful to have had the opportunity to be her student.

Next, I would like to wholeheartedly thank Professor Pan Li. Pan has been a wonderful mentor, guide and collaborator. I have benefited immensely from his enthusiasm to hear my ideas, his insightful comments and feedback, his work ethic and sheer brilliance. I will forever remember the late night brainstorming sessions we used to have and paper writing blitz during conference deadlines !

I would like to thank Professor Bruno Ribeiro, Professor Pan Li and Professor Tiark Rompf for serving on my dissertation committee and providing timely and valuable feedback. I would also like to thank my external collaborators Professor Jianzu Ma and Professor Cong Hao for their support and feedback during the course of the research project. I would also like to thank the Purdue computer science department for giving me the opportunity, support and conducive environment to pursue graduate studies.

I thank Mayank Shrivastava, Danny Godbout and Arko Mukherjee from the co-pilot applied AI team at Microsoft for hosting me for two wonderful summer research internships where I got the chance to collaborate with several excellent researchers and engineers. Through these internships, I developed appreciation for applied machine learning research, and got the opportunity to develop approaches to solve real-world problems. I would also

like to thank Jen for giving me the opportunity to work as a contract researcher in the productivity and intelligence group at Microsoft Research. Working with Karthik Tangirala, Nick Caurvina and Bryan Tower was a great pleasure and I got the amazing opportunity to apply my research skills to industry problems.

I would like to thank my fellow lab mates Jean Lin, Giselle Zeno, Mengyue Hang, Hogun Park, Gui Gomes, Mahak Goindani and Ellen Lai for research discussions and providing constructive comments during practice talks. I greatly enjoyed all the cube-side talks, tea breaks and research discussions with Prasita, Haoteng, Balasubramaniam, Abhishek and Nan Jiang.

I would like to express my appreciation to all the wonderful friends I made in grad school. To Adithya and Ksheera Sagar for being great roommates, I thoroughly enjoyed the fiery debates, life and research discussions, cooking sessions, short and long road trips, movie nights, late night ice cream runs and countless other activities. I cherish our friendship and it was a pleasure to share my grad school life experience with both of you. To Vinith and Karthik, thank you for being great friends and assignment buddies in the initial years of grad school. Thank you to all my friends Raunaq, Imon, Chandan, Kshitij, Palvit, Surya, Sahil, Sagar and Damian. I thoroughly enjoyed your company over the years.

I want to thank Ian, Prateek, Shaurya, Troy, Shao, Paul and Saad for sharing the passion for badminton with me and keeping me sane and fit during grad school. Cheers to all the unforgettable memories playing badminton tournaments, practice sessions, group gym workouts, track and field workouts, late night mahjong and eating good food.

I want to thank all my family members and well wishers for their constant support and encouragement. Special thanks to my uncle Prakash and his family for hosting me in Utah during the pandemic. They were with me throughout grad school and made it less stressful. I thank my amma and aunts who would send me homemade food and Indian snacks from Bangalore during my study in West Lafayette—it surely made my stay at Purdue so much easier and pleasant. Heartfelt thanks to my fiancé Nidhi for having confidence in me during the final leg of this dissertation. I am forever indebted to both of my parents—my dear amma Sheela and dad Suresh for always being there and encouraging me to pursue my dreams. I

would not be where I am today without my parents' prayers and their relentless drive to nurture me, teach me, support me and give me the best possible life experiences.

Lastly, this research work was supported by the National Science Foundation (NSF) under contract numbers CCF-1918483, IIS-1618690 and CCF-0939370.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Graph representation learning (GRL) has been increasing used to model and understand data from a wide variety of complex systems spanning social, technological, bio-chemical and physical domains. GRL consists of two main components (1) a parametrized encoder that provides representations of graph data and (2) a learning process to train the encoder parameters. Designing flexible encoders that capture the underlying invariances and characteristics of graph data are crucial to the success of GRL. On the other hand, the learning process drives the quality of the encoder representations and developing principled learning mechanisms are vital for a number of growing applications in self-supervised, transfer and federated learning settings. To this end, we propose a suite of models and learning algorithms for GRL which form the two main thrusts of this dissertation.

In Thrust I, we propose two novel encoders which build upon on a widely popular GRL encoder class called graph neural networks (GNNs). First, we empirically study the prediction performance of current GNN based encoders when applied to graphs with heterogeneous node mixing patterns using our proposed notion of local assortativity. We find that GNN performance in node prediction tasks strongly correlates with our local assortativity metric—thereby introducing a limit. We propose to transform the input graph into a computation graph with proximity and structural information as distinct types of edges. We then propose a novel GNN based encoder that operates on this computation graph and adaptively chooses between structure and proximity information. Empirically, adopting our transformation and encoder framework leads to improved node classification performance compared to baselines in real-world graphs that exhibit diverse mixing. Secondly, we study the trade-off between expressivity and efficiency of GNNs when applied to temporal graphs for the task of link ranking. We develop an encoder that incorporates a labeling approach designed to allow for efficient inference over the candidate set jointly, while provably boosting expressivity. We also propose to optimize a list-wise loss for improved ranking. With extensive evaluation on real-world temporal graphs, we demonstrate its improved performance and efficiency compared to baselines.

In Thrust II, we propose two principled encoder learning mechanisms for challenging and realistic graph data settings. First, we consider a scenario where only limited or even no labelled data is available for GRL. Recent research has converged on graph contrastive learning (GCL), where GNNs are trained to maximize the correspondence between representations of the same graph in its different augmented forms. However, we find that GNNs trained by traditional GCL often risk capturing redundant graph features and thus may be brittle and provide sub-par performance in downstream tasks. We then propose a novel principle, termed adversarial-GCL (AD-GCL), which enables GNNs to avoid capturing redundant information during the training by optimizing adversarial graph augmentation strategies used in GCL. We pair AD-GCL with theoretical explanations and design a practical instantiation based on trainable edge-dropping graph augmentation. We experimentally validate AD-GCL by comparing with state-of-the-art GCL methods and achieve performance gains in semi-supervised, unsupervised and transfer learning settings using benchmark chemical and biological molecule datasets. Secondly, we consider a scenario where graph data is siloed across clients for GRL. We focus on two unique challenges encountered when applying distributed training to GRL: (i) client task heterogeneity and (ii) label scarcity. We propose a novel learning framework called federated self-supervised graph learning (FedSGL), which first utilizes a self-supervised objective to train GNNs in a federated fashion across clients and then, each client fine-tunes the obtained GNNs based on its local task and available labels. Our framework enables the federated GNN model to extract patterns from the common feature (attribute and graph topology) space without the need of labels or being biased by heterogeneous local tasks. Extensive empirical study of FedSGL on both node and graph classification tasks yields fruitful insights into how the level of feature / task heterogeneity, the adopted federated algorithm and the level of label scarcity affects the clients' performance in their tasks.

1. INTRODUCTION

Data from a wide variety of complex systems spanning social, technological, bio-chemical and physical domains are primarily modelled using *graphs* (or *networks*)[1] that jointly describe a set of entities (*nodes*) and their relationships (*edges*) [1], [2]. This abstract formulation using a graph data object allows one to be extremely flexible in modelling various metadata information like node and edge attributes, relation types and timestamps. When incorporated, it leads to attributed graphs, multi-relational graphs and temporal graphs respectively—just to name a few different graph forms which we will extensively use in this dissertation. Given the rise of large online social networks, billion scale interconnected devices [3] and massive scientific initiatives from modelling biological interactomes [4]–[6] to high energy particle jet clouds in physics [7]–[9], the quantity, quality and diversity of graph data is ever increasing and building machine learning techniques that can reason about graph data is paramount to numerous downstream prediction applications. For example, in social networks, classifying users (nodes) is vital for personalized search and advertising; in e-commerce networks, predicting future links (edges) between users and products is key to providing recommendations [10]–[12]; in protein interactomes, predicting interactions (edges) between proteins is crucial to improve drug-design strategies [13]–[15]; for bio-chemical molecules, predicting molecular properties (subgraphs / whole-graph) is useful for various tagging and screening tasks [16]–[19]. Towards this end, decades of work have been put into building models and frameworks for general machine learning problems in graphs viz., node classification, link prediction and graph classification among others.

Traditional approaches heavily relied on feature engineering to obtain meaningful features at the node level (e.g., degree, centrality and assortativity) [20]; edge level (e.g., common neighbors, Katz index and Adamic-Adar index) [21], [22]; subgraph level (e.g., ego graphs, motifs and graphlets) [23], [24] and graph level (e.g., spectra, diameter, random walks and kernels) [25], [26]. Various supervised and unsupervised tasks were solved using these features as input to off-the-shelf machine learning techniques. While, hand-engineered features are principled in nature and based on well tested observations in network science, they can be

[1]↑the term *graph* and *network* are used interchangeably

inflexible and brittle—they cannot adapt during a learning process and often discard several other important aspects of the graphs that are not covered in their design. Designing these features can also be time consuming and inefficient—calculating various graph statistics often incur quadratic or exponential time.

Over the past decade, the principles of representation learning have in part fueled the meteoric rise of modern deep learning [27], [28] for various applications [29]–[37]. Representation learning prescribes to learn flexible representations of data that capture "useful" features relevant to solving downstream tasks while simultaneously discarding "less relevant" or noisy features [38]. Importantly, the inherent inductive bias of the representation learner, constraints in the learning process and priors on the data itself together drive the quality of the learnt representations. In the context of graphs, approaches in graph representation learning (GRL) that aims to encode graph-structured data into low-dimensional vector representations, have recently shown great potential for many applications in biochemistry, physics and social sciences [9], [14], [39], [40]. Central to GRL is (1) a parameterized encoder function that generates representations of a graph at different levels (node, edge, subgraph) and (2) the learning process which trains the parameters of the encoder.

This dissertation revolves around GRL touching various aspects of its fundamental components with two main thrusts: Thrust I (Chapters 3 and 4) will be devoted to the study of how the characteristics of the underlying graph data affect the current state-of-the-art encoders which motivate us to develop new techniques and algorithms to improve their effectiveness and efficiency. Thrust II (Chapters 5 and 6) focuses on the study of current encoder learning mechanisms under various setting such as semi-supervised, self-supervised, transfer and federated settings. We find that the learned representation quality from current approaches have several deficiencies and we propose principled learning techniques that lead to improvements on various real-world benchmarks.

1.1 Thrust I: Encoder Design Driven by Graph Data Characteristics

Mixing Characteristics. Graph neural networks (GNNs) [41], [42], inheriting the power of neural networks [43], [44] have become the de facto encoder of choice for GRL due to their

useful properties which include parameter sharing, permutation invariance, inductiveness and computational efficiency. In particular, message passing graph neural networks [35], [45]–[47] work by propagating node features across edges which is then followed by aggregation e.g., sum, mean, attention and so on for a number of iterations. The central idea is to utilize the neighbourhood (proximity) information of a node to construct its representation which in turn serves as a useful descriptor for predicting its label. Given the successful adoption of GNNs for various graph related tasks [10], [35], [48]–[50], much effort in the community has been devoted to understanding the nature and working of GNNs. While most prominent studies have looked at them from the standpoint of the combinatorial color refinement algorithm for graph isomorphism testing (Weisfeiler-Lehman tests) [51]–[54] with the goal of analysing and improving their expressive power, there are also some works that use the lens of graph signal processing to understand their working [55]–[59]. The latter works view the GNN's neighborhood convolution as a non-linear local node feature smoothing operation akin to graph filtering and point out that GNNs essentially enforce similarity of representations between adjacent nodes and they behave as "low-pass" filters, filtering high frequency noise components in the convolution step. Because the convolution operations are defined on neighbourhoods, the apparent local nature prohibits the use of far away information in the graph to generate node representations. Further, when multiple such GNN convolution layers are stacked with the hopes of using long-range information, the performance decays due to a so called *over-smoothing problem* [60]–[63]—resulting node representations becoming indistinguishable. We experimentally study this phenomenon in GNNs using the notion of *mixing* from network science. Specifically, mixing is defined to quantify the degree at which similar node attributes/labels connect to in local network regions [64], [65]. For instance, in social networks, people with similar habits and ideals form friendships with each other [22]. In citation networks, papers from a similar area tend to cite each other. These can be recognized as *assortative mixing* nodes. In the opposite, *disassortative mixing* nodes can be found in technological networks where node hierarchy exists, heterosexual dating networks based on gender and ecological food webs where predator and prey tend to be dissimilar. Due to the neighborhood smoothing property of GNNs, we experimentally observe that its prediction performance is strongly bounded by the assortative mixing value of a graph. Moreover,

when applied to real world networks that exhibit heterogeneous or diverse mixing patterns, we observe that the performance of GNNs for node classification tasks is highly correlated with the notion of node level assortativity. We then make a simple but important distinction between the input and the computation graph on which a GNN operates. While conventionally, these two graphs mean the same thing, our previous observations motivates us to distinguish the two and we propose to run GNNs on a computation graph that inherently has high assortativity irrespective of the mixing patterns observed in the input graph. This leads to our graph transformation algorithm which transforms the original input graph into a computation graph using proximity and structural information and we experimentally witness improved semi-supervised node classification performance under diverse mixing when off-the-shelf GNNs are run on our computation graph. We also experimentally show how our computational graph has an enhanced level of assortativity to which we attribute the observed gains.

Temporal Characteristics. Many practical machine learning applications for graphs focus on predicting future interactions among nodes, based on an observed sequence of past interactions, for example, recommendations in heterogeneous information systems [66], [67], social behavior prediction [68], [69], and financial system monitoring [70], [71]. In this case representing the data as a temporal graph [72] and formulating the prediction task as link prediction has proved quite successful. While there has been a great deal of research focused on GRL methods to improve predictive performance on a wide variety of downstream tasks (e.g., [39], [73]), much of this effort has centered on static graphs. In temporal graph representation learning (T-GRL), the main goal is to learn representations to predict how the graphs evolve over time [68], [74], [75]. Many previous T-GRL models (e.g., TGN [75]) inherit the limited expressivity of standard GNNs for link prediction. GNNs such as GCN [45] associate each node with a representation and update such a representation by aggregating the neighbor representations [39]. However, representations that use this aggregation procedure fail to encode the structural information that is important for link prediction [76]–[78]. Since most T-GRL models, including TGN, update node representation in the same manner (i.e. updating representations based on the nodes one has interacted with [66], [74], [75], [79]). They also fail to encode such structural information. Another problem is that to

date, these methods optimize node representations using a pointwise loss function geared to maximize accuracy *independently* over future links (conditioned on the graph of interactions in the past). This objective doesn't reflect the fact that many complex systems have constraints that produce dependencies among future links. For example, a user selects among a set of items to decide what to purchase next, or among a set of users to connect to next in a social network. In this case, the predictions are either presented as a ranked list to the user (ie. thru recommendations) or resources are allocated based on the ranking assuming that actions at the top of the rank are more likely to happen (eg. caching information, matching advertisers). While we can rank predictions from a model that is learned from a pointwise loss, it is likely that representations learned with a *joint* ranking loss will generalize better for ranking tasks. Further, our study shows that even when a rank learning mechanism is employed in current methods, several modelling issues arise due to a trade-off between expressivity in the learned representation and scalability of estimation/inference. We address these issues and propose Temporal Graph network for Ranking (TGRank), which significantly improves performance for link prediction tasks by (i) optimizing a list-wise loss for improved ranking, and (ii) incorporating a labeling approach designed to allow for efficient inference over the candidate set jointly, while provably boosting expressivity. We extensively evaluate TGRank over real networks, where the performance of TGRank outperforms the state-of-the-art baselines in ranking metrics while being more efficient on large networks.

1.2 Thrust II: Learning Mechanisms

Adversarial Graph Contrastive Learning. GNNs have been mostly studied in cases with supervised end-to-end training [45], [52], [53], [77], [80]–[82], where a large number of task-specific labels are needed. However, in many applications, annotating labels of graph data takes a lot of time and resources [83], [84], e.g., identifying pharmacological effect of drug molecule graphs requires living animal experiments [85]. Therefore, recent research efforts are directed towards studying self-supervised learning for GNNs, where only limited or even no labels are needed [84], [86]–[97]. Designing proper self-supervised-learning principles for GNNs is crucial, as they drive what information of graph-structured data will be captured

by GNNs and may heavily impact their performance in downstream tasks. Many previous works adopt the edge-reconstruction principle to match traditional network-embedding requirement [98]–[101], where the edges of the input graph are expected to be reconstructed based on the output of GNNs [46], [86], [91]. Experiments showed that these GNN models learn to over-emphasize node proximity [93] and may lose subtle but crucial structural information, thus failing in many tasks including node-role classification [77], [101]–[103] and graph classification [83]. To avoid the above issue, graph contrastive learning (GCL) has attracted more attention recently [84], [87]–[90], [92], [93], [95]–[97]. GCL leverages the mutual information maximization principle (InfoMax) [104] that aims to maximize the correspondence between the representations of a graph (or a node) in its different augmented forms [84], [87]–[90], [94], [95]. Perfect correspondence indicates that a representation precisely identifies its corresponding graph (or node) and thus the encoding procedure does not decrease the mutual information between them. However, researchers have found that the InfoMax principle may be risky because it may push encoders to capture redundant information that is irrelevant to the downstream tasks: Redundant information suffices to identify each graph to achieve InfoMax, but encoding it yields brittle representations and may severely deteriorate the performance of the encoder in the downstream tasks [105]. This observation reminds us of another principle, termed information bottleneck (IB) [106]–[111]. As opposed to InfoMax, IB asks the encoder to capture the *minimal sufficient* information for the downstream tasks. Specifically, IB minimizes the information from the original data while maximizing the information that is relevant to the downstream tasks. As the redundant information gets removed, the encoder learnt by IB tends to be more robust and transferable. Recently, IB has been applied to GNNs [112], [113]. But IB needs the knowledge of the downstream tasks that may not be available. Hence, a natural question emerges: *When the knowledge of downstream tasks are unavailable, how to train GNNs that may remove redundant information?* Previous works highlight some solutions by designing data augmentation strategies for GCL but those strategies are typically task-related and sub-optimal. They either leverage domain knowledge [87], [89], [95], *e.g.*, node centralities in network science or molecule motifs in bio-chemistry, or depend on extensive evaluation on the downstream tasks, where the best strategy is selected based on validation performance [89], [94]. We approach this question by

proposing a novel principle that pairs GCL with adversarial training, termed *AD-GCL* which enables GNNs to avoid capturing redundant information during the training by optimizing adversarial graph augmentation strategies used in GCL. We pair AD-GCL with theoretical explanations and design a practical instantiation based on trainable edge-dropping graph augmentation. We experimentally validate AD-GCL by comparing with the state-of-the-art GCL methods and achieve performance gains of up-to 14% in unsupervised, 6% in transfer, and 3% in semi-supervised learning settings overall with 18 different benchmark datasets for graph level the tasks of molecule property regression and classification, and social network classification.

Federated Self-Supervised Graph Learning. Federated learning (FL) is a distributed learning paradigm that enables a collection of clients to collaboratively train a machine learning model without the need of sharing their local training data [114]–[116]. FL aims for those clients to benefit from using data from each other without sacrificing data privacy or paying a substantial communication cost. Recently, FL has been applied to graph representation learning (GRL) [117]–[119], specifically, training graph neural networks (GNNs) in federated ways, which has found applications in recommender systems [120], [121], drug design [122], molecule property prediction [123], [124], financial crime detection [125], disease and hospitalization prediction [126], [127], and so on. Albeit promising, applying FL to GRL often encounters unique challenges due to practical graph-structured data allocation. In this work, we focus on the following two types of challenges. *The first challenge is client-task heterogeneity.* Many GRL applications have graph-structured data share some common features across clients while the prediction tasks vary substantially across clients. For example, different pharmaceutical companies may have different design purposes and are therefore to predict different properties (e.g., antibacterial v.s. anesthesia) of their designed drugs, although these drug molecules have already been proved to have valid structures. Different product departments in e-commerce (e.g., Electronics v.s. Office Products) may expect to partition their customers into different interest groups, although their used data such as customers' co-purchasing networks and reviews share many common patterns. This defines a special type of data heterogeneity different from most previous empirical studies in FL with label distribution shift as the application scenario. *The second challenge is label scarcity.* Al-

though graph-structure data is often not hard to get or construct, it often takes a lot of time and resources to annotate the labels [83], [128]. Think about difficulties in obtaining explicit customers feedback in e-commerce and costly in vitro experiments for drug design in the above two scenarios. Typically, annotating labels becomes even more challenging when data gets moved from centralized (non-FL) scenarios to distributed (FL) scenarios. Typically, to handle data heterogeneity, personalized FL methods based on such as meta learning [129], [130], proximal regularization [131] and moreau envelop [132] have been proposed. These methods often have a global model and train local models on clients with some adjustment of the global model. Other personalized methods based on multi-task learning [133], [134] and client clustering [135], [136] do not assume a single global model, while these methods need to optimize clients mixing or clustering parameters. Importantly, all these methods do not by default address the issue of label sparsity per task. To address both challenges simultaneously in GRL, we propose to incorporate self-supervised learning (SSL) into FL. SSL forces a model to extract patterns from the common feature (attributes and graph topology) space without the need of labels or being biased by heterogeneous local tasks. SSL has recently been applied to GRL in data-centralized scenarios when labels are sparse [88], [96], while we argue that SSL has better potential in data-decentralized scenarios due to its great power to deal with the potentially extreme heterogeneity in clients' tasks. As the first work of applying federated SSL to GRL, our main goal is to provide an extensive empirical study on positioning the potential of SSL in this setting. We consider a simple framework and name it Federated Self-supervised Graph Learning (FedSGL): FedSGL first performs SSL of GNNs in a FL fashion over all clients; Then, each client fine-tunes the obtained GNNs based on its local task and labels. We do not consider an end-to-end training framework because in theory many recent works have proved that local fine-tuning can give as good prediction performance as end-to-end personalized FL [137], [138], and in practice, some clients may have new downstream tasks added at a later point, where the central model is not contaminated by previous tasks and can quickly adapt to the new task. With this framework, we study both node classification and graph classification tasks by considering different combinations of (a) non-personalized/personalized FL algorithms with (b) commonly-used SSL frameworks, paired with (c) different local fine-tuning strategies. Our study of FedSGL yields fruitful

insights into how the level of feature heterogeneity, the adopted federated SSL algorithm and the level of label scarcity affects the clients' performance in their tasks. Specifically, (1) we observe a uniform advantage of FedSGL over local non-federated SSL, which means using the features of other clients' both labeled and unlabeled data is always beneficial even if clients hold data for very different prediction tasks. (2) FedSGL has a good chance to outperform supervised FL even if the latter adopts sophisticated personalized algorithms such as FedProx [139] and the state-of-the-art clustering-based FL for GRL [123], and even if a good portion of labels are used in supervision. (3) The benefit of fine-tune mechanism varies according to the tasks and the level of label sparsity. For graph classification tasks, we observe that fine-tuning the model is beneficial even with very sparse local labels. However, for node classification tasks, fine-tuning the model does no good in a sparse label regime. As an extra contribution, to test FedSGL over node classification tasks, we construct and introduce a new dataset with co-purchase networks based on different sales departments in Amazon, which may be beneficial for a broader community.

1.3 Dissertation Organization

The rest of the dissertation is organized as follows,

- Chapter 2 provides an in-dept overview of the tools and techniques used in GRL and introduces various important concepts which are relevant for our study.
- Chapters 3 and 4 are devoted to the study of how the characteristics of the underlying graph data affect the current state-of-the-art encoders. Specifically, Chapter 3 presents techniques to deal with various levels of assortative mixing in networks for a class of encoders called graph neural networks and empirically analysis is conducted on various benchmark datasets. Chapter 4 outlines our method and techniques to deal with temporal graph characteristics, describes the model expressivity and complexity, and presents empirical results for the task of temporal link ranking on real world datasets.

- Chapter 5 focuses on the study of self-supervised graph representation learning, outlines a novel adversarial contrastive learning framework, and presents empirical results on unsupervised, semi-supervised and transfer learning tasks using real world datasets.
- Chapter 6 is devoted to the study of decentralized training mechanisms for graph data. We present a novel framework that can deal with client data heterogeneity and label sparsity. Empirical experiments are presented on both node and graph level tasks using real world datasets.
- Chapter 7 summarizes the contributions, provides concluding remarks and discusses future work.

In all, Chapters 3 and 4 contribute to Thrust I and Chapters 5 and 6 contribute to Thrust II of this dissertation.

2. GRAPH REPRESENTATION LEARNING

In this chapter we introduce some preliminary concepts which will be extensively applied, analyzed and extended in the rest of this dissertation.

2.1 Graphs

An attributed graph $G = (V, E)$ where V is a node set and E is an edge set. G may have node attributes $\{X_v \in \mathbb{R}^F \mid v \in V\}$ and edge attributes $\{X_e \in \mathbb{R}^F \mid e \in E\}$ of dimension F. We denote the set of the neighbors of a node v as $\mathcal{N}_v$.

2.2 Learning Graph Representations

Given a set of graphs G_i, $i = 1, 2, ..., n$, in some universe $\mathcal{G}$, the aim is to learn an encoder $f : \mathcal{G} \to \mathbb{R}^d$, where $f(G_i)$ can be further used in some downstream task. Here, we call $f : \mathcal{G} \to \mathbb{R}^d$ as the *graph encoder*.

We also assume that G_i's are all IID sampled from an unknown distribution $\mathbb{P}_{\mathcal{G}}$ defined over $\mathcal{G}$. In a downstream task, each G_i is associated with a label $y_i \in \mathcal{Y}$. Another model $q : \mathbb{R}^d \to \mathcal{Y}$ will be learnt to predict Y_i based on $q(f(G_i))$. We assume (G_i, Y_i)'s are IID sampled from a distribution $\mathbb{P}_{\mathcal{G}\times\mathcal{Y}} = \mathbb{P}_{\mathcal{Y}|\mathcal{G}}\mathbb{P}_{\mathcal{G}}$, where $\mathbb{P}_{\mathcal{Y}|\mathcal{G}}$ is the conditional distribution of the graph label in the downstream task given the graph.

2.3 Graph Neural Networks (GNNs)

We focus on using GNNs, message passing GNNs in particular [35], as the encoder f. For a graph $G = (V, E)$, every node $v \in V$ will be paired with a node representation h_v initialized as $h_v^{(0)} = X_v$. These representations will be updated by a GNN. During the k^{th} iteration, each $h_v^{(k-1)}$ is updated using v's neighbourhood information expressed as,

$$h_v^{(k)} = \text{UPDATE}^{(k)}\left(h_v^{(k-1)},\ \text{AGGREGATE}^{(k)}\left(\{(h_u^{(k-1)}, X_{uv}) \mid u \in \mathcal{N}_v\}\right)\right) \tag{2.1}$$

where AGGREGATE$(\cdot)$ is a trainable function that maps the set of node representations and edge attributes X_{uv} to an aggregated vector, UPDATE$(\cdot)$ is another trainable function

that maps both v's current representation and the aggregated vector to v's updated representation. After K iterations of Eq. 2.1, the graph representation is obtained by pooling the final set of node representations as,

$$f(G) :\triangleq h_G = \text{POOL}\Big(\{h_v^{(K)} \mid v \in V\}\Big) \tag{2.2}$$

For design choices regarding aggregation, update and pooling functions we refer the reader to [39], [140], [141].

2.4 Weisfeiler-Lehman (WL) Test and Expressive Power of GNNs

Two graphs G_1 and G_2 are called to be isomorphic if there is a mapping between the nodes of the graphs such that their adjacencies are preserved. For a general class of graphs, without the knowledge of the mapping, determining if G_1 and G_2 are indeed isomorphic is challenging and there has been no known polynomial time algorithms until now [142]. The best algorithm till now has complexity $2^{O(\log n)^3}$ where n is the size of the graphs of interest [143].

The family of Weisfeiler-Lehman tests [51] (specifically the 1-WL test) offers a very efficient way perform graph isomorphism testing by generating canonical forms of graphs. Specifically, the 1-WL test follows an iterative color refinement algorithm. Let, graph $G = (V, E)$ and let $C : V \to \mathcal{C}$ denote a coloring function that assigns each vertex $v \in V$ a color C_v. Nodes with different features are associated with different colors. These colors constitute the initial colors C_0 of the algorithm i.e. $C_{0,v} = C_v$ for every vertex $v \in V$. Now, for each vertex v and each iteration i, the algorithm creates a new set of colors from the color $C_{i-1,v}$ and the colors $C_{i-1,u}$ of every vertex u that is adjacent to v. This multi-set of colors is then mapped to a new color (say using a unique hash). Basically, the color refinement follows

$$C_{i,v} \leftarrow \text{Hash}(C_{i-1,v}, \{C_{i-1,u|u\in\mathcal{N}_v}\}), \tag{2.3}$$

where the above Hash function is an injective mapping. This iteration goes on until when the list of colors stabilises, i.e. at some iteration N, no new colors are created. The final set of colors serves as the the canonical form of a graph.

Intuitively, if the canonical forms obtained by 1-WL test for two graphs are different, then the graphs are surely not isomorphic. But, it is possible for two non-isomorphic graphs to share a the same 1-WL canonical form. Though the 1-WL test can test most of the non-isomorphic graphs, it will fail in some corner cases. For example, it cannot distinguish regular graphs with the same node degrees and of the same sizes.

As GNNs share the same iterative procedure as the 1-WL test by comparing Eq. 2.3 and Eq. 2.1, GNNs are proved to be at most as powerful as the 1-WL test to distinguish isomorphic graphs [52], [53].

3. GRAPH REPRESENTATION LEARNING AND LOCAL MIXING PATTERNS

In GNNs, the standard message passing works by propagating node features across edges and followed by aggregation viz. *sum, mean or attention* for a number of rounds [45]–[47]. The central idea is to utilize the neighbourhood information to construct a representation that can be beneficial for downstream learning tasks. Looking through the lens of graph signal processing, this operation of GNN could be viewed as a non-linear form of smoothing operation on the neighborhood or a low-pass graph filtering which is invariant to graph isomorphism. Clearly, one fundamental assumption made here is that similar nodes (w.r.t node attributes and labels) have a higher tendency to connect to each other compared to nodes that are far away. In other words, the philosophy followed is that proximity information from the surroundings of a node is a useful descriptor for predicting its labels. In network science, the concept *assortative mixing* is defined to quantify the degree of similar node attributes/labels aggregated on local network regions [64], [65]. For instance, in social networks, people with similar habits and ideals form friendships with each other [22]. In citation networks, papers from a similar area tend to cite each other. These can be recognized as *assortative mixing* nodes. In the opposite, *disassortative mixing* nodes can be found in technological networks where node hierarchy exists, heterosexual dating networks based on gender and ecological food webs where predator and prey tend to be dissimilar.

In this work, we aim to study the relationships between the limits of prediction performance of GNN and different mixing patterns in a graph. Quantifying mixing patterns in graphs has conventionally been done using a global binary notion of homophily/heterophily or assortative/disassortative mixing. These global summary statistics capture the average mixing patterns in the graph as an entire entity and are meaningful only when the mixing patterns of a whole network are centered around the mean. However, most real world graphs show heterogeneous and diverse mixing patterns wherein certain parts of the graph are assortative while others disassortative [144], [145]. For instance, Figure 3.2 demonstrates the distributions of assortativity on several real world graphs in which both multimodal distributions and long tail distributions are observed. Apparently, the global metrics will fail to

measure this diversity on these complicated real world graphs. Recently, there has been a growing interest in designing new GNN models utilizing the nature of assortativity in graphs [146]–[150]. However, some of the representatives are based on heuristics leveraging node attributes [146] or intermediate node representations [148] to address disassortative nodes in the graph. Others simply incorporate node features from multi-hop neighbors [147] which might help improve predictions of disassortative nodes but at the same time suffer the over-smoothing problem of GNN [56], [57], [60]. GPR-GNN [150] may overcome the above issue using generalized graph diffusion [151] but loses model expressivity [152]. We reason that

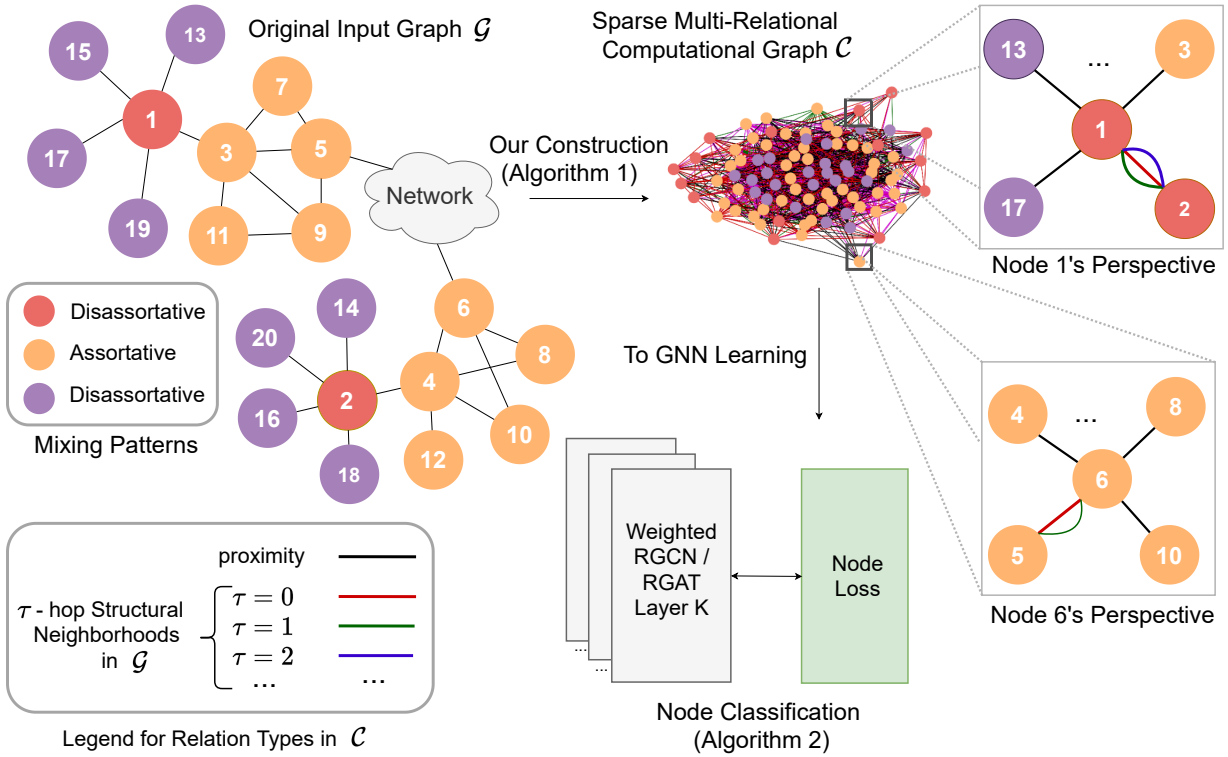

Figure 3.1. An input graph with diverse mixing pattern. Our pipeline uses both proximity and structural information to build a computation graph on which a GNN is run.

for GNNs to achieve good performance on graphs with diverse mixing, one has to provide sufficient inductive bias that lets the model adaptively choose either proximity, structural information or both for predicting node labels. This is based on the key observation that disassortative nodes (potentially far apart) may share similar structural features while assor-

tative nodes tend to share similar features within their proximity. Consider the input graph $\mathcal{G}$ in Figure 3.1 and two nodes colored red i.e. 1 and 2 having same labels as each other but different from the labels of their own neighbourhoods. Based on the theory of mixing patterns, these two nodes are disassortative. Even though they are far apart, their local connecting pattern is quite similar. For instance, by comparing degree sequences of nodes 1 and 2 in $\mathcal{G}$ at various neighbourhoods, we can see that at 0-hop both have similar degree of 5 and 5 respectively, their 1-hop neighbours all have the same degree of 1 or 5 and so on. This shows nodes 1 and 2 are structurally quite similar and therefore, we can make use of their structural equivalences to construct a new graph in which nodes like 1 and 2 have a connection with large weight. On the other hand, consider node like 5 and 6. They mix assortatively and their surroundings/proximity can provide enough information to infer their labels. Further, we could still benefit from the fact that nodes 5 and 6 have similar local structure. In all, our idea is to construct a transformed computation graph that encodes both structure and proximity information w.r.t each node and the GNN is run on this computation graph instead of the original graph. Note that because similar (either structurally or proximity) nodes have large weight in the computation graph it has an enhanced level of assortativity and this can boost the prediction performance of GNNs.

To implement this idea, we first use a local measure of assortativity that can quantify diverse mixing patterns introduced in [144]. This new metric, named *local assortativity*, is a node-centric measure of mixing patterns that calculates assortativity within a local neighbourhood. We show that the representation capability of a wide range of GNN models is highly correlated with the level of local assortative mixing in the graph, which sets a limit to the prediction performance for GNN models based on message passing (Sec. 3.4). To break this limit, we then develop a new algorithm which can transform the input graph into a new one with higher assortativity level and suitable for the deployment of GNN by leveraging both proximity and the local structural similarity of nodes at multiple scales. Figure 3.1 shows the overall framework we propose based on the idea of transforming the input graph to a new computation graph on which the GNN is run. Lastly, we conduct extensive experiments and provide analysis to show the benefits of the proposed approach.

Our code and an easy to use tool for evaluating GNNs w.r.t network local assortativity is provided online [1].

3.1 Preliminaries

A graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in which $\mathcal{V}$ and $\mathcal{E}$ denote the node set and edge set, respectively. An edge going from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$ is denoted as $(u, v) \in \mathcal{E}$. The adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is a convenient way to represent $\mathcal{G}$ where, $\boldsymbol{A}[u, v] = 1$ if $(u, v) \in \mathcal{E}$ otherwise 0. $\boldsymbol{A}$ is a real valued matrix when there are *weighted* edges. For *multi-relational* graphs, we extend the edge notation with type as $(u, v, \tau) \in \mathcal{E}$ to denote that the edge (u, v) belongs to type $\tau \in \mathcal{R}$.

We represent the node *attributes* or *features* as a matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}|\times d}$ where d is the feature dimension. The feature of a particular node $u \in \mathcal{V}$ is a vector $\boldsymbol{x}_u \in \mathbb{R}^d$. We denote the *neighbourhood* around a given node u which is a set of nodes exactly one hop/step away as $\mathcal{N}(u) = \{v : (u, v) \in \mathcal{E}\}$.

We consider the standard semi-supervised node classification task on $\mathcal{G}$, where each node $u \in \mathcal{V}$ has a class label y_u. The goal is to learn a function $f : \mathcal{V} \to \mathcal{Y}$ mapping the set of nodes to their class labels given some labelled nodes $\{(u_1, y_{u_1}), (u_2, y_{u_2}), \dots\}$ as training where $u_i \in \mathcal{V}$ and $y_{u_i} \in \mathcal{Y}$.

3.1.1 Neural Message Passing

It is a framework that encompasses a range of GNN techniques inspired by the classical color refinement algorithm for graph isomorphism testing [35], [153]. During this refinement process, vector messages are passed between nodes across edges and updated using neural networks repeatedly for K rounds. The parameters are learned by defining a suitable loss function and followed by back propagation. Concretely, during the k^{th} iteration a *hidden*

[1] ↑https://github.com/susheels/gnns-and-local-assortativity

representation $\boldsymbol{h}_u^{(k-1)}$ corresponding to each node $u \in \mathcal{V}$ is updated using u's neighbourhood information. Expressed as,

$$\boldsymbol{m}_{\mathcal{N}(u)}^k = \text{AGGREGATE}^k\Big(\{\boldsymbol{h}_v^{(k-1)} : v \in \mathcal{N}(u)\}\Big) \tag{3.1}$$

$$\boldsymbol{h}_u^{(k)} = \text{UPDATE}^k\Big(\boldsymbol{h}_u^{(k-1)},\ \boldsymbol{m}_{\mathcal{N}(u)}^k\Big) \tag{3.2}$$

where AGGREGATE$(\cdot)$ is a trainable differentiable function mapping sets of hidden node representations of u's neighbours to an aggregated message vector, UPDATE$(\cdot)$ is also a trainable differentiable function that maps both u's current hidden representation and the aggregated message vector to u's updated representation. The initial representation $\boldsymbol{h}_u^{(0)}$ is initialized using the original node feature $\boldsymbol{x}_u$. After a total of K iterations following Eq. 3.1 and 3.2, we obtain final representations $\boldsymbol{z}_u = \boldsymbol{h}_u^K, \forall u \in \mathcal{V}$. To perform node classification, we derive the class label of a node u by decoding it's final representation via, $y_u = \text{argmax}(\text{softmax}(\text{MLP}_\theta(\boldsymbol{z}_u)))$ where MLP is a neural network with trainable parameters θ and the softmax function is used to get a probability distribution over the classes.

3.1.2 Mixing in Networks

The global *assortativity coefficient* r_{global} introduced by Newman [65] is used to measure mixing in networks which is the tendency of nodes with similar attributes/labels to be connected to other nodes. To characterize the mixing pattern, a quantity M_{gh} is defined to be the fraction of edges in a network that connect a node with label g to one of label h. This helps us define a *mixing matrix* $\boldsymbol{M}$ whose elements are M_{gh}. This matrix satisfies the following sum property $\sum_g \sum_h M_{gh} = 1$. Global assortativity is a summary statistic for the whole network and is defined as,

$$r_{\text{global}} = \frac{\sum_g M_{gg} - \sum_g a_g b_g}{1 - \sum_g a_g b_g} \tag{3.3}$$

where a_g and b_g represent the number of outgoing and incoming edges of all nodes of label g as follows, $a_g = \sum_h M_{gh}$ and $b_g = \sum_h M_{hg}$. The quantities a and b can be viewed

as marginals that describe the proportion of edges starting from and ending at each of the attributes. For undirected graphs where ends of edges are of same type, quantities a_g and b_g are equal and $\boldsymbol{M}$ is symmetric. This allows us to write the elements of $\boldsymbol{M}$ as,

$$M_{gh} = \frac{1}{2m} \sum_{i:\tau_i=g} \sum_{j:\tau_j=h} A_{ij} \tag{3.4}$$

where A_{ij} is an element of adjacency matrix, $m = |\mathcal{E}|$ is the number of edges and τ_i represents the label of node i.

3.2 Related Work

Graph Neural Networks (GNNs) have been successfully adopted for many graph related tasks [10], [35], [48]–[50] and much effort in the community has been in understanding the nature and working of GNNs either with the lens of signal processing [56]–[59] or the combinatorial color refinement algorithm for graph isomorphism [52]–[54], [77]. Li, Han, and Wu [60] points out that GNNs essentially enforce similarity of representations between adjacent nodes akin to some sort of *local smoothing*. In line with this view, NT and Maehara [56] shows that GNNs behave as "low-pass" filters filtering high frequency noise components in the convolution step. Fu, Hou, Zhang, *et al.* [57] theoretically characterize the behaviour of a number of GNN models by proposing that they work by smoothing and de-noising node features. All these results show that when node features and labels vary smoothly or in other words when there is assortative mixing, GNNs tend to work well.

Because the convolution operations are defined on neighbourhoods, the apparent local nature prohibits the use of higher-order information in the graph. To alleviate this, Li, Han, and Wu [60] tried to stack multiple layers of GNNs but failed due to the over-smoothing problem resulting from node representations becoming indistinguishable. This problem has also been acknowledged in Klicpera, Bojchevski, and Günnemann [61]. Another line of work proposes graph attention [47], [149] computed using node features however, they are still enforcing smoothing albeit adaptively making use of relevant information from a node's surrounding.

In light of these results, a few works propose to supplant the basic message passing framework of GNNs with extra graph information. PPNP [61] uses PageRank, GDC [154] utilizes graph diffusion (e.g.,heat kernels) and Geom-GCN [146] extends graph convolution with geometric aggregation derived by precomputing unsupervised node embeddings. GPR-GNN [150] allows different hop neighbors being associated with different signs of scalar weights to model high pass filters. Jumping Knowledge Networks [155] leverages different neighborhood ranges for each node to enable better structure-aware representations. Non-local GNNs [148] use attention to adaptively get relevant long range graph information while H_2GCN [147] and MixHop [156] directly include information from higher order neighbourhoods within each convolution step. A comprehensive review of various graph neural networks can be found in [140], [141].

3.3 Local Mixing in Graphs

The global assortative coefficient r_{global} defined in Eq. 3.3 captures the average mixing pattern for the whole network but r_{global} is only meaningful if all nodes have mixing concentrated around the mean. It has been studied that real world graphs exhibit high variation in mixing patterns and we are essentially interested in how GNNs perform under such a diverse mixing. For this we first utilize a node level measure of assortativity r_{local} introduced by Peel, Delvenne, and Lambiotte [144] that is calculated w.r.t a local neighbourhood. This allows us to interpolate the mixing from individual nodes to global graph level by varying the size of the local neighbourhoods. Consider a simple random walker on an undirected graph. It walks by selecting an edge at i with an equal probability of A_{ij}/d_i where d_i is the degree of node i. Then, the stationary probability of being at node i is given by $\pi_i = d_i/2m$. This means each edge is traversed with a probability of $\pi_i A_{ij}/d_i = 1/2m$. Based on this and noting that $A_{ij} \in \{0, 1\}$, we can rewrite Eq. 3.4 as,

$$M_{gh} = \sum_{i:\tau_i=g} \sum_{j:\tau_j=h} \pi_i \frac{A_{ij}}{d_i} \tag{3.5}$$

Eq. 3.5 reinterprets the mixing $\boldsymbol{M}$ from the point of view of using random walk to visit the entire graph and thus reveals that the global assortativity counts all edges in the graph equally. For the local measure of assortativity the edges are weighted according to how local they are to a node of interest l by replacing the stationary distribution π_i with an alternative distribution over the nodes $w(i;l)$. Then Eq. 3.5 becomes,

$$M_{gh}(l) = \sum_{i:\tau_i=g} \sum_{j:\tau_j=h} w(i;l) \frac{A_{ij}}{d_i} \tag{3.6}$$

The personalized PageRank vector is utilized as a proxy for $w(i;l)$. Concretely, it is simple random walk with restarts i.e. during a simple random walk, the walker can return to the initial node of interest l with a probability of $(1-\alpha)$. Varying α allows us to interpolate from the trivial local neighbourhood (when $\alpha = 0$ i.e. walker never leaves the node) to the global assortativity (when $\alpha = 1$ i.e. no restarts). Finally, the local assortativity metric for a node l parameterized by α is,

$$r_\alpha(l) = \frac{\sum_g M_{gg}(l) - \sum_g a_g^2}{1 - \sum_g a_g^2} \tag{3.7}$$

Note that we can recover the global assortativity metric from Eq. 3.7, $r_1(l) = r_{\text{global}}$ because when $(\alpha = 1)$ no restarts happen, $w_\alpha(i;l)$ falls back to π_i. When calculating the $r_\alpha(l)$ in practice, instead of choosing α heuristically, inspired by TotalRank [157], the PageRank vector is averaged over the entire range of $\alpha \in [0,1]$ as,

$$w_{\text{tt}}(i;l) = \int_0^1 w_\alpha(i;l)\, d\alpha \tag{3.8}$$

With $w_{tt}(i;l)$ in place of $w(i;l)$ in Eq. 3.6 and finally using the resultant mixing matrix in Eq. 3.7 gives us our local assortativity metric $r_{\text{local}}(l)$.

In Figure 3.2, we examine various networks from different domains for existence of diverse mixing patterns using r_{local}. The details of these networks are given in the supplemental (Sec. 3.7.1). In almost all of them we witness skewed and multimodal distributions. It is interesting to note that the global assortativity coefficient $r_{global} \approx 0$ (Eq. 3.3) while nodes

exhibit diverse mixing over the full spectrum of r_{local}. This observation highlights that r_{global} is not necessarily reliable as it doesn't give a complete picture.

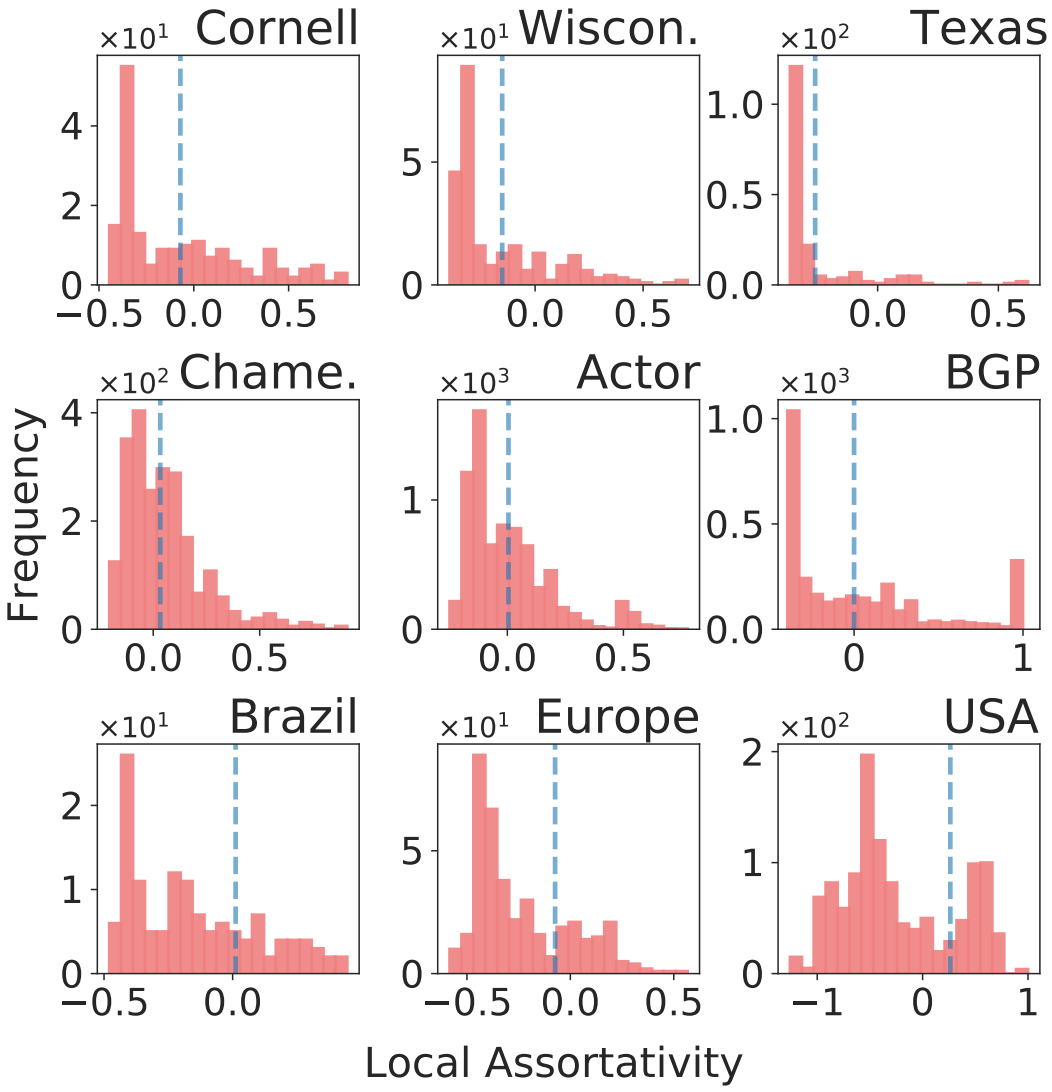

Figure 3.2. Observed distribution of node level assortativity in various graphs. The blue dotted line indicates global assortativity coefficient.

3.4 Problems of GNNs under Diverse Local Mixing

Given the observation in Figure 3.2, we are interested in studying the behaviour of various leading GNN models such as GCN [45], GIN [52] and GAT [47], when applied to graphs with diverse mixing patterns. The task of semi-supervised node classification is used

as a proxy to understand the power of modelling graph data w.r.t different levels of local assortativity embedded in the graph. As shown in Figure 3.3, the performance of GNN models are highly correlated with the node-level local assortativity r_{local} within the same graph they are deployed. Across all the tested real world graphs, another clear pattern is that most of the popular GNN methods perform poorly for disassortative nodes l with $r_{\text{local}}(l) < 0$. The reason is that the features of disassortative nodes are vastly different from their neighbourhoods' and, GNN methods simply cannot create useful node representations based on the information provided by their neighbours through conventional node smoothing operation (alluded in theoretical works [56], [57]). We further characterize our reasoning below with the help of two definitions.

Definition 3.4.1 (Neighbourhood Label Smoothness)**.** *A node $u \in \mathcal{V}$ with class label $y_u \in \mathcal{Y}$ has label smoothness parameter defined on the neighbourhood $\mathcal{N}(u)$ as, $\epsilon_u = \frac{1}{|\mathcal{N}(u)|} \sum_{i \in \mathcal{N}(u)} P(y_i = y_u | y_u)$*

Definition 3.4.2 (Neighbourhood Feature Smoothness)**.** *A node $u \in \mathcal{V}$ with feature vector $\boldsymbol{x}_u \in \mathbb{R}^d$ has a smoothness parameter defined on the neighbourhood $\mathcal{N}(u)$ as, $\lambda_u = \|\boldsymbol{x}_u - \frac{1}{|\mathcal{N}(u)|} \sum_{i \in \mathcal{N}(u)} \boldsymbol{x}_i\|^2$*

Labels of disassortative nodes and their neighbours have high probability of being different which means their labels tend to be not smooth in that neighbourhood, which is quite clear from Def. 3.4.1 (their ϵ_u's tends to be low). Then, it is also likely that features of such nodes are not smooth either (high λ_u). This is based on a reasonable assumption that features $\boldsymbol{x}_u$ and class labels y_u are correlated. However, GNNs try to smooth the node features in the latent space which in turn smooth out the class label predictions. Thus, performing poorly in a disassortative regime. On the flip side for neighbourhoods mixing assortatively, owing to smooth labels all across, ϵ_u will be large, which also means high feature smoothness (low λ_u), a regime that is very beneficial to GNNs. A key take away is that $\epsilon_u \propto 1/\lambda_u$ and this characterization explains the observations we witness in Figure 3.3.

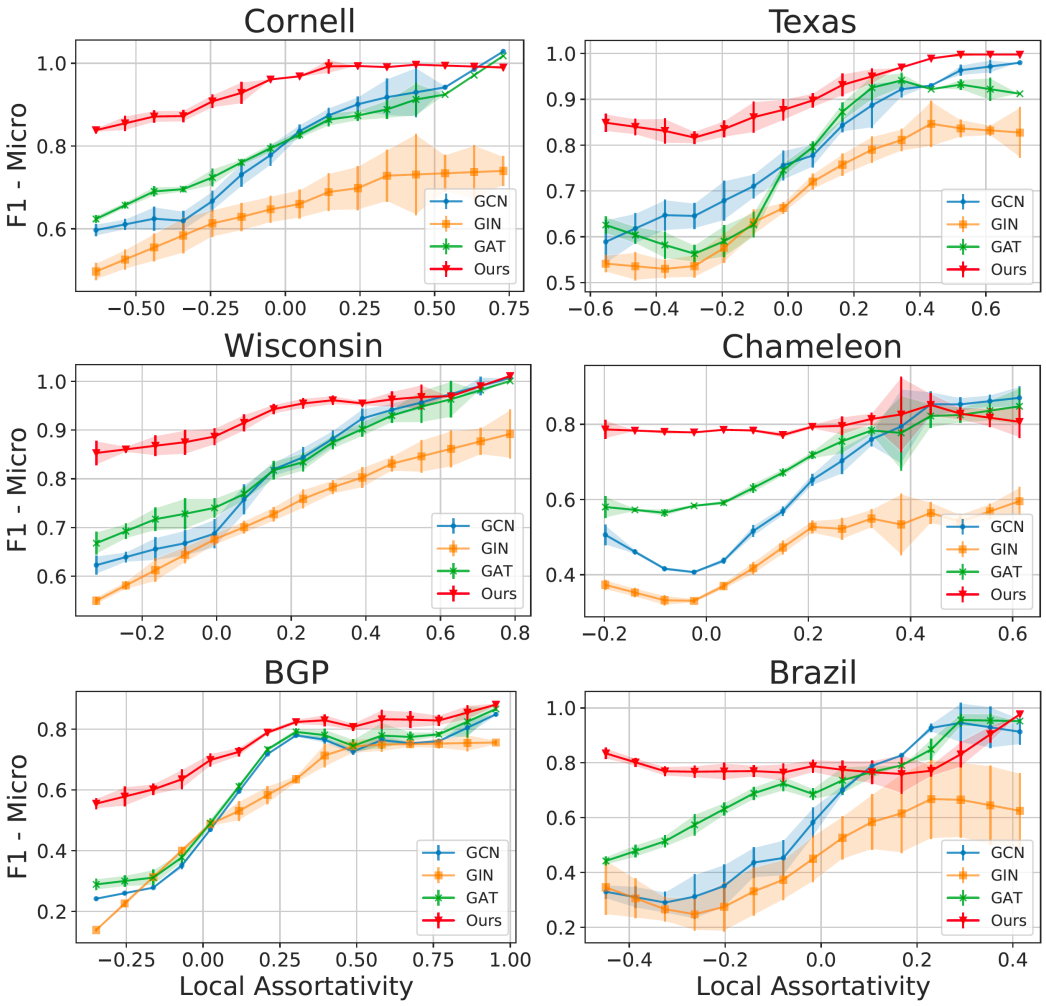

Figure 3.3. GNNs vs Local Assortativity on various graphs. Node classification is performed on various datasets. Y-axis shows mean F1-Micro with standard deviation over 10 runs.

3.5 Decoupling the GNN Computation Graph and the Underlying Original Graph

In Section 3.4 we showed that real world graphs contain diverse mixing patterns and experimental results demonstrate that current GNN methods based on message passing are unable to provide good representations for nodes that show disassortative mixing. To break the limit introduced by the original graph structure, we design a graph transformation algorithm which can generate a new graph with higher assortativity. The central idea is to leverage the structural and proximity information in the original graph. Ideally, the

resulted new graph should have the following properties: (1) Constructed from the original input graph using the same set of nodes without the class label information. (2) Encode the structural equivalences in the original input graph in a model free manner. (3) Encode proximity information as seen in the original graph.

These properties requires the construction of a new graph to be solely based on structural and proximity regularities of the original graph. Traditionally, GNN methods define convolutions on the original input graph and various design choices are made. Different from them, our framework aims to apply such GNN models on a transformed graph which we dub as the "computation graph" for the same machine learning task. Note that our framework consists of 2 stages. Stage (1) transforms the original input graph to a computation graph and in Stage (2) GNN methods are applied on the computation graph for learning. In the next section, we introduce one approach for graph transformation and later define message passing on the transformed graph.

3.6 A Practical Framework for Improving the Performance of GNNs

One obvious choice to encode structural equivalences between nodes is to compare ordered degree sequences at various hierarchies [101]. The rationale is that any two nodes with same degree are structurally similar, and if their one hop neighbours also have same degree, then they are even more structurally similar and so on. Therefore, a key observation is that the structural similarity between two nodes monotonically increases when their degree sequences get progressively similar. More formally, let $\mathcal{N}_\tau(g)$ denote the set of neighbouring nodes at exactly τ hops away from node g in graph $\mathcal{G}$. Let $s(V)$ represent a non-increasing (ordered) sequence of degrees of a set $V \subset \mathcal{V}$ of nodes. The goal is to compare ordered degree sequences at various neighbourhoods for every pair of nodes (g, h) in $\mathcal{G}$. The notion of *structural distance* [101] is recursively defined as follows,

$$f_\tau(g, h) = f_{\tau-1}(g, h) + \mathcal{D}(s(\mathcal{N}_\tau(g)), s(\mathcal{N}_\tau(h))) \tag{3.9}$$

where $\mathcal{D}(S_1, S_2) \geq 0$ measures the distance between ordered degree sequences S_1 and S_2 and $f_{-1}(\cdot) = 0$. $f_\tau(g, h)$ is defined for $\tau \geq 0$ and $|\mathcal{N}_\tau(g)|, |\mathcal{N}_\tau(h)| > 0$ i.e. only when neigh-

bourhoods at τ exist. The cost function $\mathcal{D}(\cdot, \cdot)$ should ideally give small values for similar ordered degree sequences while provide large values for vastly different ones. Following [101], we make use of Fast Dynamic Time Warping (DTW) [158] that is best suited for loosely comparing sequences of different sizes. The recursive definition of structural distance in Eq. 3.9, makes sure that $f_\tau(\cdot, \cdot)$ can only increase as we successively progress through τ. So, for nodes g and h, that are structurally similar, structural distance is low.

3.6.1 Incorporating Structure and Proximity Information in the Computation Graph

We construct a weighted multi-relational computation graph $\mathcal{C}$ which encodes the structural distances at various hierarchies between pairs of nodes and proximity information available in the original input graph $\mathcal{G}$. Let T denote a number less the diameter of the graph $\mathcal{G}$ and $n = |\mathcal{V}|$ the number of nodes. To construct the new graph $\mathcal{C}$, we first add the original node set $\mathcal{V}$ and for each pair of nodes (g, h) we create $T + 1$ different types of edges where each type corresponds to the τ-hop neighbourhoods on which we calculated structural distance in $\mathcal{G}$. To encode structural equivalence between nodes we define edge weights $w_\tau(g, h)$ between g and h with structural relation type τ to vary inversely with structural distance $f_\tau(g, h)$ as follows:

$$w_\tau(g, h) = e^{-f_\tau(g,h)}, \quad \tau = 0, 1, \ldots T \tag{3.10}$$

The edge weight for type τ between g and h is large when their τ-hop neighbors have similar network structure properties (low structural distance).

For proximity information, we simply use the original edges of $\mathcal{G}$ and add it to $\mathcal{C}$ with weight one. In total, this construction creates $\mathcal{C}$ which has $\mathcal{V}$ nodes and at most $\mathcal{E} + (T+1)\binom{n}{2}$ edges. Naively using the above graph construction results in a large number of edges being introduced and when run, requires $O(n^2)$ structural similarity calculations. However in practice we use an heuristic algorithm which achieves $O(n \log n)$ calculations. The intuition is that we don't need to look at node pairs with large degree differences. For instance, given nodes u and v with degree 1 and 20, we don't have to compute the similarity between u

and v as their structural similarity will be extremely small. We cap ourselves to a budget of $O(\log n)$ nodes to look at for each node. The budget is picked based on the heuristic of most similar degrees. Thus, pairwise structural similarity calculations are restricted to $O(\log n)$ for each node at each τ-hop neighbourhood. We thus have $O(n \log n)$ edges for each τ instead of $O(n^2)$. For completeness we provide the efficient practical implementation of the algorithm for constructing the computation graph in Algo. 1

In this work, we provide one specific implementation of our general idea of using both structure and proximity information from the original graph into the computation graph. Other structural techniques viz. RolX [102], GraphWave [103] and generalized proximity inspired methods viz. Graph Diffusion [154], PageRank [61] can also be adopted.

3.6.2 Message Passing on the Multi-relational Computation Graph

The constructed multi-relational graph $\mathcal{C}$ and the original graph $\mathcal{G}$ share the same node set $\mathcal{V}$ and can be defined as $(\mathcal{V}, \mathcal{E}', \mathcal{R})$. Each edge in $\mathcal{C}$ going from node u to v of type τ is represented as a triplet $(u, v, \tau) \in \mathcal{E}'$. Recall that there are $T+1$ structural type edges and one proximity type edge. Thus, the total number of relations $|\mathcal{R}| = T+1+1$. We now define the message passing procedure on $\mathcal{C}$ following the standard GNN formulation (ref. Eq. 3.1, 3.2).

To account for the different relation types in $\mathcal{C}$, following [159], we introduce a relation specific transformation matrix $\boldsymbol{W}_\tau$ for each type $\tau \in \mathcal{R}$ of edge and specify the AGGREGATE function as follows,

$$\boldsymbol{m}_u = \sum_{\tau \in \mathcal{R}} \sum_{v \in \mathcal{N}_1^\tau(u)} \boldsymbol{W}_\tau \, \boldsymbol{h}_v \; w_\tau(u, v) \; \alpha_\tau(u, v) \tag{3.11}$$

where $w_\tau(u, v)$ is the τ type specific edge weight between nodes u and v defined in Algo. 1. In line with our motivation of letting the model adaptively choose between structural and proximity information, we make use of an attention mechanism [47] defined using attention coefficients $e_\tau(u, v)$ that indicates the importance of node v's feature to node u w.r.t a particular relation type τ.

$$e^\tau_{u,v} = a_\tau(\boldsymbol{W}_\tau \boldsymbol{h}_u, \boldsymbol{W}_\tau \boldsymbol{h}_v) = \boldsymbol{a}^T [\boldsymbol{W}_\tau \boldsymbol{h}_u \oplus \boldsymbol{W}_\tau \boldsymbol{h}_v] \tag{3.12}$$

Algorithm 1: Efficient construction of computation graph

Input: Original Input Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$;
τ-hop Neighborhood Function on $\mathcal{G}$ as $\mathcal{N}_\tau(u) : u \to 2^{\mathcal{V}}$; Ordered Degree Sequence Function $s(V)$, $V \subset \mathcal{V}$; Sequence Comparison Cost Function $\mathcal{D}$

Hyper-Params.: No. of structural relations $T < dia(\mathcal{G})$

Output: Computation graph $\mathcal{C}$

begin
  $\mathcal{E}' \leftarrow \emptyset$; $n \leftarrow |\mathcal{V}|$;
  $w_\tau \in \mathbb{R}^{n \times n} \leftarrow 0, \forall \tau \in \{0, 1 \cdots T\}$; $w_p \in \mathbb{R}^{n \times n} \leftarrow 0$;
  $f_{-1} \leftarrow 0$;
  $S \leftarrow [degree(u)]\ \forall u \in \mathcal{V}$;
  Sort S; // $O(n \log n)$
  for $g \in \mathcal{V}$ **do**
    $pos \leftarrow$ BinarySearch$(S, degree(g))$; // $O(\log n)$
    $P \leftarrow \log(n)$ positions left and right of pos in S;
    for $h \in P$ **do**
      // P contains $O(\log n)$ nodes
      for $\tau \in \{0, 1 \cdots T\}$ **do**
        // Calculate structural dist. Eq. 3.9
        $f_\tau(g, h) \leftarrow f_{\tau-1}(g, h) + \mathcal{D}(s(\mathcal{N}_\tau(g)), s(\mathcal{N}_\tau(h)))$;
        // Calculate edge weights.
        $w_\tau(g, h) \leftarrow e^{-f_\tau(g,h)}$; // Eq. 3.10
        // Extend the edge set
        $\mathcal{E}' \leftarrow \mathcal{E}' \cup (g, h, \tau)$;
      end
    end
  end
  for $(g, h) \in \mathcal{E}$ **do**
    $\mathcal{E}' \leftarrow \mathcal{E}' \cup (g, h, p)$; $w_p(g, h) \leftarrow 1$
  end
  $\mathcal{R} \leftarrow \{0, 1 \cdots T\} \cup p$;
  return $\mathcal{C} = \left(\mathcal{V}, \mathcal{E}', \mathcal{R}, \{w_\tau, \forall \tau \in \mathcal{R}\}\right)$
end

where $a : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is a shared attention mechanism parameterized by a learnable attention weight vector $\boldsymbol{a} \in \mathbb{R}^{2d}$, $\cdot^T$ is transpose operation and $\oplus$ is concatenation operation. We inject the available computation graph information as follows,

$$\alpha_\tau(u, v) = \frac{\exp(\text{LeakyReLU}(e^\tau_{u,v}))}{\sum\limits_{x \in \mathcal{N}^\tau_1(u)} \exp(\text{LeakyReLU}(e^\tau_{u,x}))} \tag{3.13}$$

Finally, The UPDATE function is defined as,

$$\boldsymbol{h}'_u = \sigma\Big(\boldsymbol{W}_{\text{self}}\, \boldsymbol{h}_u + \ \boldsymbol{W}_{\text{neig}}\, \boldsymbol{m}_u\Big) \tag{3.14}$$

We name our general model as WRGNN (weighted relational GNN) and from the above definitions, we select two model variants viz. WRGAT and WRGCN for experimental analysis which specifies if attention mechanism is used or not respectively. Algorithm 2 provides a procedure for applying such a K layer WRGNN on the computation graph for semi-supervised node classification.

3.7 Experimental Results

In this section, we evaluate the performance of our framework against other methods under semi-supervised node classification setting. Note that our framework is quite flexible so any GNN model based on message passing could be adopted on our computation graph. To evaluate the performance of our method, we use real world graphs from different domains viz. Hyperlinked Web Pages [146], Citation Networks [160], Air Traffic Networks [101] and Internet's Inter-Domain Routing Network [149], [161]. These graphs are know to exhibit diverse mixing as we showed in Sec. 3.4 and thus provides us with the means to assess GNN based methods.

Algorithm 2: Procedure for Node Classification

Input: Computation Graph $\mathcal{C} = \Big(\mathcal{V}, \mathcal{E}', \mathcal{R}, \{w_\tau, \forall \tau \in \mathcal{R}\}\Big)$;
Node Features $\{\boldsymbol{x}_u \in \mathbb{R}^d, \forall u \in \mathcal{V}\}$;
1-hop τ-relation specific Neighbourhood Function on $\mathcal{C}$ as $\mathcal{N}_1^\tau(u) = \{v : (u, v, \tau) \in \mathcal{E}'\}$;

Output: Predicted node labels

begin
  $\boldsymbol{h}_u^0 \leftarrow \boldsymbol{x}_u \ , \forall u \in \mathcal{V}$;
  for $k = 1, \dots K$ **do**
    for $u \in \mathcal{V}$ **do**
      $\boldsymbol{m}_u^k \leftarrow \sum_{\tau \in \mathcal{R}} \sum_{v \in \mathcal{N}_1^\tau(u)} \boldsymbol{W}_\tau^k \, \boldsymbol{h}_v^{k-1} \, w_\tau(u, v) \, \alpha_\tau(u, v)$;
      $\boldsymbol{h}_u^k \leftarrow \sigma\Big(\boldsymbol{W}_{\text{self}}^k \, \boldsymbol{h}_u^{k-1} + \boldsymbol{W}_{\text{neig}}^k \, \boldsymbol{m}_u^k\Big)$
    end
    $\boldsymbol{h}_u^k \leftarrow \boldsymbol{h}_u^k / \|\boldsymbol{h}_u^k\|_2, \ \forall u \in \mathcal{V}$;
  end
  $\boldsymbol{z}_u \leftarrow \boldsymbol{h}_u^K, \ \forall u \in \mathcal{V}$;
  `// Predict node labels`
  for $u \in \mathcal{V}$ **do**
    $\boldsymbol{p}_u \leftarrow \text{softmax}(\text{MLP}_\theta(\boldsymbol{z}_u))$;
    $y_u \leftarrow \text{argmax}(\boldsymbol{p}_u)$
  end
  return $y_u \ , \forall u \in \mathcal{V}$
end

Table 3.1. Dataset statistics. Details of ★ discussed in Sec. 3.7.2.

	Hyperlinked Web Pages Network						Citation Network			Air Traffic Network			Internet Network
	Chameleon	Squirrel	Actor	Cornell	Texas	Wisconsin	Cora	Citeseer	Pubmed	Brazil	Europe	USA	BGP (small)
#Nodes $\lvert\mathcal{V}\rvert$	2,277	5,201	7,600	183	183	251	2,708	3,327	19,717	131	399	1,190	10,176
#Edges $\lvert\mathcal{E}\rvert$	31,421	198,493	26,752	280	295	466	5,429	4,732	44,338	1,038	5,995	13,599	206,799
#Classes $\lvert\mathcal{Y}\rvert$	5	5	5	5	5	5	7	6	3	4	4	4	7
#Node Features d	2,325	2,089	931	1,703	1,703	1,703	1,433	3,703	500	1	1	1	287
Assortativity r_{global}	0.0331	0.0070	0.0047	-0.0706	-0.2587	-0.1524	0.7710	0.6713	0.6860	0.0116	-0.0737	0.2629	0.0029
Train/Val/Test Splits ★	60/20/20						60/20/20			80/10/10			70/10/20

3.7.1 Datasets

Table 3.1 provides the statistics of the datasets we use for evaluation. We select a wide range of frequently evaluated datasets from different domains and below we provide brief descriptions.

- **Chameleon and Squirrel** collected by [162] are networks of hyperlinked web pages on Wikipedia related to animal topics. The nodes (here pages) are labelled from one of 5 classes based on the average traffic (views) they received. Node features are bag-of-words representation of nouns in the respective pages. We download the processed data from Pei, Wei, Chang, *et al.* [146].

- **Actor** is a co-occurrence network based on the film-director-actor-writer network from [163]. In this dataset, nodes represent actor web pages on Wikipedia and edges symbolize co-occurrence on the same web page. Node features are bag-of-words representation of the corresponding pages and labels are placed according to topics on actor web page. The dataset is from Pei, Wei, Chang, *et al.* [146].

- **Cornell, Texas and Wisconsin** collected as part of CMU WebKB project. Nodes are university web pages and edges are hyperlinks between them. Node labels are one of student, project, course, staff or faculty. Node features are bag-of-words representation of the corresponding web pages. The dataset is also from Pei, Wei, Chang, *et al.* [146].

- **Cora, Citeseer and Pubmed** introduced by [160], [164] are citation networks where node represent scientific papers and edges are citation relationships. Node features are bag-of-words representation of the paper and labels are the scientific field they represent.

- **Air Traffic Networks** from three regions **Brazil, Europe and USA** is collected by the respective civil aviation agencies. Nodes represent airports and edges mean the presence of commercial routes between nodes. Nodes are labelled according to the traffic (aircraft landings and takeoffs) or level of activity (by passenger count) an airport witnesses. We get the data from Ribeiro, Saverese, and Figueiredo [101].

- **BGP Network** collected by [161] represents the inter-domain structure of the Internet. Nodes represent autonomous systems and edges indicate business relationships between nodes. Node features contain categorical location and topology information and labels are based on the type or tier of the autonomous system. We get the processed data from Hou, Zhang, Cheng, *et al.* [149].

3.7.2 Baseline Methods and Experiment Setup

We primarily consider methods that utilize GNN models which adopted messaging passing operation as their main backbones. GCN [45] and GraphSage [46] are methods where convolutions are strictly based on first order neighbour aggregation scheme for each layer. GCN-Cheby [165] generalizes convolutions with the help of k-hop localized spectral filters. GAT [47] adaptively aggregates immediate neighbour information using attention coefficients which are also derived from node features. MixHop [156] and H_2GCN [147] generalize the node aggregation beyond the first order neighbourhoods and dynamically considers node features k-hops away. It is important to note that all baseline methods operate on the original graph and thus have access to only proximity information albeit in different forms.

We perform semi-supervised node classification and use the classification accuracy and F1-Micro scores as performance metrics to evaluate different approaches. The training/validation/testing data splits for all the methods to be compared is shown in Table 3.1. For Hyperlinked Web Page and Citation Networks, we report the performance of mean $\pm$ std. dev. on 10 random splits provided by Pei et al. [146] which is available on their GitHub [2]. Reported values for Air Traffic Networks and BGP Networks are based on 20 and 10 random splits respectively. All the implemented methods including our own are all trained until the loss function converges and the final models are selected based on the prediction performance on the validation sets. The sensitivity analysis and hyper-parameter search is performed on the validation set and more details are provided next.

For all our experiments, we use Adam [166] algorithm with learning rate of $\{1e-2, 1e-3\}$ and weight decay of $\{0, 1e-5, 5e-4, 5e-6\}$ to optimize our model. Our WRGNN model

[2]↑https://github.com/graphdml-uiuc-jlu/geom-gcn/tree/master/splits

variants contains 2 layers with an another fully connected MLP_θ on top of it. We select ReLU as the nonlinear activation. For the model with attention mechanism i.e. WRGAT, we use LeakyReLU with a negative input slope of 0.2. We sweep all the hidden dimensions from $\{16, 32, 64, 128\}$ for the WRGNN layer and $\{32, 64, 128\}$ for the final MLP_θ layer using cross validation. We set the maximum learning epochs as 500 with early stopping parameter 100. Specifically, for Hyperlinked Web Page Networks (Table 3.2) dropout operation with a probability of 0.8 is applied on each WRGNN layer. For BGP Network our model uses dropout of 0.5, learning rate as 1e-2 with weight decay of 0. Finally, for Air Traffic Networks (Table 3.3), dropout is set to 0.6, learning rate is $1e-3$ with weight decay $5e-6$ and T is set to 5, 5 and 8 for Brazil, Europe and USA datasets, respectively. The hyper-parameter T related to structural similarity calculations is chosen based on the validation set. Its sensitivity against validation accuracy for some datasets are provided in Fig. 3.6 and 3.7.

3.7.3 Local Assortativity Distribution Shift

We first perform a quick study to confirm our graph transformation algorithm indeed enhances the level of local assortativity in comparison to the original input graph. To achieve this, we focus on all the disassortative nodes ($r_{\text{local}} < 0$) from the input graph $\mathcal{G}$ and track how they mix in the transformed computation graph $\mathcal{C}$. From Figure 3.4, a clear distribution shift of local assortativity could be observed. That is, the previously disassortative nodes in G, are more assortative in the new computation graph after transformation. This empirically verifies the claim we raised earlier about similar structural regularities between a pair of nodes being captured in the computation graph as a result of our transformation (hence the increased assortativity). In addition, we have a reason to believe that performance improvement of our relational GNN model variants WRGCN and WRGAT are rooted in the increase of local assortativity for disassortative nodes in the original graph. This is clearly witnessed in Figure 3.3 (shown in red).

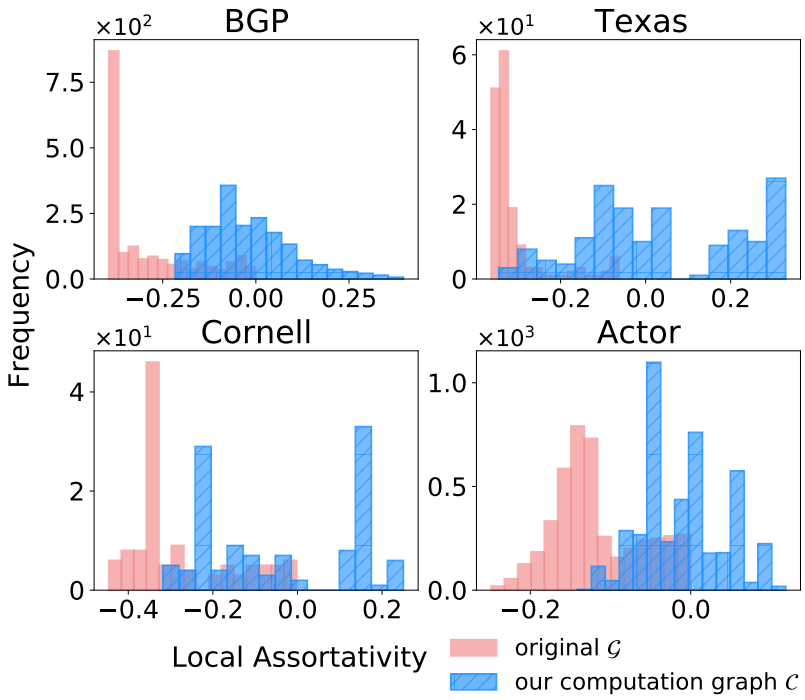

Figure 3.4. Distributions of node level assortativity for the original graph and the newly constructed computation graph on different datasets.

3.7.4 Node Classification Performance

Hyperlinked Web Page and Citation Networks. Table 3.2 shows the evaluation of our framework against other GNN based baselines on the node classification task for these datasets. Mean test accuracy and standard deviation numbers are reported for each method. Values for H_2GCN [147]and Geom-GCN [146] are taken from the respective papers. We use the same train/val/test splits as [146], [147] for comparability. We find that our framework consistently performs well for the Hyperlinked Web Page Networks owing to the rich structural regularities that our computation graph captures. On Citation Networks which predominantly have assortative mixing, our framework gives comparable performance to baselines. H_2GCN and MixHop utilize higher order neighbourhoods in each convolution which does help over standard GNN methods, but can also be a lot harder to train and they also suffer from oversmoothing problem. Our framework makes it possible to define computation graphs that can directly tap into the structural regularities thereby making effective use of graph information, while the message passing defined on such a computation graph

Table 3.2. Semi-supervised node classification showing mean test accuracy ± std. over 10 runs. Club Suit [♣] denotes result obtained from the best model variant of respective papers.

	Chameleon	Squirrel	Actor	Cornell	Texas	Wisconsin	Cora	Citeseer	Pubmed
GCN	59.82 ± 2.58	36.89±1.34	30.26±0.79	57.03±4.67	59.46±5.25	59.80±6.99	87.28±1.26	76.68±1.64	87.38±0.66
GraphSage	58.73±1.68	41.61±0.74	34.23±0.99	75.95±5.01	82.43±6.14	81.18±5.56	86.90±1.04	76.04±1.30	88.45±0.50
GAT	54.69±1.95	30.62±2.11	26.28±1.73	58.92±3.32	58.38±4.45	55.29±8.71	86.37±1.69	75.46±1.72	87.62±0.42
GCN-Cheby	55.24±2.76	43.86±1.64	34.11±1.09	74.32±7.46	77.30±4.07	79.41±4.46	86.86±0.96	76.25±1.76	88.08±0.52
MixHop	60.50±2.53	43.80±1.48	32.22±2.34	73.51±6.34	77.84±7.73	75.88±4.90	83.10±2.03	70.75±2.95	80.75±2.29
Geom-GCN ♣	60.90	38.14	31.63	60.81	67.57	64.12	85.27	77.99	90.05
H_2GCN ♣	59.39±1.98	37.90±2.02	35.86±1.03	82.16±4.80	84.86±6.77	86.67±4.69	87.67±1.42	76.72±1.50	88.50±0.64
Ours (WRGAT)	65.24±0.87	48.85±0.78	36.53±0.77	81.62±3.90	83.62±5.50	86.98±3.78	88.20±2.26	76.81±1.89	88.52±0.92

Table 3.3. Node classification on Air Traffic Networks and BGP Network. Mean test acccuracy ± std. is shown over 20 runs.

	Brazil	Europe	USA	BGP
GCN	64.55±4.18	54.83±2.69	56.58±1.11	53.33±0.18
GraphSage	70.65±5.33	56.29±3.21	50.85±2.83	65.19±0.28
GIN	71.89±3.60	57.05±4.08	58.87±2.12	49.51±1.52
Struc2vec	70.88±4.26	57.94±4.01	61.92±2.61	48.40±1.39
Ours	76.92±5.45	57.12±2.81	63.02±1.87	66.54±0.48

takes care of adapting to node features. Overall performance rank of various methods in Table 3.2 on both disassortative and assortative datasets is shown in Figure 3.5. It is clear that our model variant WRGAT run on the computation graph achieves the lowest rank overall hence supporting our claim of achieving superior performance in both disassortative and assortative regime.

Air Traffic and BGP Networks. Table 3.3 gives comparisons of our framework against other GNN methods for node classification on multiple Air Traffic Networks (ATNs) and the Internet Domain Network (BGP). It is clear from the table that, our framework achieves strong performance compared to other baselines. ATNs don't have node attributes and baseline GNN methods perform poorly while our framework utilizing structure is better suited for the task. Strong performance is seen because our construction of the computation graph explicitly looks at different neighbourhoods that is very beneficial in air traffic networks.

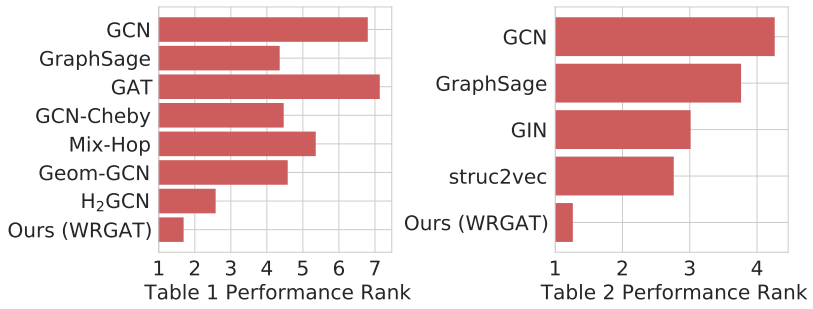

Figure 3.5. Overall performance rank of various methods for datasets in Table 3.2 & 3.3. Lower rank signifies better performance

Major hubs connect to local airports (disassortative) while they also mix with other hubs (assortative). Structure alone is capable of capturing this diversity and our computation graph takes a step in that direction. The BGP network also exhibits diverse mixing and we believe that the strong performance is due to the adaptive selection of both proximity and structural information.

3.7.5 Sensitivity Analysis

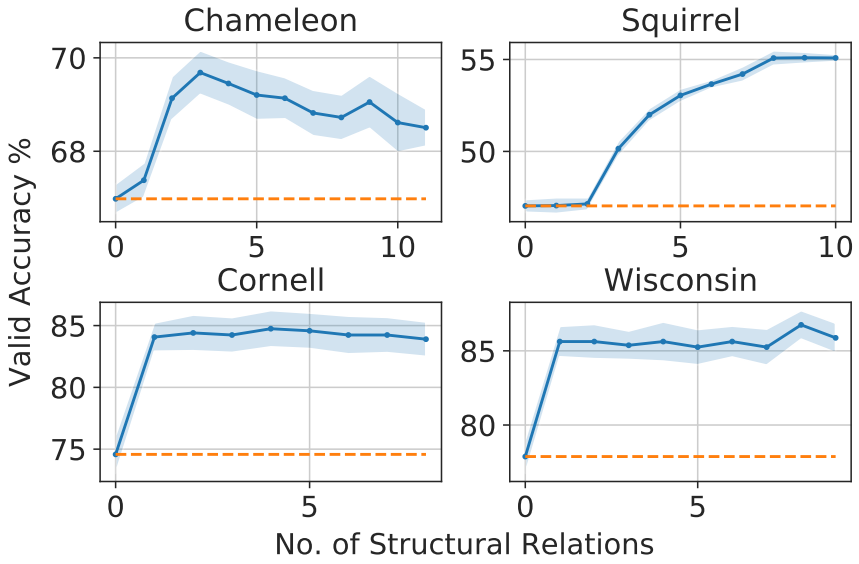

Figure 3.6. Sensitivity w.r.t structure relations. Baseline is using only proximity information.

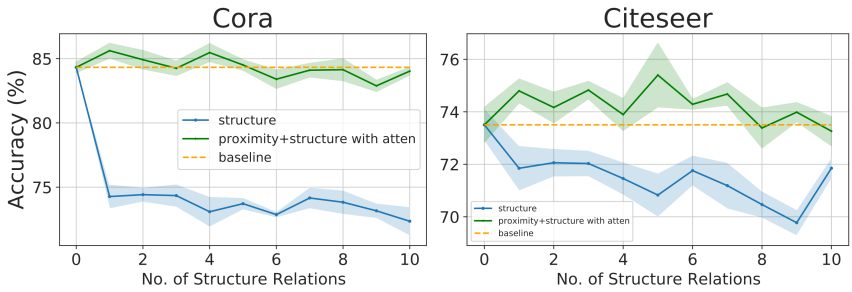

Figure 3.7. Sensitivity w.r.t structure relations. Baseline is using only proximity information.

We provide sensitivity analysis based on validation split w.r.t the number of structural relations used in our computation graph, which is our main hyper-parameter. Figure 3.6 shows the analysis for disassortative networks and is quite clear that adding structural information at various scales significantly improves validation performance when compared to baseline of just using proximity information. Figure 3.7 shows an interesting picture for highly assortative networks. Here, the take away is that structure alone is not useful (blue line), while an adaptive structure+proximity with attention model i.e. using WRGAT (green line) is able to give better consistent performance over proximity only baseline.

3.7.6 Ablation Analysis

Figure 3.8 provides ablation analysis for a number of networks and shows test performance gains over proximity only baseline for our model variants (WRGCN and WRGAT) in y-axis and against the use of either proximity, structure or both kinds of information in x-axis. For the top row consisting of predominantly disassortative networks, the take away is that structure only information is capable of providing better performance over proximity only information, but when both kinds of information is available, the attention model is best suited. For the bottom row consisting of highly assortative networks, clearly structure only information hurts performance compared to baseline. We reason that structure becomes less important for graphs with high assortativity however, when both kinds of information is available and attention is used we witness the best gains. These results further supports our model choices.

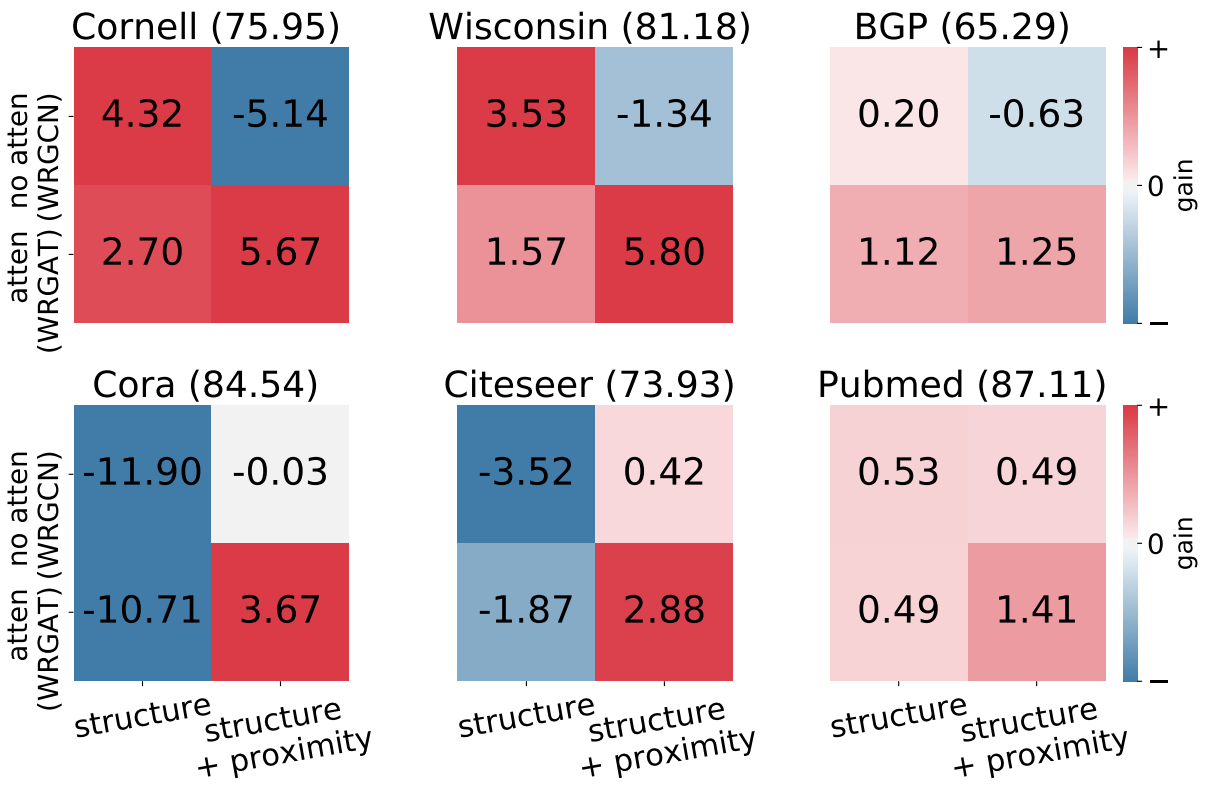

Figure 3.8. Ablation w.r.t different model and graph information choices. Baseline (proximity only) is shown in parenthesis. Numbers indicate gain over baseline.

The previous ablation study shows test performance gains for the whole dataset. We now study how these different model configurations behave w.r.t node level local assortativity we introduced earlier. With Figure 3.9 we are able to study their behaviour in finer detail due to the notion of local assortativity. It supports our hypothesis that in the high local assortativity regime, proximity information dominates in discriminatory power and structure only information leads to worsened performance as it becomes irrelevant. However when using both kinds with an adaptive mechanism we can see increased performance over the full spectrum.

3.8 Discussion

The level of mixing plays a crucial role in characterizing real world networks. In this work we have used the quantification of local mixing patterns to study the predictive per-

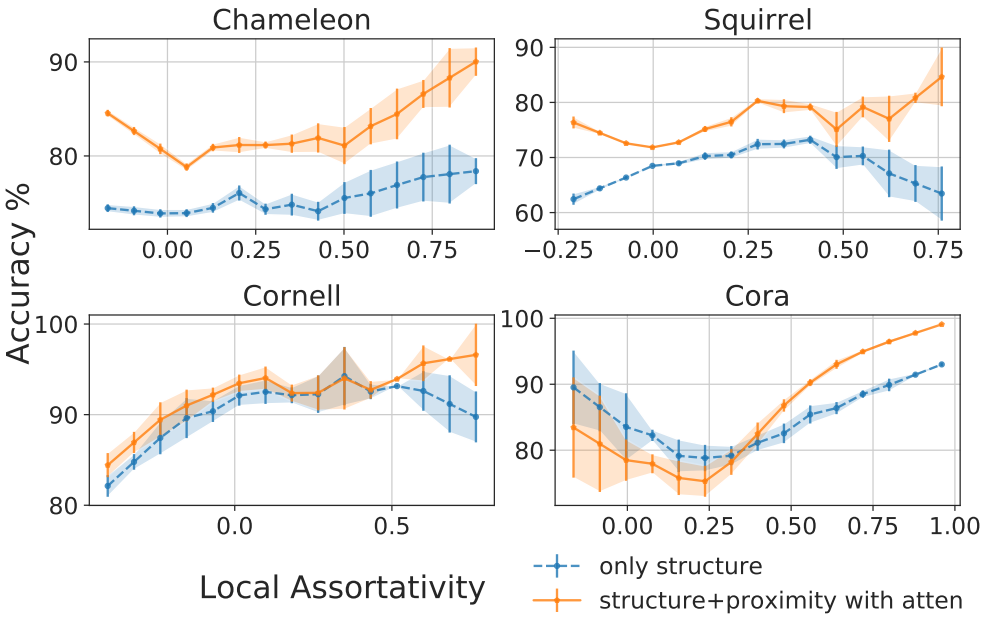

Figure 3.9. Ablation w.r.t graph info and local assortativity

formance of graph neural networks. Our observations and results offer a new perspective to study the limitations of GNNs. Motivated by the findings of our analysis, we develop a graph transformation technique that has shown to experimentally improve assortativity owing to the use of structural regularities in the input graph and there by increasing GNN performance. Extensive experiments on various real world networks from different domains supports our claim of running GNNs on a transformed computation graph and adaptively choosing between structure and proximity information rather than on the original. The connections we find with mixing patterns and GNN learnability provides motivation for future work to provide possible theoretical claims relating the two. We also hope that this study leads to the creation of other benchmark datasets with diverse mixing patterns which can aid in the robust evaluation of future GNN methods.

4. GRAPH REPRESENTATION LEARNING FOR RANKING TEMPORAL LINKS

Many practical machine learning applications in complex systems focus on predicting future interactions among entities, based on an observed sequence of past interactions, for example, recommendations in heterogeneous information systems [66], [67], social behavior prediction [68], [69], and financial system monitoring [70], [71]. In this case representing the data as a temporal graph [72] and formulating the prediction task as link prediction has proved quite successful.

Recently, temporal graph representation learning (T-GRL) that learns the representations of temporal graphs has shown great potential to improve link prediction accuracy in temporal settings [66], [68], [74], [75], [79], [167]. To date the methods optimize the representations using a pointwise loss function geared to maximize accuracy *independently* over future links (conditioned on the graph of interactions in the past). This objective doesn't reflect the fact that many complex systems have constraints that produce dependencies among future links. For example, a user selects among a set of items to decide what to purchase next, or among a set of users to connect to next in a social network. In this case, the predictions are either presented as a ranked list to the user (ie. thru recommendations) or resources are allocated based on the ranking assuming that actions at the top of the rank are more likely to happen (eg. caching information, matching advertisers). While we can rank predictions from a model that is learned from a pointwise loss, it is likely that representations learned with a *joint* ranking loss over the list of candidates will generalize better for ranking tasks.

Although it is a common practice that pointwise losses can be transformed for ranking by applying listwise losses over all ranking candidates [168], applying these ideas to T-GRL is not straightforward, particularly for inductive models. This is due to a tradeoff between expressivity in the learned representation and scalability of estimation/inference, which we discuss next.

First, many previous T-GRL models inherit the limited expressivity of graph neural networks (GNNs) for link prediction. They associate each node with a representation and update that representation by aggregating the representations of its interacted nodes [66],

[74], [75], [79], [167]. However, the obtained representations are known to fail to encode some structural information important for link prediction [68], [76]–[78]. Figure 4.1 gives a toy example in temporal graphs to illustrate the issue. Many T-GRL models (e.g. TGN [75]) will learn the same representations for v_1 and v_3 at t_3, so the models cannot distinguish whether u is more likely to be a friend of v_1 or v_3, and would predict them as equally likely. What those representations fail to capture is that v_3 and u have had a common friend before t_3 while v_1 and u have not. *Labeling tricks* have been proposed recently on static graphs to address this issue [77], [78]. However, the standard idea of labeling tricks that labels the node pair of interest different from other nodes suffer from the following scalability issue.

Second, it is often difficult to compute a listwise loss for link prediction problems with large candidate sets. Even if a rough filtering of candidates is adopted (e.g., only consider the nodes within the first several hops away from the query node), which is similar to the *matching* or *pre-ranking* procedure in traditional ranking systems [169], the pool may still contain hundreds or even thousands of candidates. The key idea to compute these losses efficiently is to compute the representations for all the candidates jointly with one forward pass of the encoder rather than independently.

However, the standard labeling trick [78] may not allow such joint computation. In particular, a recent model CAWN [68] can online construct structural features for a node pair as its adopted labeling trick, while such construction has to be done independently for each candidate link and cannot be shared over a candidate set.

In this work, we address these issues and propose **T**emporal **G**raph network for **RANK**ing (TGRank)[1]. Our pipeline is shown in Fig. 4.2. TGRank can (i) optimize a list-wise loss for improved ranking, and (ii) incorporate a labeling trick to improve expressivity while allowing for inference over the candidate set jointly, to improve efficiency. Specifically, during the training, for each query with a center node and a set of candidate nodes, TGRank labels the center node different from others and leverages GNNs to diffuse such a label to every ranking candidate. The parameterized label diffusion procedure aggregates the timestamps, the multiplicity, and the features of historical interactions along the network from the center nodes to all the candidates, and gives provably more expressive power to better capture the

[1]↑https://github.com/susheels/tgrank

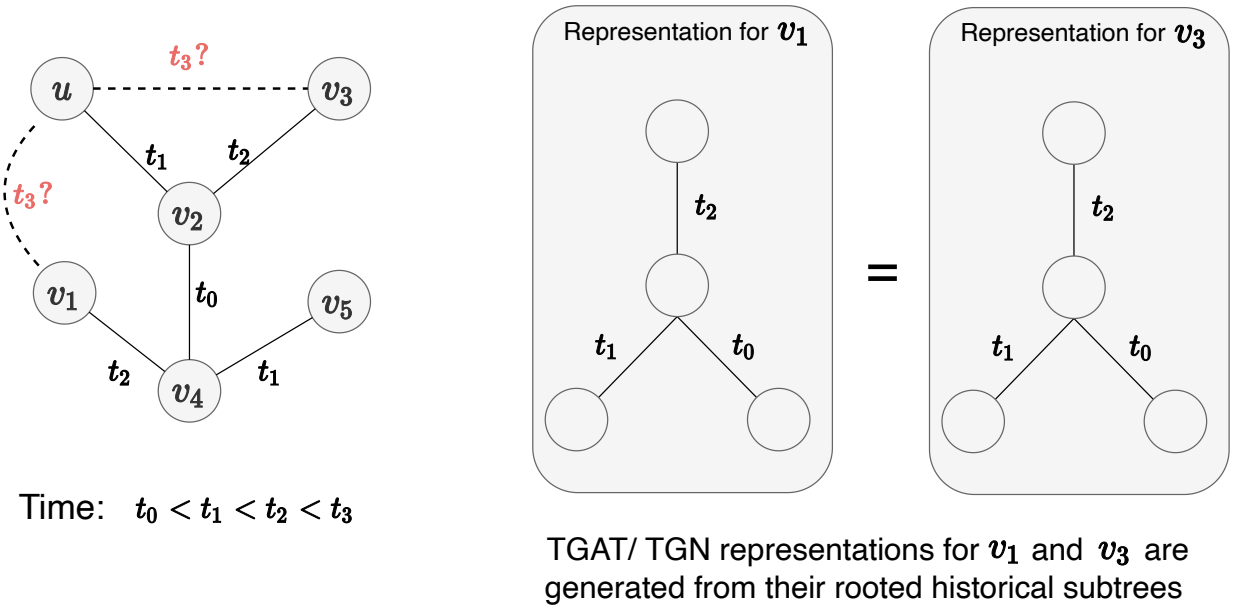

Figure 4.1. Illustration of how node representations given by GNN models (right) can fail to distinguish some pairwise relationships (left).

needed structural features for link prediction (see the example in Fig. 4.2). Since all the candidate nodes will be associated with the likelihoods of interactions jointly, this facilitates efficient optimization of the ranking loss. During testing, a similar label diffusion procedure is done to jointly infer over all candidates. Table 4.1 lists the characteristics of TGRank compared to CAWN [68] and TGN [75], and shows the previous tradeoff between expressivity and efficiency. It also includes a basic extension to TGN, which optimizes list-wise loss (TGN-LW) rather than pointwise loss.

We extensively evaluate TGRank over 6 real-world temporal graphs with up-to 1M+ node future interactions (100M+ ranking candidates) for training and 200k node future interactions (20M+ ranking candidates) for testing. TGRank outperforms the best baselines on average by 14.21% for transductive ranking and by 16.25% for inductive ranking mean reciprocal rank (MRR). TGRank also achieves up-to 65× speed-up compared to CAWN during inference and is more efficient in both training (up-to 8× speed-up) and inference (up-to 2× speed-up) than the baselines without labeling tricks e.g., TGN-LW.

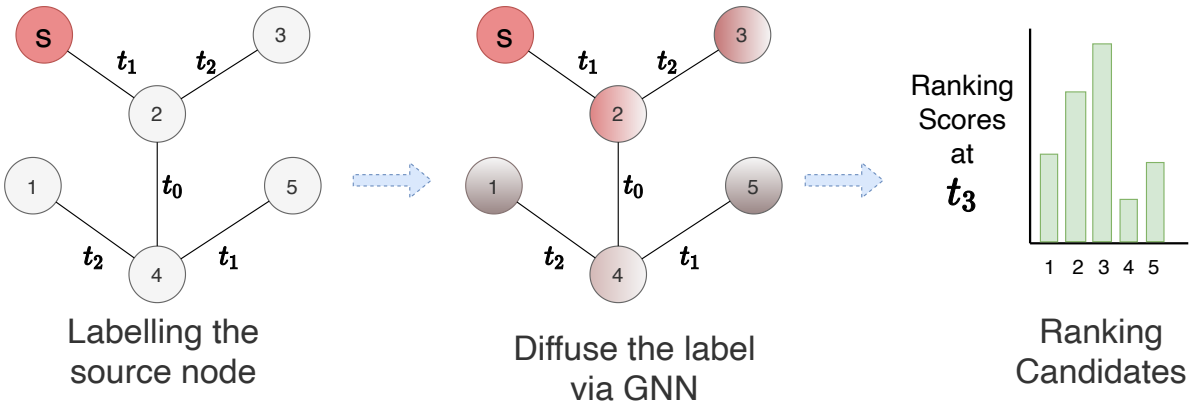

Figure 4.2. Pipeline of TGRank. Here, the center node s is tasked at ranking candidates in the subgraph surrounding it at a time t_3 (simulating the example in Fig. 4.1: node $s \leftrightarrow u$, node $1 \leftrightarrow v_1$, node $3 \leftrightarrow v_3$). TGRank follows three steps. (1) Labeling the s differently from candidates. (2) Parameterized label diffusion using GNNs to propagate timestamps, multiplicity and features. (3) Ranking candidates. Importantly, the pools of effective candidates are assumed to be in the sub-graph surrounding s and a list-wise loss is used to optimize ranking scores jointly.

Table 4.1. Characteristics of our method (TGRank) and comparison T-GRL methods.

	CAWN	TGN	TGN-LW	**TGRank**
Representation: inductive	✓	✓	✓	✓
Representation: more expressive	✓	✗	✗	✓
Objective: ranking loss	✗	✗	✓	✓
Scalable inference over candidates	✗	✓	✓	✓

4.1 Related Work

The earliest works for T-GRL break temporal networks into snapshots by aggregating temporal interactions that appear within consecutive time ranges [70], [170]–[173]. This allows for utilizing off the shelf static GNNs [174], [175] to extract the structural patterns and recurrent neural nets (RNNs) or transformers to combine the extracted patterns along with time to make prediction. However, all these methods cannot capture the evolving dynamics in different levels of granularity. In light of these issue, several recent works deal with the stream of temporal interactions directly [66], [68], [74], [75], [79], [176]–[179]. Know-E [179], JODIE [66], DyRep [79] and DyGNN [178] track representations of individual nodes

over time and update them when the nodes occur in an interaction event. To improve the inductive prediction performance, TGAT [74] does not track node representations over time. Instead, TGAT samples from the historical neighborhood for every node of interest and applies attention layers to aggregate the neighbor's representations to get a temporal node representation. TGN [75] both tracks node representations over time and adopts the historical sampling to update them. All these models, unless using node IDs as features, suffer from the node ambiguity issue of the example as illustrated in the Fig. 4.1 because they cannot capture some important structural features [77], [78]. Some works also adopt network embedding techniques to encode temporal networks [176], [177], [180]–[182], which do not suffer from the node ambiguity issue but at the cost of the capability of making inductive prediction and using network attributes. A recent approach CAWN [68] directly constructs structural features for each node-pair of interest, which also keeps inductive capability, but the complexity of CAWN is so high as a number of random walks need to be sampled to construct the structural features.

None of the above models have considered being optimized for ranking objectives, though many of them are indeed aware of the practical importance of ranking prediction and adopt ranking metrics in their evaluation [66], [70], [79], [171], [178], [179]. Learning-to-Rank (LtR) studies how to directly optimize a model to make ranking prediction and has been extensively studied in recommender system and information retrieval literatures. Some works that have applied GRL for dynamic recommendation are relevant to us [67], [183]–[186]. Among them, only a few [67], [183], [186] adopt LtR objectives while two of them focus on session recommendation which works in a substantially different setting [183], [186]. TGSRec [67] adopts an attention-based T-GRL encoder which is very similar to TGN [75] but with a LtR objective, but it can only apply to bipartite networks.

4.2 Preliminaries

In this section, we introduce some preliminaries for TGRank, including notations and definitions.

Definition 4.2.1 (Temporal Graph)**.** *A temporal graph is modelled as a sequence* $\mathcal{E} = \{(e_1, t_1), (e_2, t_2), \dots\}$ *of temporal edge (interaction) events. Each edge* $e_i = (u_i, v_i)$ *occurs between two nodes* u_i *and* v_i *with a timestamp* $t_i \in \mathbb{R}_{>0}$*.*

For now, we ignore edge features that may be associated with each edge event and later, we will show how it can be incorporated. Note that in temporal graphs, there may exist multi-edges between the same pair of nodes. We use $\mathcal{E}_t = \{(e, t') \in \mathcal{E} | t' < t\}$ and $\mathcal{V}_t = \{v \in e | (e, t) \in \mathcal{E}_t\}$ to represent the set of all edges and nodes that occur **before** time t respectively. Collectively, we will use $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ to represent the corresponding graph. We now define the temporal neighborhood around a node.

Definition 4.2.2 (*k-hop t-Temporal Neighborhood*)**.** *Given a node* s *and cutoff time* t*, the k-hop t-temporal subgraph around* $s \in \mathcal{V}_t$*,* $\mathcal{G}^k_{t,s}$ *is the subgraph induced by node set* $\{v \in \mathcal{V}_t \mid \textsc{Spd}\,(s, v) \leq k\}$ *on* $\mathcal{G}_t$ *where,* $\textsc{Spd}\,(\cdot, \cdot)$ *is the shortest path distance between two nodes. Note that* $\mathcal{G}^k_{t,s}$ *only consists of edges occurring before time* t*.*

Next, we also introduce the graph diffusion based on PageRank [187]. The initial vector used in Personalized PageRank [188] in the definition motivates the labeling trick of TGRank encoder.

Definition 4.2.3 (Graph Diffusion)**.** *Given a static graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *with adjacency matrix* A *and diagonal degree matrix* D*, define the diffusion matrix as* $W = AD^{-1}$*. The graph pagerank diffusion procedure follows that given some initial labels over the node set* $x^0 \in \mathbb{R}^{|V|}$*, for some* $\alpha > 0$*, do*

$$x^k = \alpha W x^{k-1} + (1 - \alpha) x^0, \quad k = 1, 2, ... \tag{4.1}$$

By setting the initial label x *is a 0-1 vector that contains a unique 1 for a given seed node,* x^∞ *in Eq. 4.1 gives Personalized Pagerank vector [188], where each entry characterizes the relevance between the corresponding node and the seed node.*

4.3 Ranking Problems and TGRank

Our goal is to learn a T-GRL model that can accurately predict how temporal graph evolves. Specifically, the model can predict the next node or set of nodes that a center node may interact with. We formulate this question as a ranking problem as follows.

Definition 4.3.1 (The Ranking Problem for Temporal Graph Prediction)**.** *Consider a set of m queries $\{q_1, q_2, ..., q_m\}$. Each query $q_i = (s_i, \mathbb{D}_i, t_i)$ occurs at time t_i, and consists of a center node s_i whose action is to be predicted and a number of associated ranking candidate nodes $\mathbb{D}_i = (d_1^i, d_2^i, \ldots, d_{n_i}^i)$, $d_j^i \in \mathcal{V}_{t_i}$. The ranking problem is to learn ϕ to encode the temporal graph $\mathcal{G}_{t_i}$ and get $\phi(q_i; \mathcal{G}_{t_i}) \in \mathbb{R}^{n_i}$, where each entry $[\phi(q_i; \mathcal{G}_{t_i})]_j$ is the relevant score for the candidate $j \in \mathbb{D}_i$. Predict the next-step action of s_i to interact with the top-ranked candidates based on $\phi(q_i; \mathcal{G}_{t_i})$.*

Later, we ignore the historical graph $\mathcal{G}_{t_i}$ and use $\phi(q_i)$ to denote the encoder for simplicity. The candidate list may include the entire node set while a more practical and interesting setting is to consider a more challenging candidate list. In recommender systems, the candidate list is often given by a matching or pre-ranking algorithm that search the most relevant hundreds or thousands of candidates and put them into the list. In our setting, we assume that all the nodes in the first k-hop historical subgraph $\mathcal{G}_{t_i,s_i}^k$ of the center node s_i give the ranking candidates. We are aware that this assumption may lose some feature-wise relevant candidates, so we verify this assumption by conducting data analysis on real temporal graphs, which shows that on average 97.39% of all the positive edges connect to the nodes in this list and this list cover 3.17 % to 55.12 % of the entire node set for these networks. Complete statistics are shown in Table 4.2. We leave a more extensive study on the definition of the candidate set in the future.

Next, we will introduce TGRank, including the design of its encoder, the theoretical argument on its higher expressivity, and the complexity analysis.

4.3.1 Algorithmic Motivation

To motivate the design of TGRank encoder, we first introduce the insight behind it—which is that the candidate node that the center node s will likely interact with next is often strongly related to the network structure that bridges these two nodes in the graph. The form of the structure can be transformed into some relevance measure. In static networks, extensive work [22] has been done to compare different relevance measures to characterize the likelihood of an edge between two nodes. Shortest path distance (SPD) is typically not an effective measure as otherwise, since s will connect to the nodes that are 2-hops away with equal likelihood. Although SPD is ineffective, if we count the number of 2-hop paths between the two nodes, we can get an extremely effective measure of *the number of common neighbors*. Importantly, different graph diffusion procedures along the network structure from s to the candidate nodes also often provide effective relevance measures, such as *hitting time* [189] and *personalized Pagerank scores* [188].

However, designing the relevance measures becomes much more involved in temporal networks. There may be multiple edges between two nodes and those edges are further associated with timestamps. Intuitively, both larger multiplicity and having more recent timestamps indicate a stronger connection or higher relevance between two nodes. Real-world complex systems may contain even more complicated dynamics where the order of edges also matters, such as the metabolic networks, where stimuli and inhibition interactions between proteins often follow an alternative manner [190].

The encoder ϕ in TGRank is essentially designed to address such complexity by learning a data-driven relevance measure between the center node and the ranking candidates. The measure is parameterized and trained to optimally combine edge timestamps, multiplicity and attributes (if there are some) to fit the data. In a high level, our encoder is inspired by the personalized Pagerank diffusion as Def. 4.2.3. We view the center node as a seed node and associate it with a label that is different from all other nodes (a labeling trick [78]). Then, use a GNN to parameterize the diffusion procedure of the label over the network. The diffused value on each candidate node can be used as its relevance to the center node. Note that our encoder shares some similar spirits with the recent works on static graphs [191],

[192] while these two models do not apply to temporal graphs. This idea is also inspired by SEAL (for static graphs) [78], [193] and CAWN (for temporal graphs) [68]. However, these models suffer from scalability issues as discussed in Sec. 4.1. Next, we specify our encoding algorithm.

4.3.2 TGRank Encoder

Given a query $q = (s, \mathbb{D}, t)$, TGRank will encode the temporal graph $\mathcal{G}_t$ before time t and learn the relevance scores $\phi(q) \in \mathbb{R}^{|\mathbb{D}|}$.

TGRank first associates each node v in the temporal graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ with an initialized representation $\boldsymbol{h}_v^0$. The center node s is initialized in a different way from the other nodes, where $\boldsymbol{h}_s^0 = \boldsymbol{c} \in \mathbb{R}^F$ that is a learnable vector and $\boldsymbol{h}_v^0 = \mathbf{0}$ for $v \neq s$. TGRank then adopts a GNN to mimic the graph diffusion of the initialized features. That is, for $r = 1, 2, ..., R,$, do

$$\boldsymbol{m}_{u,v,t'}^r = \textsc{Message}^r \left(\boldsymbol{h}_u^{r-1}, \text{time-encoding}(t - t') \right), \ \forall (u, v, t') \in \mathcal{E}_t \tag{4.2}$$

$$\boldsymbol{h}_v^r = \textsc{Update}^r \left(\boldsymbol{h}_v^0, \ \boldsymbol{h}_u^{r-1}, \ \left\{ \boldsymbol{m}_{u,v,t'}^r \ \middle| \ (u, v, t') \in \mathcal{E}_t \right\} \right), \ \forall v \in \mathcal{V}_t \tag{4.3}$$

For time encoding, we adopt parametrized Fourier features [194] which follows Eq. 4.4,

$$\text{time-encoding}(t) = [\cos(a_1 t) + b_1, \cos(a_2 t) + b_2, \ldots, \cos(a_F t) + b_F] \tag{4.4}$$

where, a_i, b_i, for $i \in \{1, 2, \ldots, F\}$ represent learnable parameters. This time encoding has been widely used in T-GRL works [68], [74], [75], [195] and is known to be more expressive [196]–[198].

Note that in the node representation update step Eq. 4.3, we inject the initial representation $\boldsymbol{h}_v^0$, which is inspired by the graph diffusion in Def. 4.2.3. Empirically, we find this helps reduce the oversmoothing problem that GNNs typically suffer from [62], [63]. TGRank can also encode *node and edge attributes* by simply concatenating node attributes with the initial representation and edge attributes with the input of the message function in Eq. 4.2.

In our instantiation, the $\textsc{Message}^r(\cdot)$ function with a simple single linear layer followed by nonlinearity ReLU. The $\textsc{Update}^r(\cdot)$ function is a sum pooling of all its input followed by a

linear layer. Other elaborate instantiations inspired by static GNNs [39] (e.g., neighborhood attention [47]), can also be incorporated, but our simple and efficient functions show strong empirical performance.

All the node representations at the end of R iterations of Eq. 4.2 and Eq. 4.3 can be used for ranking. Specifically, we set the corresponding entry $[\phi(q)]_v = \text{MLP}(\boldsymbol{h}_v^R)$, for $v \in \mathbb{D}$. In our experiments, we predict the next single edge for the center node, so we adopt the listwise loss used as follows,

$$\mathcal{L}(\phi(q), v^*) = -\log\left(\frac{\exp([\phi(q)]_{v^*})}{\sum_{u\in\mathbb{D}}\exp([\phi(q)]_u)}\right) \tag{4.5}$$

where v^* corresponds to the unique candidate node that s is to connect to next (i.e., the ground truth). This loss can be summarized over all queries $\{q_1, q_2, ..., q_m\}$ in the training dataset, minimizing which optimizes the parameters of $\phi(\cdot)$. Note that minimizing the above listwise loss is equivalent to maximizing the logarithm of a relaxation of the mean reciprocal rank (MRR), i.e. $\log\left(\frac{1}{m}\sum_{i=1}^{m} 1/\text{rank}([\phi(q_i)]_{j^*})\right)$, where j^* corresponds to the only candidate linked by s_i next [199].

In comparison, previous works, such as TGAT [74], TGN [75] and CAWN [68] adopt a pointwise loss, e.g., computing the cross-entropy loss for each node $u \in \mathbb{D}$ separately and then doing sum.

Temporal Graph Downsampling

Note that the sizes of node or edge sets of real temporal graphs $\mathcal{G}_t$ may grow over time. Moreover, the edges with timestamps long ago may not be as effective as the recent edges. Therefore, it is reasonable to downsample the nodes and edges in $\mathcal{G}_t$. We adopt the following downsampling strategy. We first extract the k-hop neighborhood $\mathcal{G}_{t,s}^k$ which by assumption includes all effective candidates. Starting from the center node s, for each sampled node in $i-1$-hop, we sample at most B_1 most recent neighbors of this node in i-hop, $i = 1, 2, ..., k$. For every two sampled neighboring nodes, we keep at most B_2 most recent edges between them. In our experiments, we keep $B_1, B_2 \leq 20$ and also evaluate the generalization capability of TGRank by letting the training and testing stages adopt different

Algorithm 3: TGRank Training

Input: TRAIN BATCH $\left\{(q_i = (s_i, \mathbb{D}_i, t_i), v_i^*)\right\}_{i=1}^{m}$;

Hyper-Params : Emb. Dim. F; Layers R;
Temporal graph downsampling parameters k, B_1, B_2;

$loss \leftarrow 0$

foreach $(q = (s, \mathbb{D}, t), v^*) \in$ *TRAIN BATCH* **do**

  Get historical graph $\mathcal{G}_t$ with query cut-off time t (Def. 4.2.1);

  Down-sample $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ with k, B_1, B_2 (Sec 4.3.2);

  Initialize labelling vector $\boldsymbol{c} \in \mathbb{R}^F$;

  Set center node s with initial representation, $\boldsymbol{h}_s^0 \leftarrow \boldsymbol{c}$;

  Set initial representations, $\boldsymbol{h}_v^0 \leftarrow \mathbf{0}\ \forall v \neq s \in \mathcal{V}_t$

  Get node representations $\boldsymbol{h}_v^R \leftarrow \textsc{Gnn}(R, \mathcal{G}_t)\ , \forall v \in \mathcal{V}_t$ (Eq. 4.2, 4.3 and 4.4);

  $[\phi(q)]_v \leftarrow \textsc{Mlp}(\boldsymbol{h}_v^R)\ , \forall v \in \mathbb{D}$

  $loss \leftarrow loss + \mathcal{L}(\phi(q), v^*)$ `// Ranking Loss (Eq. 4.5)`

end

return $loss/m$

B_1, B_2's, which demonstrate the robustness of TGRank to the downsampling strategy (See Sec. 4.5.4). Overall, the training pipeline is shown in Alg. 3.

4.3.3 Expressive and Inductive Ranking

Although the encoding procedure of TGRank is similar to standard GNNs, what makes TGRank substantially good for ranking is the initializing strategy in Sec. 4.3.2, where the center node and other nodes are labeled differently. Here, we provide a theoretical justification of the increased expressiveness of TGRank for ranking problems in temporal networks. We generalize the proof in [78], while the new technical challenge is to deal with multi-edges and timestamps that the previous analysis on static simple graphs cannot handle.

For notational simplicity, we temporarily do not consider any downsampling discussed in Sec. 4.3.2 and assume all nodes in the graph are ranking candidates. We first define a class of center nodes where R-layer standard GNNs without the labeling strategy cannot the rank candidate nodes for these center nodes properly.

Definition 4.3.2 (R-GNN Troublesome Center Node)**.** *Consider on a temporal graph $\mathcal{G}_t$ at time t. Define a R-GNN troublesome center node as the node who has at least one historical neighbor node v_j and a non-neighbor node v_k such that the representation of v_j learnt by running R-layer standard GNNs without the labeling strategy on the temporal graph $\mathcal{G}_t$ is the same as the representation of v_k.*

Obviously, for a R-GNN troublesome center node, the two other nodes that cause trouble to standard GNNs can be easily distinguished when using the labeling strategy. Running just *1-layer* GNN can diffuse the center node label to the neighbor node v_j but not the non-neighbor node v_k, which distinguish the two nodes. We prove that under certain conditions there could a lot of R-GNN troublesome center nodes in Thm. 4.3.1.

Theorem 4.3.1 (Expressivity)**.** *Consider a non-attributed temporal graph $\mathcal{G}_t$ with n nodes at time t. Suppose there are at most $\tau - 1$ distinct timestamps and the degree of each node in the graph is between 1 and $\mathcal{O}\left((\log_\tau n)^{\frac{1-\epsilon}{R+1}}\right)$ for any constant $\epsilon > 0$, then there exists at least $\omega(\frac{n^\epsilon}{\log_\tau n})$ R-GNN troublesome center nodes in $\mathcal{G}_t$.*

Proof. Our proof is inspired by the proof of Theorem 2 in [78]. However, additional effort to deal with the multi-edges and timestamps is needed. The proof has two steps. First, we would like to show that there are $\omega(n^\epsilon)$ nodes that cannot be distinguished by k-layer GNNs without the labeling strategy. Then, we prove that among these nodes, there are at least $\omega(\frac{n^\epsilon}{\log n})$ nodes whose 1-hop neighbors are all k-GNN troublesome center nodes. Also, note that in $\mathcal{G}_t$, timestamps can be viewed as edge features. When we say two graphs are isomorphic, not only the edges but also the features on the edges are needed to match.

Step 1. Recall $\mathcal{G}_{t,s}^k$ is the k-hop t-temporal neighborhood around s. As each node is with degree $d = \mathcal{O}(\log_\tau^{\frac{1-\epsilon}{k+1}} n)$, then the number of nodes in $\mathcal{G}_{t,s}^k$, denoted by $|\mathcal{V}(\mathcal{G}_{t,s}^k)|$, satisfies

$$|\mathcal{V}(\mathcal{G}_{t,s}^k)| \leq \sum_{i=0}^{k-1} d^i = \mathcal{O}(d_m^k), \text{ where } d_m = \log_\tau^{\frac{(1-\epsilon)}{k+1}} n.$$

We set the max $K = \max_{s \in \mathcal{V}_t} |\mathcal{V}(\mathcal{G}_{t,s}^k)|$ and thus $K = \mathcal{O}(d_m^k)$.

Now we expand subgraphs $\mathcal{G}_{t,s}^k$ to $\bar{\mathcal{G}}_{t,s}^k$ by adding $K - |\mathcal{V}(\mathcal{G}_{t,s}^k)|$ independent nodes for each node $s \in \mathcal{V}_t$. Then, all $\bar{\mathcal{G}}_{t,s}^k$ have the same number of nodes, which is K, though they may not be connected graphs.

Next, we consider the number of non-isomorphic graphs in $\{\bar{\mathcal{G}}_{t,s}^k | s \in \mathcal{V}_t\}$. Because in these graphs, the degree of each node is bound by d_m. Each node in these graphs at most has $O((\tau K)_m^d)$ different ways to be connected to the rest of the corresponding graph (Here we allow multi-edges and edges with timestamps). As each graph contains K nodes, the number of non-isomorphic graph structures in this set is at most $O((\tau K)^{Kd_m}) = O((\tau d_m)^{kd_m^{k+1}}))$. As $d_m = \log_\tau^{\frac{(1-\epsilon)}{k+1}} n$ and k is a constant, we have $O((\tau d_m)^{kd_m^{k+1}}) \leq O(n^{1-\epsilon})$. So, the number of non-isomorphic graphs in $\{\bar{\mathcal{G}}_{t,s}^k | s \in \mathcal{V}_t\}$ is at most $O(n^{1-\epsilon})$.

Therefore, due to the pigeonhole principle, there exist $n/o(n^{1-\epsilon}) = \omega(n^\epsilon)$ many nodes v whose $\bar{\mathcal{G}}_{t,s}^k$ are isomorphic to each other. Denote the set of these nodes as $\mathcal{V}_{iso}$. Next, we focus on looking for k-GNN troublesome center nodes from the neighbors of nodes in $\mathcal{V}_{iso}$.

Step 2. Let us partition $\mathcal{V}_{iso} = \cup_{i=1}^q \mathcal{V}_i$ so that for all nodes in $\mathcal{V}_i$, they share the same first-hop neighbor sets. Then, consider any pair of nodes u, v such that u, v are from different $\mathcal{V}_i$'s. Since u, v share identical k-hop t-temporal neighborhood structures, a k-layer standard GNN will give them the same representation. Then, we may pick one u's first-hop neighbor w that is not v's first-hop neighbor. We know such w exists by the definition of $\mathcal{V}_i$. As w is u's first-hop neighbor and is not v's first-hop neighbor, w is a k-GNN troublesome center node because without labeling w, u, v will have the same representation given by a k-layer standard GNN. So, we just need to count w. Based on the partition V_{iso}, we know the number of such k-GNN troublesome center nodes is at least $q - 1$. We now derive a lower bound of q.

Because of the definitions of the partition, $\sum_{i=1}^q |\mathcal{V}_i| = |\mathcal{V}_{iso}| = \omega(n^\epsilon)$ and the size of each $\mathcal{V}_i$ satisfies

$$1 \leq |\mathcal{V}_i| \leq d_m = \mathcal{O}(\log_\tau^{\frac{(1-\epsilon)}{k+1}} n).$$

So, a lower bound of q is just $\omega(\frac{n^\epsilon}{d_m}) \geq \omega(\frac{n^\epsilon}{\log_\tau n})$, which concludes the proof. □

Besides the better expressivity, we may also show the inductive property of TGRank. As claimed in Thm. 4.3.2, TGRank guarantees making the same ranking prediction as along as two k-hop temporal neighborhood are symmetric. Thm. 4.3.2 is easy to prove due to the permutation equivariance of GNNs and the time shift invariance of time encoding. Many GRL-based recommendation models are not inductive because they associate each node with a vector that can be trained differently across nodes.

Theorem 4.3.2 (Inductive Ranking)**.** *Consider two queries $q_i = (s_i, \mathbb{D}_i, t_i)$, $i \in \{1, 2\}$. Denote the two temporal k-hop neighborhoods for these two queries $\mathcal{G}^k_{t_i,s_i} = (V_i, E_i)$, $i \in \{1, 2\}$. Suppose these two queries and these two neighbors are symmetric in the sense that there exists a bijective mapping between the node sets $\pi : V_1 \to V_2$ such that (1) $\pi(s_1) = s_2$; (2) $\mathbb{D}_2 = \{\pi(d) | d \in \mathbb{D}_1\}$; (3) for every edge $(u, v, t) \in E_1$, $(\pi(u), \pi(v), t - t_1 + t_2) \in E_1$, and for every edge $(u, v, t) \in E_2$, $(\pi^{-1}(u), \pi^{-1}(v), t - t_2 + t_1) \in E_2$. Then, TGRank will output the same relevance scores for $\mathbb{D}_1$ and $\mathbb{D}_2$, i.e., $[\phi(q_1; \mathcal{G}^k_{t_1,s_1})]_u = [\phi(q_2; \mathcal{G}^k_{t_2,s_2})]_v$ for all $u \in \mathbb{D}_1, v \in \mathbb{D}_2$ and $v = \pi(u)$.*

4.3.4 Complexity Analysis

Let's assume that each method uses a k-hop subgraph to determine a node representation and the complexity of using GNN to compute such a represention is linear w.r.t. the size of the subgraph. The complexity of computing a representation for each query $q = (s, \mathbb{D}, t)$ depends on: (i) the number of candidates in $\mathbb{D}$, and (ii) the size of subgraph to determine a node representation via GNNs, which we will refer to as $\mathcal{G}^k_{t,v}$ for node v. We denote the size of the graph $|\mathcal{G}^k_{t,v}| = \max\{|\mathcal{V}^k_{t,v}|, |\mathcal{E}^k_{t,v}|\}$.

TGRank. We take the k-hop subgraph around s and add $\mathbb{D}$ to that set. Since the GNN with the labeling trick makes inference over this set just once, the overall complexity is $\mathcal{O}(|\mathcal{G}^k_{t,s}| + |\mathbb{D}|)$.

CAWN [68]. For each $d \in \mathbb{D}$, CAWN computes M walks of length k from d and s respectively and applies the labeling trick over that set of nodes to compute an edge representation. Assuming those random walks approximate the subgraphs $\mathcal{G}^k_{t,s}$ and $\mathcal{G}^k_{t,d}$, the complexity is $\mathcal{O}\left((|\mathcal{G}^k_{t,s}| + |\mathcal{G}^k_{t,d}|) \cdot |\mathbb{D}|\right)$.

TGN [75]. For each $d \in \mathbb{D}$, TGN will compute a representation for s and d respectively with a k-hop neighborhood. While GNNs on static graphs can compute node representations jointly, TGN needs to sample edges over time for each node independently. Thus, with the current TGN, the overall complexity will be $\mathcal{O}(|\mathcal{G}^k_{t,s}| + |\mathcal{G}^k_{t,d}| \cdot |\mathbb{D}|)$.

This shows that as the size of the candidate set grows, CAWN will be much less efficient than TGRank, which amortizes computation over a joint set rather than compute representations independently for each pair (source+one candidate). Theoretically, the complexity of TGN and CAWN are the same, but TGN often samples from direct neighbors ($k = 1$) to compute representations, while CAWN needs much longer walks $k > 1$ to keep good performance. This empirically leads to increased run times for CAWN during inference (see experiments in Sec 4.5.5). However, TGRank can still utilize larger subgraphs $\mathcal{G}^k_{t,s}$ with $k > 1$ during training as the optimization w.r.t all candidates is done jointly. So, empirically, in fact we observe that TGRank is comparable (if not better) to TGN and is much faster than CAWN (see Table 4.5).

4.4 Experimental Setting

Table 4.2. Dataset Statistics

	Reddit	Wikipedia	MOOC	LastFM	Enron	UCI
# nodes & attributes	10,984 & 172	9,227 & 172	7,144 & 0	1,980 & 0	184 & 0	1,899 & 0
# edges & attributes	672,447 & 172	157,474 & 172	411,749 & 4	1,293,103 & 0	125,235 & 0	59,835 & 0
# transductive & inductive test edges	100,867 & 21,470	23,621 & 11,715	61,763 & 29179	193,966 & 98,442	18,785 & 4,206	8,976 & 5,932
coverage of positive candidates	99.47 %	93.73 %	99.68 %	99.81 %	95.05 %	96.63 %
coverage of entire node set	4.10 %	3.17 %	4.70 %	55.12 %	52.05 %	50.66 %

We use six real-world temporal graph datasets for evaluation: Reddit [66], Wikipedia [66], MOOC [66], LastFM [66], Enron [68] and UCI [200]. In Reddit, Wikipedia, MOOC and LastFM users interact with items forming a bipartite graph with edges having continuous timestamps, with/without edge features. Enron and UCI are unipartite communication networks between users with edges having continuous timestamps, no additional attributes. Dataset statistics are shown in Table 4.2 and detailed descriptions are provided below.

4.4.1 Dataset Description

For our empirical evaluation we use six real world benchmark datasets. They consist of Reddit, Wikipedia, MOOC, LastFM, Enron and UCI. Table 4.2 shows basic statistics and below we list brief descriptions. Coverage of both positive candidates and entire node sets are calculated on the test set for each dataset. Specifically, for each source in test, its k-hop historical subgraph is sampled with constraint parameters $k = 3, B_1 = B_2 = 50$ and coverage statistics are calculated based on this sampled graph and averaged across all sources in the test set for a dataset.

- **Reddit** is a bipartite interaction graph formed over a span of 30 days. Here, users and sub-reddits form two classes of nodes and an edge occurs when a user interacts with a sub-reddit. Interactions contain continuous timestamps and textual LIWC attributes related to the post that a user writes to the sub-reddit. The processed dataset is obtained from *http://snap.stanford.edu/jodie/reddit.csv*

- **Wikipedia** is a bipartite interaction graph formed over a span of 30 days. Here, users and wiki pages form two classes of nodes and an edge occurs when a user edits a wiki page. Interactions contain continuous timestamps and textual LIWC attributes related to the edit that a user made to the wiki page. The processed dataset is obtained from *http://snap.stanford.edu/jodie/wikipedia.csv*

- **MOOC** is a bipartite user action graph formed by two classes of nodes: students and course content. Edges are formed when students interact with a course content. Interactions contain continuous timestamps and vector attributes related to the type of event (e.g., access course chapter, view course video). The processed dataset is obtained from *http://snap.stanford.edu/jodie/mooc.csv*

- **LastFM** is a bipartite user interaction graph formed by who-listens-to-which song on an online music platform. Edges are formed when users listen to a song. Interactions contain continuous timestamps without any attributes. The processed dataset is obtained from *http://snap.stanford.edu/jodie/lastfm.csv*

- **Enron** is a communication network formed by email exchanges between employees of an organization over several years. The processed dataset is obtained from *https://www.cs.cmu.edu/~./enron/*
- **UCI** is a communication network arising from messages sent between users of an online community of students over a span of six months. The processed dataset is obtained from *http://konect.cc/networks/opsahl-ucsocial/*

4.4.2 Baselines

We compare our framework TGRank against representative T-GRL methods designed for continuous time graphs: JODIE [66], DyRep [79], TGAT [74], TGN [75] and CAWN [68]. The baselines we adopt have shown to be consistently better than network embedding methods [99], [100], [176] and temporally adapted static GNNs (e.g., GAT [47], GraphSAGE [46]) on same datasets we have chosen [cf. 74], [75]. Thus, we focus on comparing TGRank with only the aforementioned methods. Additionally, for having more comparative baselines, we extend TGAT and TGN which were originally proposed with a binary classification based link prediction loss to our listwise ranking loss in Eq. 4.5. We term them TGAT-LW and TGN-LW respectively. For both the aforementioned baselines we sample the candidates for each source node using the exact same strategy as used by TGRank from Sec. 4.3.2 during training to keep comparison fair.

4.4.3 Evaluation Tasks, Protocol and Metrics.

The general setup for all datasets is as follows. First, for a dataset in time range $[0, T]$ we follow [74], [75] and perform a 70%-15%- 15% chronological split of the dataset for defining its training $[0, T_{train})$, validation $[T_{train}, T_{val})$ and testing $(T_{val}, T]$ intervals. Secondly, following previous works [68], [74], [75] we randomly sample 10% of all nodes and consider them as "new" nodes (used for inductive task explained later). In the **transductive task**, the training, validation and testing sets involve all interactions occurring in their respective split intervals. In the **inductive task**, for a testing query, neither the center node nor the true candidate in the testing set is not observed during training. We generate the

queries $q = (s, \mathbb{D}, t)$ as follows. Given a positive edge (s, v, t), we consider all nodes in the k-hop historical subgraph $\mathcal{G}^k_{t,s}$ and adopt the downsampling strategy in Sec. 4.3.2 with hyperparameters B_1, B_2. We set the obtained node set as $\mathbb{D}$. We use mean reciprocal rank (MRR) and mean Hits@K to evaluate the models ranking ability over all evaluation queries. More details are provided below.

4.4.4 Detailed Temporal Graphs Downsampling in Experiments

Our real-world temporal network evaluation sets only consist of a single ground truth candidate for a source node at a certain evaluation time, so we consider the following procedure to build the query and its ground truth ranking list. Specifically, given an evaluation interaction (s, v^*, t) where s is the source, v^* is its single ground truth candidate and t is the evaluation time of the interaction, to build its query $q = (s, \mathbb{D}, t)$, we consider the k-hop historical subgraph $\mathcal{G}^k_{s,t}$ of the source s and build the candidate list $\mathbb{D} = (d_1, d_2, \ldots, d_n)$, where $d_j \in \mathcal{G}^k_{s,t}$. Additionally, to control the quality and quantity of the candidates that are used for each source in evaluation, we follow the same downsampling stategy like in training and consider constraints on $\mathcal{G}^k_{s,t}$ with parameters k, B_1 and B_2 that represent the total number of hops, number of recent neighbors sampled for each node per hop and number of most recent edges sampled between two nodes respectively. Note that these evaluation parameters can be different from that used for training and later in experiments we will show the interplay between them. An important point to note is that v^* may or may not be present in the constrained $\mathcal{G}^k_{s,t}$, but is always included in $\mathbb{D}$. We use mean reciprocal rank (MRR) and mean Hits@K to evaluate the models ranking ability over all evaluation queries.

4.4.5 Additional Implementation Details

Hyper-parameters of TGRank

We set the embedding dimension for all datasets to 128. The time encoding dimension is set to 128 for all datasets. Train batch size is chosen from $\{32, 64, 128\}$. The number of encoder layers R is chosen from $\{1, 2, 3\}$. Gradient optimization is carried out using Adam with learning rate of $1e-4$ for all datasets. The upper cap for number of epochs is set to

25. Models are selected using early stopping with patience set to 5 rounds. Validation MRR performance is chosen to guide the early stopping with a tolerance of $1e-10$. Dropout layers with drop probability of 0.1 is used during training for the Message, Update and Ranking MLP function. Layer normalization is used for the update function.

Baselines

Specifically for implementing TGAT-LW and TGN-LW, we adapt the code available at https://github.com/twitter-research/tgn to include our listwise loss. JODIE, DyRep, TGAT are run using the code provided at the same aforementioned link.

Resources

For calculating the wall-clock time, a single Nvidia GeForce GTX 1080 Ti (12GB) card is used along with 6 Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz and a total of 24 GB RAM.

4.4.6 Classification based Loss Formulation

Previous T-GRL encoders produced latent representations for individual nodes. Thus, in these methods the goal of the decoder is to map pairs of such representations into a single latent space that represents a score for a link. TGAT [74] used a parameter free decoder which just computes similarity of two latent node representation using dot products. TGN [75] use a MLP that takes a vector concatenation of two node representations to provide a link representation. For e.g., if $\phi(u,t)$ represents the node representation for a node u at time t as generated by TGN,

$$\boldsymbol{z}_{u,v}(t) = \textsc{Mlp}\left(\phi(u,t) \;\|\; \phi(v,t)\right) \tag{4.6}$$

where $\boldsymbol{z}_{u,v}(t)$ represents a score for link (u,v) at time t.

Now, the goal is to classify if two nodes interact (or form links) with each other at a given time. In other words, link prediction is cast as a binary classification problem. Since the

dataset contains only positive observation, negative observations are sampled by corrupting the positives. Finally, binary cross entropy loss is used as the training objective:

$$\mathcal{L}(s, v^*, t) = -\log(\sigma(\boldsymbol{z}_{s,v^*}(t))) - \log(1 - \sigma(\boldsymbol{z}_{s,v'}(t))) \tag{4.7}$$

where node v' is randomly sampled from $\mathcal{V}_t$ and $\sigma(\cdot)$ is the sigmoid function. While the above formulation is commonly employed for training T-GRL methods [68], [74], [75], [79], it can be inconsistent with the goal of ranking evaluation. Specifically, consider a ranking evaluation with metrics such as MRR, given a true link (s, v^*), we want to rank v^* higher compared to other nodes in a list of candidates $\mathbb{D}$.

4.5 Experimental Analysis

In this section, we conduct empirical evaluations of TGRank over real-world temporal networks.

Table 4.3. Transductive and Inductive temporal link prediction performance in MRR (mean ± std.) and Hits@5 (mean ± std.). TLE signifies that a 12 hour time limit exceeded during link ranking inference.

Task	Method	Reddit		Wikipedia		MOOC		LastFM		Enron		UCI	
		MRR	Hits@5	MRR	Hits@5	MRR	Hits@5	MRR	Hits@5	MRR	Hits@5	MRR	Hits@5
Transductive	DyRep	0.519±0.007	0.642±0.011	0.473±0.013	0.606±0.009	0.122±0.016	0.172±0.013	0.065±0.018	0.084±0.010	0.200±0.013	0.297±0.018	0.029±0.009	0.027±0.010
	JODIE	0.376±0.003	0.539±0.008	0.287±0.006	0.514±0.008	0.110±0.013	0.156±0.020	0.031±0.012	0.034±0.016	0.104±0.018	0.126±0.021	0.023±0.008	0.019±0.007
	TGAT	0.413±0.006	0.536±0.007	0.497±0.012	0.737±0.006	0.188±0.009	0.264±0.008	0.098±0.011	0.122±0.014	0.252±0.007	0.424±0.012	0.198±0.004	0.288±0.005
	TGN	0.552±0.004	0.678±0.010	0.575±0.015	0.795±0.013	0.208±0.012	0.329±0.011	0.089±0.008	0.095±0.013	0.242±0.008	0.397±0.009	0.203±0.001	0.280±0.006
	TGAT-LW	0.594±0.007	0.693±0.009	0.577±0.040	0.660±0.011	0.214±0.003	0.275±0.001	0.109±0.004	0.131±0.011	0.210±0.008	0.325±0.016	0.187±0.007	0.256±0.006
	TGN-LW	0.602±0.003	0.785±0.005	0.701±0.011	0.831±0.008	0.284±0.002	0.404±0.008	0.134±0.005	0.159±0.006	0.380±0.018	0.560±0.002	0.272±0.018	0.311±0.019
	CAWN	TLE	TLE	**0.795**±0.022	0.868±0.018	0.206±0.003	0.265±0.016	TLE	TLE	0.386±0.011	0.614±0.003	0.518±0.005	0.683±0.008
	TGRank	**0.663**±0.005	**0.821**±0.009	0.792±0.006	**0.870**±0.001	**0.321**±0.001	**0.436**±0.002	**0.165**±0.009	**0.230**±0.010	**0.414**±0.007	**0.650**±0.006	**0.685**±0.002	**0.781**±0.004
Inductive	DyRep	0.388±0.008	0.528±0.007	0.474±0.004	0.609±0.007	0.125±0.014	0.173±0.008	0.112±0.015	0.152±0.009	0.137±0.014	0.200±0.016	0.035±0.015	0.040±0.008
	JODIE	0.042±0.010	0.048±0.009	0.264±0.009	0.459±0.008	0.069±0.012	0.085±0.013	0.055±0.012	0.061±0.017	0.119±0.010	0.162±0.009	0.029±0.009	0.028±0.011
	TGAT	0.374±0.013	0.493±0.013	0.523±0.007	0.746±0.010	0.169±0.004	0.230±0.007	0.117±0.013	0.134±0.011	0.201±0.008	0.360±0.011	0.221±0.007	0.321±0.004
	TGN	0.537±0.009	0.650±0.008	0.604±0.007	0.786±0.006	0.228±0.005	0.355±0.010	0.120±0.010	0.126±0.021	0.219±0.005	0.376±0.008	0.230±0.006	0.315±0.004
	TGAT-LW	0.429±0.014	0.538±0.009	0.608±0.041	0.685±0.010	0.201±0.004	0.256±0.002	0.124±0.012	0.178±0.008	0.185±0.007	0.296±0.010	0.217±0.009	0.297±0.008
	TGN-LW	0.581±0.011	0.703±0.014	0.708±0.005	0.819±0.004	0.272±0.003	0.378±0.009	0.136±0.009	0.162±0.007	0.379±0.010	0.584±0.008	0.310±0.019	0.356±0.020
	CAWN	TLE	TLE	**0.798**±0.008	0.847±0.007	0.200±0.010	0.261±0.008	TLE	TLE	0.395±0.013	0.609±0.007	0.521±0.004	0.681±0.009
	TGRank	**0.608**±0.004	**0.761**±0.008	0.797±0.008	**0.866**±0.009	**0.299**±0.002	**0.403**±0.005	**0.194**±0.008	**0.273**±0.005	**0.432**±0.008	**0.648**±0.006	**0.684**±0.003	**0.782**±0.002

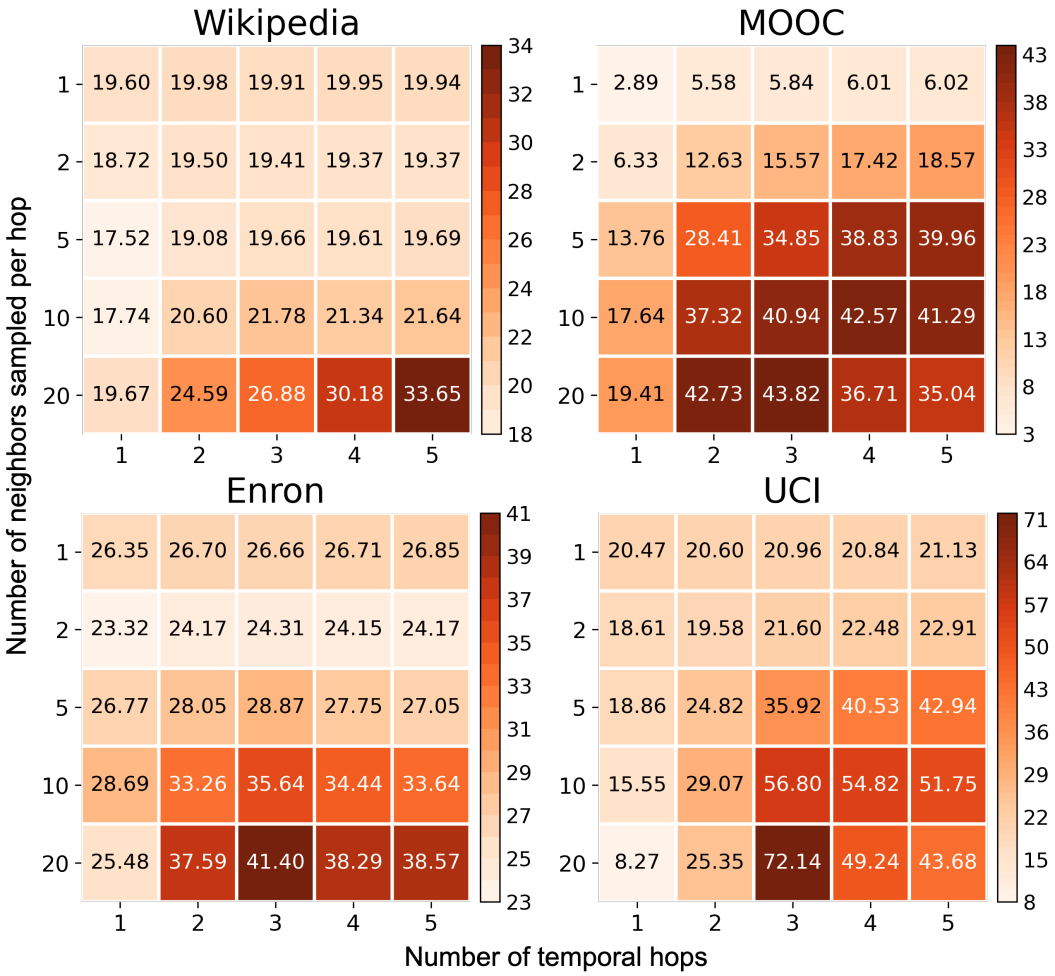

Figure 4.3. Heatmap values indicate % gain in MRR of TGRank over TGN for different parameters used to generate candidates from the historical subgraph of a source during ranking evaluation. The x-axis shows the number of hops k considered and y-axis $B_1 (= B_2)$ jointly shows the number of most recent neighbors sampled per node per hop (B_1) and the number of most recent edges between two nodes (B_2).

4.5.1 Temporal Interaction Ranking Results

In Table 4.3 we compare TGRank with baselines for both transductive and inductive temporal interaction ranking tasks. All methods are evaluated on the same testing query-true list pairs where candidates are generated by setting sampling parameters $k = 3$, $B_1 = B_2 = 20$.

Focusing on the transductive task, it is clear from Table 4.3 that our method TGRank significantly outperforms all baselines on all datasets. On average, for attributed datasets (Reddit, Wikipedia and MOOC) our method improves over the strongest baseline by 7.6% in MRR and 4.21% in Hits@5. While, for non-attributed datasets (LastFM, Enron and UCI) we witness that increased expressivity of TGRank helps and we see average improvement of 61.08% in MRR and 70.60% Hits@5 compared to TGN-LW and average improvement of 19.7% in MRR and 10.5% Hits@5 over CAWN for Enron and UCI datasets.

By actively sampling historical graphs, viz., TGAT, TGN, TGAT-LW, TGN-LW, CAWN and TGRank outperform than other methods not fully utilizing it (JODIE and DyRep). This highlights that historical graph is key to the interaction ranking problem. Moreover, our method TGRank that is more expressive due to the label trick is consistently better than all the baselines. We stress that the gains we witness for TGRank are higher in datasets devoid of attributes as structural and temporal patterns play an even bigger role. Lastly, methods using a listwise ranking loss are better than methods trained using binary classification loss by comparing TGN-LW and TGAT-LW with their original counterparts TGN and TGAT, respectively.

Moving to the inductive task in Table. 4.3, it is clear that TGRank shows improved performance over baselines on all datasets. Specifically, compared to the best baseline, we witness averaged gains of 4.8% in MRR and 5.6% Hits@5 over attributed datasets. Over non-attributed datasets, we see gains of 59.08% in MRR and 66.37% in Hits@5 compared to TGN-LW (less expressive). Whereas, compared to CAWN, we witness average gains of 20.2% in MRR and 10.6% in Hits@5 for Enron and UCI datasets combined.

Once again, even in the inductive task, the gains we witness are considerably higher for non-attributed datasets. This additionally supports our claim that TGRank can be effectively used for inductive ranking tasks (Thm. 4.3.2). While TGAT, TGN and their ranking counterparts TGAT-LW, TGN-LW are also inductive by design, they fall short in comparison to our method TGRank, primarily due to their reduced expressivity (ambiguity problem). While CAWN can maintain high expressivity, it incurs excessive run-times and for large datasets like Reddit and LastFM, we were unable to finish the inference on all test

queries within a 12 hour time window (hence a TLE). More details on wall run times are provided in Sec 4.5.5.

We also compare different methods via the non-ranking evaluation protocol, i.e., a binary classification task to predict when an edge exists or not, as commonly adopted by previous works in Table 4.6. The results further demonstrate the effectiveness of the labeling method and our encoder.

4.5.2 Ranking evaluation for different ways of candidate selection

One might argue that the results in Table 4.3 might not give a complete picture of the effectiveness of TGRank as the quality and quantity of candidates selected during ranking evaluation is fixed for a source in the query. Thus, we compare TGRank with TGN over different evaluation parameters $0 < k \leq 5$, and $0 < B_1 = B_2 \leq 20$. Note that each value taken by the parameters will result in a different ranking evaluation over all data points in testing for a dataset. Fig. 4.3 shows heatmaps of averaged % gain (using MRR) of TGRank over TGN under different evaluation parameters. The take away is that TGRank is consistently better than TGN as we see positive gain throughout the chosen evaluation parameter space. Noticeably greater improvements are seen when candidates are selected further away from the source (higher k) and number of candidates increase (higher B_1) per source.

4.5.3 Ablation Study

Table 4.4. Ablation results for TGRank using transductive ranking. Here candidate nodes are sampled from temporal neighborhood

No.	Ablation	Wikipedia		MOOC		Enron		UCI	
		MRR	Hits@5	MRR	Hits@5	MRR	Hits@5	MRR	Hits@5
1	**TGRank (original)**	0.792±0.006	0.870±0.001	0.321±0.001	0.436±0.001	0.414±0.007	0.650±0.006	0.685±0.002	0.781±0.004
2	replace listwise loss with pointwise loss	0.772±0.008	0.859±0.009	0.302±0.004	0.405±0.003	0.348±0.003	0.572±0.001	0.372±0.002	0.566±0.003
3	randomize timestamps before encoding	0.767±0.002	0.860±0.003	0.231±0.003	0.344±0.006	0.386±0.002	0.621±0.002	0.663±0.004	0.748±0.009
4	coalesce temporal edges, time counts	0.741±0.005	0.863±0.004	0.234±0.002	0.352±0.002	0.405±0.002	0.620±0.008	0.669±0.003	0.742±0.007
5	no labelling trick	0.609±0.005	0.735±0.004	0.228±0.001	0.332±0.004	0.341±0.002	0.559±0.009	0.389±0.006	0.593±0.007
6	replace time encoding with just timestamp	0.734±0.001	0.813±0.001	0.066±0.005	0.067±0.002	0.176±0.002	0.253±0.003	0.058±0.001	0.065±0.001

We further conduct ablation analysis on attributed (Wikipedia, MOOC) and non-attributed (Enron, UCI) datasets to probe into the critical components of our method. Table 4.4 shows

the results. Row Ab.1 is our original TGRank method. In row Ab.2, we replace the listwise ranking loss with a binary classification loss, we witness a drop in performance (compared to row 1) on all datasets, with the drop being prominent for non-attributed datasets. In row Ab.3 timestamps are randomized before encoding during training. The results show that breaking the time invariance property used in Theorem 4.3.2 does have a significant negative effect on the performance across all datasets. In row Ab.4, multi-edges between two nodes are coalesced and time information is replaced with counts. This leads to a decay in performance that is more prominent in attributed datasets compared to non-attributed datasets. We reason that for Enron and UCI, which are communication networks, the number of interactions between two nodes which we provide as counts is a very important signal in ranking future interactions. On the other hand in attributed datasets, when we coalesce such multi-edges, their attributes are averaged and some crucial information loss is bound to happen and therefore we witness a steeper decay in performance. In row Ab.5, we show performance of TGRank without initial labels (i.e., limiting the expressivity of the model). The results show that not labelling the source differently from candidates during our label diffusion procedure, performance across all datasets is severely impacted. This clearly validates our motivation that candidate representations should be "aware" of the source node when used for ranking and provides empirical evidence of the increased expressivity via label diffusion as characterized by Thm. 4.3.1. Finally, in row Ab. 6, we drop the learned time encoding from TGRank and simply use the original time stamps on the edges. The results provides experimental evidence to the importance of using time encoding to model the timestamps and how our method can effectively use it.

4.5.4 Hyperparameter Sensitivity

We conduct an experiment to vary the quality and quantity of candidates per source in a query and measure its effect on evaluation performance where a fixed number and type of candidates are given. The analysis is shown in Fig 4.4. Specifically, during training, the candidates are generated from the k-hop historical subgraph around a center node with the following parameters: number of hops $0 < k \leq 3$, number of most recent neighbors sampled

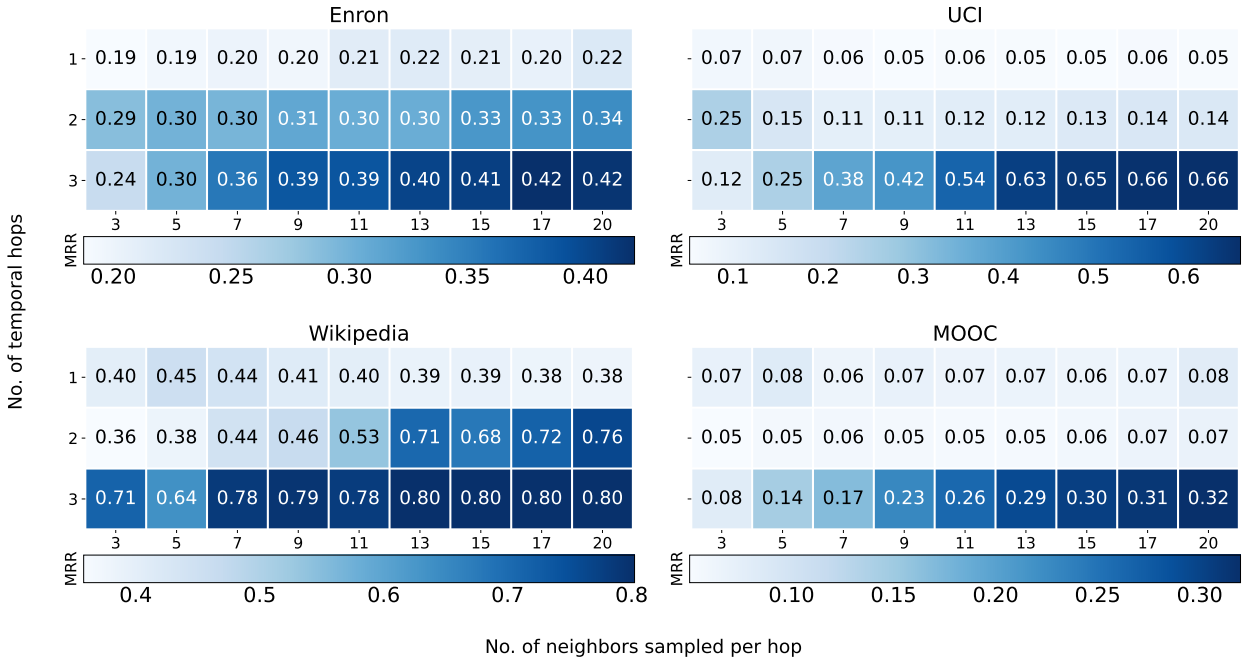

Figure 4.4. Heatmap shows the effect of using different ways of selecting candidates during training on a fixed 3-hop, 20 most recent neighbors per node per hop and 20 most recent edges between nodes type sampling during testing evaluation. Numbers indicate averaged test MRR.

per node per hop B_1. As for testing, the sampling is fixed with $k = 3$, $B_1 = 20$. We fix number of most recent edges sampled between two nodes $B_2 = B_1$ for both training and testing. The results show that when the training candidate selection parameters approach that of the testing, the performance of TGRank becomes optimal. However, note that as long as the quality of candidates used in training match that of the candidates used in testing, comparative performance is achieved across all datasets (row 3 of heatmaps in Fig 4.4). This means that one can significantly reduce the size of the downsampled subgraph by placing more importance to the depth at which candidates are sampled rather than the number of candidates at each depth.

4.5.5 Run Time Comparisons

We provide the wall-clock times of TGN-LW, CAWN and TGRank on all the datasets we considered in Table 4.5. Both training and test (inference on transductive ranking test set)

Table 4.5. Wall times in seconds. Train represents time taken per training epoch averaged over first 5 epochs. Test represents time taken for inference on transductive test ranking queries.

Method	Reddit		Wikipedia		MOOC		LastFM		Enron		UCI	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
TGN-LW	1366.98	881.33	210.22	106.09	608.46	314.05	6701.49	2217.71	54.42	45.21	145.77	45.99
CAWN	1844.52	TLE (12h+)	292.64	2491.76	904.62	16180.02	2103.20	TLE (12h+)	192.18	438.01	91.40	1524.54
TGRank (ours)	644.65	706.73	102.96	68.24	518.41	252.94	807.06	1845.19	50.72	33.53	136.01	38.89

times are reported. Note that in TGN-LW for scalability only 1-layer graph convolutions are used. Ours uses 3-layers (Wikipedia, MOOC, Enron, UCI) and 2-layers (Reddit, LastFM). All methods listed use the same temporal sampling scheme for each dataset to find candidates so that comparison is fair. Firstly, comparing TGRank and TGN-LW, it is clear that TGRank achieves 2∼8× speed-up of TGN-LW in both train and test stages and the difference is clearly visible for large datasets like LastFM and Reddit. CAWN, incurs excessive runtimes during inference as it needs to sample long random walks between source and candidate nodes. For large datasets like Reddit and LastFM, CAWN was unable to complete the inference over all test queries within a 12 hour window resulting in a TLE (time limit exceeded). TGRank achieves 10∼65× speed-up compared to CAWN during inference with increased gains witnessed for large datasets like Reddit, MOOC and LastFM.

4.6 Additional Experimental Results

Fig. 4.5 shows the sensitivity analysis of our method. Here, we plot the sensitivity of the number of layers of encoder used in relation to the number of most recent neighbors sampled per node per hop. Across all four datasets, we witness that increasing the layer or in other words rounds of message passing for label diffusion has a strong positive correlation with validation MRR. While, traditional T-GRL methods like TGN and TGAT may suffer from over-smoothing issues as prevalent in standard GNNs. Moreover the authors of TGN have proposed that a single layer of their GNN based attention encoder is sufficient and comparable to larger layers [cf. 75, fig. 3 (b) p. 8]. Our use of initial node representations in the update functions of message passing may alleviate issues surrounding over-smoothing in temporal networks, although more study is required.

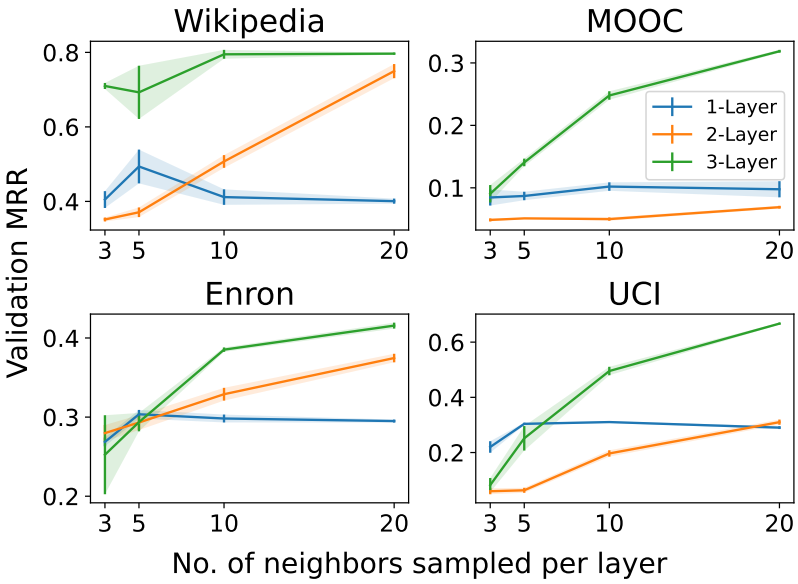

Figure 4.5. Sensitivity Analysis

4.6.1 On Interaction Ranking v.s. Prediction

Table 4.6. Transductive temporal link prediction performance in AP (mean ± std.)

Method	Reddit	Wikipedia	MOOC	LastFM	Enron	UCI
DyRep	97.98±0.1	94.59±0.2	79.57±1.5	69.24±1.4	77.68±1.6	54.60±3.1
JODIE	97.84±0.3	95.70±0.2	81.16±1.0	69.32±1.0	77.31±4.2	86.73±1.0
TGAT	98.12±0.2	95.34±0.1	64.36±3.3	54.77±0.4	68.02±0.1	77.51±0.7
TGN	98.68±0.1	**98.51±0.1**	82.10±0.6	78.80±0.5	78.05±1.2	90.40±1.2
TGRank	**98.83±0.2**	98.20±0.5	**83.47±0.5**	**79.53±0.4**	**83.65±0.9**	**92.14±0.8**

In Table 4.6 we also compare different methods via the non-ranking evaluation protocol, i.e., a binary classification task to predict when an edge exists or not, as commonly adopted by previous works [68], [74], [75]. Here, for all positive links in the evaluation set an equal number of negative links are sampled and the average precision (AP) metric is used to evaluate the models prediction ability. Even in this setting, where the baseline models were first proposed, one can witness that TGRank significantly outperforms in five out of six datasets. An interesting observation is that baseline methods show strong performance in terms of relatively high AP values for all datasets, but fail to provide similar strong

performance in our ranking tasks (see Table 4.3). This illustrates that using methods trained and evaluated on classification objectives for ranking tasks can lead to sub-par results, and gives further evidence to support the claim that training and evaluation objectives should match.

4.7 Discussion

In this chapter, we developed a novel Temporal Graph network for Ranking (TGRank) that optimizes a list-wise loss and incorporates a labeling trick to effectively and efficiently improve performance for link ranking tasks. We discussed the complexity/expressivity trade-off for previous methods that have either (i) increased expressivity (CAWN) or (ii) are more amenable to joint inference over candidate sets (TGN), but not both. Then, we showed how our labeling trick provably increases the expressivity of learned representations and as such leads to increased ranking accuracy. Our labeling approach allows us to efficiently infer representations over the candidate set jointly. In the experiment, we observe consistent, significant gains in ranking performance compared to the baselines, and we also achieve efficient run times.

5. ADVERSARIAL GRAPH CONTRASTIVE LEARNING

Graph representation learning (GRL) aims to encode graph-structured data into low-dimensional vector representations, which has recently shown great potential in many applications in biochemistry, physics and social science [9], [14], [39]. Graph neural networks (GNNs), inheriting the power of neural networks [43], [44], have become the almost *de facto* encoders for GRL [42], [140], [141], [201]. GNNs have been mostly studied in cases with supervised end-to-end training [45], [52], [53], [77], [80]–[82], where a large number of task-specific labels are needed. However, in many applications, annotating labels of graph data takes a lot of time and resources [83], [84], e.g., identifying pharmacological effect of drug molecule graphs requires living animal experiments [85]. Therefore, recent research efforts are directed towards studying self-supervised learning for GNNs, where only limited or even no labels are needed [84], [86]–[97].

Designing proper self-supervised-learning principles for GNNs is crucial, as they drive what information of graph-structured data will be captured by GNNs and may heavily impact their performance in downstream tasks. Many previous works adopt the edge-reconstruction principle to match traditional network-embedding requirement [98]–[101], where the edges of the input graph are expected to be reconstructed based on the output of GNNs [46], [86], [91]. Experiments showed that these GNN models learn to over-emphasize node proximity [93] and may lose subtle but crucial structural information, thus failing in many tasks including node-role classification [77], [101]–[103] and graph classification [83].

To avoid the above issue, graph contrastive learning (GCL) has attracted more attention recently [84], [87]–[90], [92], [93], [95]–[97]. GCL leverages the mutual information maximization principle (InfoMax) [104] that aims to maximize the correspondence between the representations of a graph (or a node) in its different augmented forms [84], [87]–[90], [94], [95]. Perfect correspondence indicates that a representation precisely identifies its corresponding graph (or node) and thus the encoding procedure does not decrease the mutual information between them.

However, researchers have found that the InfoMax principle may be risky because it may push encoders to capture redundant information that is irrelevant to the downstream tasks:

Redundant information suffices to identify each graph to achieve InfoMax, but encoding it yields brittle representations and may severely deteriorate the performance of the encoder in the downstream tasks [105]. This observation reminds us of another principle, termed information bottleneck (IB) [106]–[111]. As opposed to InfoMax, IB asks the encoder to capture the *minimal sufficient* information for the downstream tasks. Specifically, IB minimizes the information from the original data while maximizing the information that is relevant to the downstream tasks. As the redundant information gets removed, the encoder learnt by IB tends to be more robust and transferable. Recently, IB has been applied to GNNs [112], [113]. But IB needs the knowledge of the downstream tasks that may not be available.

Hence, a natural question emerges: *When the knowledge of downstream tasks are unavailable, how to train GNNs that may remove redundant information?* Previous works highlight some solutions by designing data augmentation strategies for GCL but those strategies are typically task-related and sub-optimal. They either leverage domain knowledge [87], [89], [95], *e.g.*, node centralities in network science or molecule motifs in bio-chemistry, or depend on extensive evaluation on the downstream tasks, where the best strategy is selected based on validation performance [89], [94].

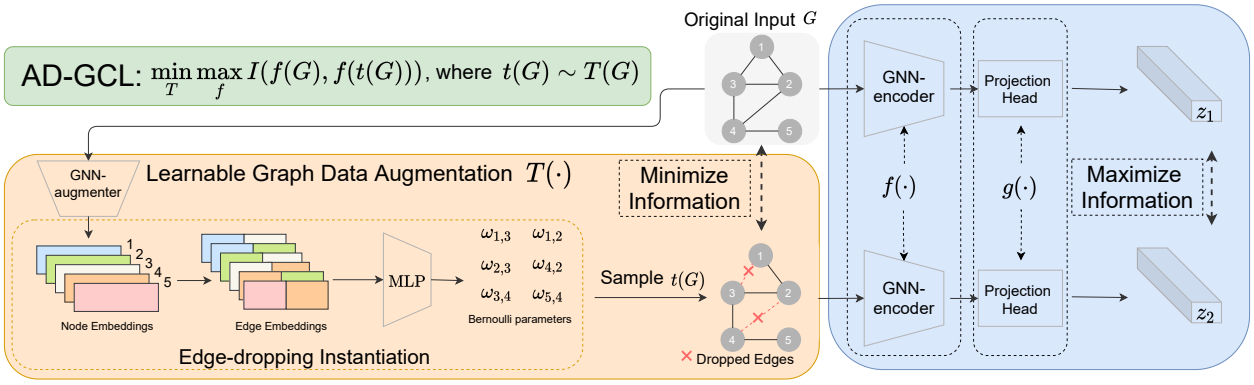

Figure 5.1. The AD-GCL principle and its instantiation based on learnable edge-dropping augmentation. AD-GCL contains two components for graph data encoding and graph data augmentation. The GNN encoder $f(\cdot)$ maximizes the mutual information between the original graph G and the augmented graph $t(G)$ while the GNN augmenter optimizes the augmentation $T(\cdot)$ to remove the information from the original graph. The instantiation of AD-GCL proposed in this work uses edge dropping: An edge e of G is randomly dropped according to Bernoulli(ω_e), where ω_e is parameterized by the GNN augmenter.

In this chapter, we approach this question by proposing a novel principle that pairs GCL with adversarial training, termed *AD-GCL*, as shown in Fig.5.1. We particularly focus on training self-supervised GNNs for graph-level tasks, though the idea may be generalized for node-level tasks. AD-GCL consists of two components: The first component contains a GNN encoder, which adopts InfoMax to maximize the correspondence/mutual information between the representations of the original graph and its augmented graphs. The second component contains a GNN-based augmenter, which aims to optimize the augmentation strategy to decrease redundant information from the original graph as much as possible. AD-GCL essentially allows the encoder capturing the minimal sufficient information to distinguish graphs in the dataset. We further provide theoretical explanations of AD-GCL. We show that with certain regularization on the search space of the augmenter, AD-GCL can yield a lower bound guarantee of the information related to the downstream tasks, while simultaneously holding an upper bound guarantee of the redundant information from the original graphs, which matches the aim of the IB principle. We further give an instantiation of AD-GCL: The GNN augmenter adopts a task-agnostic augmentation strategy and will learn an input-graph-dependent non-uniform-edge-drop probability to perform graph augmentation.

Finally, we extensively evaluate AD-GCL on 18 different benchmark datasets for molecule property classification and regression, and social network classification tasks in different setting viz. unsupervised learning (Sec. 5.5.2), transfer learning (Sec. 5.5.3) and semi-supervised learning (Sec. 5.5.4) learning. AD-GCL achieves significant performance gains in relative improvement and high mean ranks over the datasets compared to state-of-the-art baselines. We also study the theoretical aspects of AD-GCL with apt experiments and analyze the results to offer fresh perspectives (Sec. 5.6): Interestingly, we observe that AD-GCL outperforms traditional GCL based on non-optimizable augmentation across almost the entire range of perturbation levels.

5.1 Preliminaries

We first introduce some preliminary concepts and notations for further exposition.

5.1.1 The Mutual Information Maximization Principle

GCL is built upon the InfoMax principle [104], which prescribes to learn an encoder f that maximizes the mutual information or the correspondence between the graph and its representation. The rationale behind GCL is that a graph representation $f(G)$ should capture the features of the graph G so that representation can distinguish this graph from other graphs. Specifically, the objective of GCL follows

$$\text{InfoMax:} \quad \max_f I(G; f(G)), \quad \text{where } G \sim \mathbb{P}_{\mathcal{G}}. \tag{5.1}$$

where $I(X_1; X_2)$ denotes the mutual information between two random variables X_1 and X_2 [202]. The learnt encoder f can yield graph representations that are used for the downstream tasks.

Note that the encoder $f(\cdot)$ given by GNNs is not injective in the graph space $\mathcal{G}$ due to its limited expressive power [52], [53]. Specifically, for the graphs that cannot be distinguished by 1-WL test [51], GNNs will associate them with the same representations. 1-WL test is discussed in detail in the Section 2.4. In contrast to using CNNs as encoders, one can never expect GNNs to identify all the graphs in $\mathcal{G}$ based their representations, which introduces a unique challenge for GCL.

In this section, we introduce our adversarial graph contrastive learning (AD-GCL) framework and one of its instantiations based on edge perturbation.

5.2 Theoretical Motivation for Adversarial Graph Contrastive Learning (AD-GCL)

The InfoMax principle in Eq. 5.1 could be problematic in practice for general representation learning. Tschannen et al. have shown that for image classification, representations capturing the information that is entirely irrelevant to the image labels are also able to maximize the mutual information but such representations are definitely not useful for image classification [105]. A similar issue can also be observed in graph representation learning, as illustrated by Fig.5.2: We consider a binary graph classification problem with graphs in

the dataset ogbg-molbace [203]. Two GNN encoders with exactly the same architecture are trained to keep mutual information maximization between graph representations and the input graphs, but one of the GNN encoders in the same time is further supervised by random graph labels. Although the GNN encoder supervised by random labels still keeps one-to-one correspondance between every input graph and its representation (i.e., mutual information maximization), we may observe significant performance degeneration of this GNN encoder when evaluating it over the downstream ground-truth labels. More detailed experiment setup is given below.

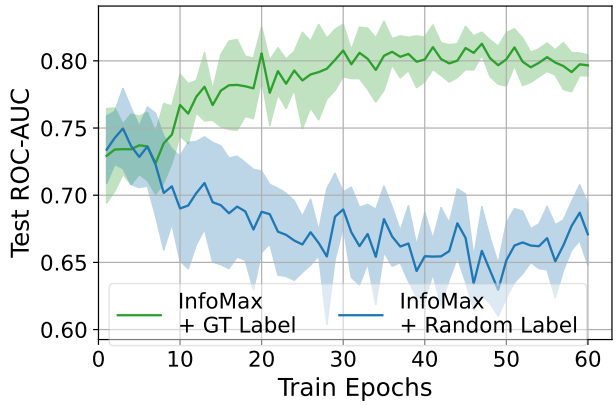

Figure 5.2. Two GNNs keep the mutual information maximized between graphs and their representations. Simultaneously, they get supervised by ground-truth labels (green) and random labels (blue) respectively.The curves show their testing performance on predicting ground-truth labels.

The aim of this experiment is to show that having GNNs that can maximize mutual information between the input graph and its representation is insufficient to guarantee their performance in the downstream tasks, because redundant information may still maximize mutual information but may degenerate the performance. To show this phenomenon, we perform two case studies: (1) a GNN is trained following the vanilla GCL (InfoMax) objective and (2) a GNN is trained following the vanilla GCL (InfoMax) objective while simultaneously a linear classifier that tasks the graph representations output by the GNN encoder is trained with random labels. These two GNNs have exactly the same architectures, hyperparametes and initialization. Specifically, the GNN architecture is GIN [204], with embedding dimension of 32, 5 layers with no skip connections and a dropout of 0.0.

Both GNN encoders are trained as above. In the first step of the evaluation, we want to test whether these GNNs keep mutual information maximization. For all graphs in the ogbg-molbace dataset, either one of the GNN provides a set of graph representations. For each GNN, we compare all its output graph representations. We find that, the output representations of every two graphs have difference that is greater than a digit accuracy. This implies that either one of the GNN keeps an one-to-one correspondance between the graphs in the dataset and their representations, which guarantees mutual information maximization.

We further compare these two GNNs encoders in the downstream task by using true labels. We impose two linear classifiers on the output representations of the above two GNN encoders to predict the true labels. The two linear classifiers have exactly the same architecture, hyperparametes and initialization. Specifically, a simple logistic classifier implemented using sklearn [205] is used with L2 regularization. The L2 strength is tuned using validation set. For the dataset ogbg-molbace, we follow the default train/val/test splits that are given by the original authors of OGB [203]. Note that, during the evaluation stage, the GNN encoders are fixed while the linear classifiers get trained. The evaluation performance is the curves as illustrated in Figure 5.2.

This observation inspires us to rethink what a good graph representation is. Recently, the information bottleneck has applied to learn graph representations [112], [113]. Specifically, the objective of graph information bottleneck (GIB) follows

$$\text{GIB:} \quad \max_{f} I(f(G); Y) - \beta I(G; f(G)), \tag{5.2}$$

where $(G, Y) \sim \mathbb{P}_{\mathcal{G}\times\mathcal{Y}}, \beta$ is a positive constant. Comparing Eq. 5.1 and Eq. 5.2, we may observe the different requirements between InfoMax and GIB: InfoMax asks for maximizing the information from the original graph, while GIB asks for minimizing such information but simultaneously maximizing the information that is relevant to the downstream tasks. As GIB asks to remove redundant information, GIB naturally avoids the issue encountered in Fig.5.2. Removing extra information also makes GNNs trained w.r.t. GIB robust to adverserial attack and strongly transferrable [112], [113].

Unfortunately, GIB requires the knowledge of the class labels Y from the downstream task and thus does not apply to self-supervised training of GNNs where there are few or no labels. Then, the question is how to learn robust and transferable GNNs in a self-supervised way.

To address this, we will develop a GCL approach that uses adversarial learning to avoid capturing redundant information during the representation learning.

In general, GCL methods use graph data augmentation (GDA) processes to perturb the original observed graphs and decrease the amount of information they encode. Then, the methods apply InfoMax over perturbed graph pairs (using different GDAs) to train an encoder f to capture the remaining information.

Definition 5.2.1 (Graph Data Augmentation (GDA))**.** *For a graph $G \in \mathcal{G}$, $T(G)$ denotes a graph data augmentation of G, which is a distribution defined over $\mathcal{G}$ conditioned on G. We use $t(G) \in \mathcal{G}$ to denote a sample of $T(G)$.*

Specifically, given two ways of GDA T_1 and T_2, the objective of GCL becomes

$$\text{GDA-GCL: } \max_f I(f(t_1(G)); f(t_2(G))), \text{ where } G \sim \mathbb{P}_{\mathcal{G}}, t_i(G) \sim T_i(G), i \in \{1, 2\}. \tag{5.3}$$

In practice, GDA processes are often pre-designed based on either domain knowledge or extensive evaluation, and improper choice of GDA may severely impact the downstream performance [83], [94]. We will review a few GDAs adopted in existing works in Sec.5.4.

In contrast to previous predefined GDAs, our idea, inspired by GIB, is to *learn* the GDA process (over a parameterized family), so that the encoder f can capture the **minimal information** that is sufficient to identify each graph.

5.2.1 Formulating AD-GCL

We optimize the following objective, over a GDA family $\mathcal{T}$ (defined below).

$$\text{AD-GCL: } \min_{T \in \mathcal{T}} \max_f I(f(G); f(t(G))), \quad \text{where } G \sim \mathbb{P}_{\mathcal{G}}, t(G) \sim T(G), \tag{5.4}$$

Definition 5.2.2 (Graph Data Augmentation Family)**.** *Let $\mathcal{T}$ denote a family of different GDAs $T_\Phi(\cdot)$, where Φ is the parameter in some universe. A $T_\Phi(\cdot) \in \mathcal{T}$ is a specific GDA with parameter Φ.*

The min-max principle in AD-GCL aims to train the encoder such that even with a very aggressive GDA (i.e., where $t(G)$ is very different from G), the mutual information / the correspondence between the perturbed graph and the original graph can be maximized. Compared with the two GDAs adopted in GDA-GCL (Eq.5.3), AD-GCL views the original graph G as the anchor while pushing its perturbation $T(G)$ as far from the anchor as it can. The automatic search over $T \in \mathcal{T}$ saves a great deal of effort evaluating different combinations of GDA as adopted in [94].

5.2.2 Relating AD-GCL to the Downstream Task

Next, we will theoretically characterize the property of the encoder trained via AD-GCL. The analysis here not only further illustrates the rationale of AD-GCL but helps design practical $\mathcal{T}$ when some knowledge of Y is accessible. But note that our analysis does not make any assumption on the availability of Y.

Note that GNNs learning graph representations is very different from CNNs learning image representations because GNNs are never injective mappings between the graph universe $\mathcal{G}$ and the representation space $\mathbb{R}^d$, because the expressive power of GNNs is limited by the 1-WL test [51]–[53]. So, we need to define a quotient space of $\mathcal{G}$ based on the equivalence given by the 1-WL test.

Definition 5.2.3 (Graph Quotient Space)**.** *Define the equivalence $\cong$ between two graphs $G_1 \cong G_2$ if G_1, G_2 cannot be distinguished by the 1-WL test. Define the quotient space $\mathcal{G}' = \mathcal{G}/\cong$.*

So every element in the quotient space, i.e., $G' \in \mathcal{G}'$, is a representative graph from a family of graphs that cannot be distinguished by the 1-WL test. Note that our definition also allows attributed graphs.

Definition 5.2.4 (Probability Measures in $\mathcal{G}'$)**.** *Define $\mathbb{P}_{\mathcal{G}'}$ over the space $\mathcal{G}'$ such that $\mathbb{P}_{\mathcal{G}'}(G') = \mathbb{P}_{\mathcal{G}}(G \cong G')$ for any $G' \in \mathcal{G}'$. Further define joint probability, $\mathbb{P}_{\mathcal{G}'\times\mathcal{Y}}(G', Y') = \mathbb{P}_{\mathcal{G}\times\mathcal{Y}}(G \cong G', Y = Y')$. Given a GDA $T(\cdot)$ defined over $\mathcal{G}$, define a distribution on $\mathcal{G}'$, as, $T'(G') = \mathbb{E}_{G\sim\mathbb{P}_{\mathcal{G}}}[T(G)|G \cong G']$ for $G' \in \mathcal{G}'$.*

Now, we provide our theoretical results with a reasonable assumption that GNNs with proper design may achieve the power of the 1-WL test [52] (see. Sec. 2.4).

Theorem 5.2.1. *Suppose the encoder f is implemented by a GNN as powerful as the 1-WL test. Suppose $\mathcal{G}$ is a countable space and thus $\mathcal{G}'$ is a countable space. Then, the optimal solution (f^*, T^*) to AD-GCL satisfies, letting $T'^*(G') = \mathbb{E}_{G\sim\mathbb{P}_{\mathcal{G}}}[T^*(G)|G \cong G']$,*

1. *$I(f^*(t^*(G)); G \,|\, Y) \leq \min_{T\in\mathcal{T}} I(t'(G'); G') - I(t'^*(G'); Y)$ where,*
   $t'(G') \sim T'(G')$, $t'^*(G') \sim T'^*(G')$, $(G, Y) \sim \mathbb{P}_{\mathcal{G}\times\mathcal{Y}}$ and $(G', Y) \sim \mathbb{P}_{\mathcal{G}'\times\mathcal{Y}}$.
2. *$I(f^*(G); Y) \geq I(f^*(t'^*(G')); Y) = I(t'^*(G'); Y)$ where,*
   $t'^*(G') \sim T'^*(G')$, $(G, Y) \sim \mathbb{P}_{\mathcal{G}\times\mathcal{Y}}$ and $(G', Y) \sim \mathbb{P}_{\mathcal{G}'\times\mathcal{Y}}$.

Proof. Because $\mathcal{G}$ and $\mathcal{G}'$ are countable, $P_{\mathcal{G}}$ and $P_{\mathcal{G}'}$ are defined over countable sets and thus discrete distribution. Later we may call a function $z(\cdot)$ can distinguish two graphs G_1, G_2 if $z(G_1) \neq z(G_2)$.

Moreover, for notational simplicity, we consider the following definition. Because f^* is as powerful as the 1-WL test. Then, for any two graphs $G_1, G_2 \in \mathcal{G}$, $G_1 \cong G_2$, $f^*(G_1) = f^*(G_2)$. We may define a mapping over $\mathcal{G}'$, also denoted by f^* which simply satisfies $f^*(G') :\triangleq f^*(G)$, where $G \cong G'$, and $G \in \mathcal{G}$ and $G' \in \mathcal{G}'$.

We first prove the statement 1, *i.e.,* the upper bound. We have the following inequality: Recall that $T'^*(G') = \mathbb{E}_{G\sim\mathbb{P}_{\mathcal{G}}}[T^*(G)|G \cong G']$ and $t'^*(G') \sim T'^*(G')$.

$$
\begin{aligned}
I(t'^*(G'); G') &= I(t'^*(G'); (G', Y)) - I(t'^*(G'); Y|G')] \\
&\overset{(a)}{=} I(t'^*(G'); (G', Y)) \\
&= I(t'^*(G'); Y) + I(t'^*(G'); G'|Y) \\
&\overset{(b)}{\geq} I(f^*(t'^*(G')); G'|Y) + I(t'^*(G'); Y)
\end{aligned} \tag{5.5}
$$

where (a) is because $t'^*(G') \perp_{G'} Y$, (b) is because the data processing inequality [202]. Moreover, because f^* could be as powerful as the 1-WL test and thus could be injective in $\mathcal{G}'$ a.e. with respect to the measure $\mathbb{P}_{\mathcal{G}'}$. Then, for any GDA $T(\cdot)$ and $T'(G') = \mathbb{E}_{G\sim\mathbb{P}_{\mathcal{G}}}[T(G)|G \cong G']$,

$$I(t'(G'); G') = I(f^*(t'(G')); f^*(G')) = I(f^*(t(G)); f^*(G)), \tag{5.6}$$

where $t'(G') \sim T'(G')$, $t(G) \sim T(G)$. Here, the second equality is because $f^*(G) = f^*(G')$ and $T'(G') = \mathbb{E}_{G\sim\mathbb{P}_{\mathcal{G}}}[T(G)|G \cong G']$.

Since $T^* = \arg\min_{T\in\mathcal{T}} I(f(t^*(G)); f(G))$ where $t^*(G) \sim T^*(G)$ and Eq.5.6, we have

$$I(t'^*(G'); G') = \arg\min_{T\in\mathcal{T}} I(t'(G'); G'), \text{ where } t'(G') \sim T'(G') = \mathbb{E}_{G\sim\mathbb{P}_{\mathcal{G}}}[T(G)|G \cong G']. \tag{5.7}$$

Again, because by definition $f^* = \arg\max_f I(f(G); f(t^*(G)))$ and f^* could be as powerful as the 1-WL test, its counterpart defined over $\mathcal{G}'$, i.e., $f^\star$, must be injective over $\mathcal{G}' \cap \text{Supp}(\mathbb{E}_{G'\sim\mathbb{P}_{\mathcal{G}'}}[T^*(G')])$ a.e. with respect to the measure $\mathbb{P}_{\mathcal{G}'}$ to achieve such mutual information maximization. Here, $\text{Supp}(\mu)$ defines the set where μ has non-zero measure. Because of the definition of $T'^*(G') = \mathbb{E}_{G\sim\mathbb{P}_{\mathcal{G}}}[T^*(G)|G \cong G']$,

$$\mathcal{G}' \cap \text{Supp}(\mathbb{E}_{G'\sim\mathbb{P}_{\mathcal{G}'}}[T^*(G')]) = \mathcal{G}' \cap \text{Supp}(\mathbb{E}_{G\sim\mathbb{P}_{\mathcal{G}}}[T^*(G)]).$$

Therefore, f^* is a.e. injective over $\mathcal{G}' \cap \text{Supp}(\mathbb{E}_{G\sim\mathbb{P}_{\mathcal{G}}}[T^*(G)])$ and thus

$$I(f^*(t'^*(G')); G'|Y) = I(f^*(t^*(G)); G'|Y), \tag{5.8}$$

Moreover, as f^* cannot cannot distinguish more graphs in $\mathcal{G}$ than $\mathcal{G}'$ as the power of f^* is limited by 1-WL test, thus,

$$I(f^*(t^*(G)); G'|Y) = I(f^*(t^*(G)); G|Y). \tag{5.9}$$

Plugging Eqs.5.7, 5.8, 5.9 into Eq.5.5, we achieve

$$I(f^*(t^*(G)); G|Y) \leq \arg\min_{T \in \mathcal{T}} I(t'(G'); G') - I(t'^*(G'); Y)$$

where $t'(G') \sim T'(G') = \mathbb{E}_{G \sim \mathbb{P}_\mathcal{G}}[T(G)|G \cong G']$ and $t'^*(G') \sim T'^*(G') = \mathbb{E}_{G \sim \mathbb{P}_\mathcal{G}}[T^*(G)|G \cong G']$, which gives us the statement 1, which is the upper bound.

We next prove the statement 2, *i.e.,* the lower bound. Recall (T^*, f^*) is the optimal solution to Eq.5.4 and $t^*(\cdot)$ denotes samples from $T^*(\cdot)$.

Again, because $f^* = \arg\max_f I(f(G); f(t^*(G)))$, f^* must be injective on

$$\mathcal{G}' \cap \text{Supp}(\mathbb{E}_{G' \sim \mathbb{P}_{\mathcal{G}'}}[T^*(G')]) \tag{5.10}$$

a.e. with respect to the measure $\mathbb{P}_{\mathcal{G}'}$. Given $t'^*(G')$, $t'^*(G') \to f^*(t'^*(G'))$ is an injective deterministic mapping. Therefore, for any random variable Q,

$$I(f^*(t'^*(G')); Q) = I(t'^*(G'); Q), \quad \text{where } G' \sim \mathbb{P}_{\mathcal{G}'}, t'^*(G') \sim T'^*(G').$$

Of course, we may set $Q = Y$. So,

$$I(f^*(t'^*(G')); Y) = I(t'^*(G'); Y), \quad \text{where } (G', Y) \sim \mathbb{P}_{\mathcal{G}' \times \mathcal{Y}}, t'^*(G') \sim T'^*(G'). \tag{5.11}$$

Because of the data processing inequality [202] and $T'^*(G') = \mathbb{E}_{G \sim \mathbb{P}_\mathcal{G}}[T^*(G)|G \cong G']$, we further have

$$I(f^*(t^*(G)); Y) \geq I(f^*(t'^*(G')); Y), \tag{5.12}$$

where $(G', Y) \sim \mathbb{P}_{\mathcal{G}' \times \mathcal{Y}}, (G, Y) \sim \mathbb{P}_{\mathcal{G} \times \mathcal{Y}}, t'^*(G') \sim T'^*(G'), t^*(G) \sim T^*(G)$.

Further because of the data processing inequality [202],

$$I(f^*(G); Y) \geq I(f^*(t^*(G)); Y). \tag{5.13}$$

Combining Eqs.5.11, 5.12, 5.13, we have

$$I(f^*(G);Y) \geq I(f^*(t^*(G));Y) \geq I(f^*(t'^*(G'));Y) = I(t'^*(G');Y),$$

which concludes the proof of the lower bound. □

5.2.3 Implications of AD-GCL Priciple

The statement 1 in Theorem 5.2.1 guarantees a upper bound of the information that is captured by the representations but irrelevant to the downstream task, which matches our aim. This bound has a form very relevant to the GIB principle (Eq.5.2 when $\beta = 1$), since $\min_{T\in\mathcal{T}} I(t'(G');G') - I(t'^*(G');Y) \geq \min_f [I(f(G);G) - I(f(G);Y)]$,

where f is a GNN encoder as powerful as the 1-WL test. But note that this inequality also implies that the encoder given by AD-GCL may be worse than the optimal encoder given by GIB ($\beta = 1$). This makes sense as GIB has the access to the downstream task Y.

The statement 2 in Theorem 5.2.1 guarantees a lower bound of the mutual information between the learnt representations and the labels of the downstream task. As long as the GDA family $\mathcal{T}$ has a good control, $I(t'^*(G');Y) \geq \min_{T\in\mathcal{T}} I(t'(G');Y)$ and $I(f^*(G);Y)$ thus cannot be too small. This implies that it is better to regularize when learning over $\mathcal{T}$. In our instantiation, based on edge-dropping augmentation (Sec. 5.3), we regularize the ratio of dropped edges per graph.

5.3 Instantiation of AD-GCL via Learnable Edge Perturbation

We now introduce a practical instantiation of the AD-GCL principle (Eq. 5.4) based on learnable edge-dropping augmentations as illustrated in Fig. 5.1. The objective of AD-GCL has two folds: (1) Optimize the encoder f to maximize the mutual information between the representations of the original graph G and its augmented graph $t(G)$; (2) Optimize the GDA $T(G)$ where $t(G)$ is sampled to minimize such a mutual information. We always set the encoder as a GNN f_Θ with learnable parameters Θ and next we focus on the GDA, $T_\Phi(G)$ that has learnable parameters Φ.

5.3.1 Learnable Edge Dropping GDA Model

Edge dropping is the operation of deleting some edges in a graph. As a proof of concept, we adopt edge dropping to formulate the GDA family $\mathcal{T}$. Other types of GDAs such as node dropping, edge adding and feature masking can also be paired with our AD-GCL principle. Interestingly, in our experiments, edge-dropping augmentation optimized by AD-GCL has already achieved much better performance than any pre-defined random GDAs even carefully selected via extensive evaluation [94] (See Sec.5.5). We attribute such improvement to the AD-GCL principle. Another reason that supports edge dropping is due to our Theorem 5.2.1 statement 2, which shows that good GDAs should keep some information related to the downstream tasks. Many GRL downstream tasks such as molecule classification only depends on the structural fingerprints that can be represented as subgraphs of the original graph [48]. Dropping a few edges may not change those subgraph structures and thus keeps the information sufficient to the downstream classification. But note that this reasoning does not mean that we leverage domain knowledge to design GDA, as the family $\mathcal{T}$ is still broad and the specific GDA still needs to be optimized. Moreover, experiments show that our instantiation also works extremely well on social network classification and molecule property regression, where the evidence of subgraph fingerprints may not exist any more.

5.3.2 Parameterizing Edge Dropping Weights

For each $G = (V, E)$, we set $T_\Phi(G)$, $T \in \mathcal{T}$ as a random graph model [206], [207] conditioning on G. Each sample $t(G) \sim T_\Phi(G)$ is a graph that shares the same node set with G while the edge set of $t(G)$ is only a subset of E. Each edge $e \in E$ will be associated with a random variable $p_e \sim \text{Bernoulli}(\omega_e)$, where e is in $t(G)$ if $p_e = 1$ and is dropped otherwise.

We parameterize the Bernoulli weights ω_e by leveraging another GNN, *i.e.,* the *augmenter*, to run on G according to Eq.2.1 of K layers, get the final-layer node representations $\{h_v^{(K)}|v \in V\}$ and set

$$\omega_e = \text{MLP}([h_u^{(K)}; h_z^{(K)}]), \quad \text{where } e = (u, z) \text{ and } \{h_v^{(K)} \mid v \in V\} = \text{GNN-augmenter}(G) \tag{5.14}$$

To train $T(G)$ in an end-to-end fashion, we relax the discrete p_e to be a continuous variable in $[0, 1]$ and utilize the Gumbel-Max reparametrization trick [208], [209]. Specifically, $p_e = \text{Sigmoid}((\log \delta - \log(1 - \delta) + \omega_e)/\tau)$, where $\delta \sim \text{Uniform}(0,1)$. As temperature hyper-parameter $\tau \to 0$, p_e gets closer to being binary. Moreover, the gradients $\frac{\partial p_e}{\partial \omega_e}$ are smooth and well defined. This style of edge dropping based on a random graph model has also been used for parameterized explanations of GNNs [210].

5.3.3 Regularization for AD-GCL

As shown in Theorem 5.2.1, a reasonable GDA should keep a certain amount of information related to the downstream tasks (statement 2). Hence, we expect the GDAs in the edge dropping family $\mathcal{T}$ not to perform very aggressive perturbation. Therefore, we regularize the ratio of edges being dropped per graph by enforcing the following constraint: For a graph G and its augmented graph $t(G)$, we add $\sum_{e \in E} \omega_e/|E|$ to the objective, where ω_e is defined in Eq.5.14 indicates the probability that e gets dropped.

Putting everything together, the final objective is as follows.

$$\min_{\Phi} \max_{\Theta} I(f_\Theta(G); f_\Theta(t(G))) + \lambda_{\text{reg}} \mathbb{E}_G \Big[\sum_{e \in E} \omega_e/|E| \Big], \text{ where } G \sim \mathbb{P}_{\mathcal{G}}, t(G) \sim T_\Phi(G). \tag{5.15}$$

Note Φ corresponds to the learnable parameters of the augmenter GNN and MLP used to derive the ω_e's and Θ corresponds to the learnable parameters of the GNN f.

5.3.4 Estimating the AD-GCL Objective

In our implementation, the second (regularization) term is easy to estimate empirically. For the first (mutual information) term, we adopt InfoNCE as the estimator [211]–[213], which is known to be a lower bound of the mutual information and is frequently used for contrastive learning [105], [211], [214]. Specfically, during the training, given a minibatch of m graphs $\{G_i\}_{i=1}^m$, let $z_{i,1} = g(f_\Theta(G_i))$ and $z_{i,2} = g(f_\Theta(t(G_i)))$ where $g(\cdot)$ is the projection head implemented by a 2-layer MLP as suggested in [214]. With $sim(\cdot,\cdot)$ denoting cosine similarity, we estimate the mutual information for the mini-batch as follows.

$$I(f_\Theta(G); f_\Theta(t(G))) \to \hat{I} = \frac{1}{m}\sum_{i=1}^{m} \log \frac{\exp(sim(z_{i,1}, z_{i,2}))}{\sum_{i'=1, i'\neq i}^{m} \exp(sim(z_{i,1}, z_{i',2}))} \tag{5.16}$$

Algorithm 4 describes the self-supervised training algorithm for AD-GCL with learnable edge-dropping GDA. Note that augmenter $T_\Phi(\cdot)$ with parameters Φ is implemented as a GNN followed by an MLP to obtain the Bernoulli weights ω_e.

5.4 Extended Related Work

Here, we focus on the topics that are most relevant to graph contrastive learning (GCL).

Contrastive learning (CL) [104], [211], [212], [215]–[217] was initially proposed to train CNNs for image representation learning and has recently achieved great success [214], [218]. GCL applies the idea of CL on GNNs. In contrast to the case of CNNs, GCL trained using GNNs posts us new fundamental challenges. An image often has multiple natural views, say by imposing different color filters and so on. Hence, different views of an image give natural contrastive pairs for CL to train CNNs. However, graphs are more abstract and the irregularity of graph structures typically provides crucial information. Thus, designing contrastive pairs for GCL must play with irregular graph structures and thus becomes more challenging. Some works use different parts of a graph to build contrastive pairs, including nodes *v.s.* whole graphs [84], [93], nodes *v.s.* nodes [92], nodes *v.s.* subgraphs [83], [219]. Other works adopt graph data augmentations (GDA) such as edge perturbation [90] to generate contrastive pairs. Recently. GraphCL [94] gives an extensive study on different combinations of GDAs including node dropping, edge perturbation, subgraph sampling and feature mask-

Algorithm 4: Training Learnable Edge-Dropping GDA under AD-GCL principle.

Input: Data $\{G_m \sim \mathcal{G} \mid m = 1, 2 \dots M\}$;
Encoder $f_\Theta(\cdot)$; Augmenter $T_\Phi(\cdot)$; Projection Head $g_\Psi(\cdot)$; Cosine Similarity $sim(\cdot)$

Hyper-Params : Edge-Dropping Regularization Strength λ_{reg}; learning rates α, β

Output: Trained Encoder $f_\Theta(\cdot)$

begin
  for *number of training epochs* **do**
    for *sampled minibatch* $\{G_n = (V_n, E_n) : n = 1, 2 \dots N\}$ **do**
      for n = *1 to* N **do**
        $h_{1,n} = f_\Theta(G_n)$
        $z_{1,n} = g_\Psi(h_{1,n})$
        $t(G_n) \sim T_\Phi(G_n)$
        set $p_e, \forall e \in E_n$ **from** $t(G_n)$
        $\mathcal{R}_n = \sum_{e \in E_n} p_e / |E_n|$
        $h_{2,n} = f_\Theta(t(G_n))$;
        $z_{1,n} = g_\Psi(h_{2,n})$
      end
      define $\mathcal{L}_n = -\log \frac{\exp(sim(z_{1,n}, z_{2,n}))}{\sum_{n\prime=1, n\prime \neq n}^{N} \exp\left(sim(z_{1,n}, z_{2,n\prime})\right)}$
      `/* calculate NCE loss for minibatch */`
      $\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n$
      `/* calculate regularization term for minibatch */`
      $\mathcal{R} = \frac{1}{N} \sum_{n=1}^{N} \mathcal{R}_n$
      `/* update augmenter params via gradient ascent */`
      $\Phi \leftarrow \Phi + \alpha \nabla_\Phi (\mathcal{L} - \lambda_{\text{reg}} * \mathcal{R})$
      `/* update enocder & projection head params via gradient descent */`
      $\Theta \leftarrow \Theta - \beta \nabla_\Theta(\mathcal{L}); \quad \Psi \leftarrow \Psi - \beta \nabla_\Psi(\mathcal{L})$
    end
  end
  return *Encoder* $f_\Theta(\cdot)$
end

ing. Extensive evaluation is required to determine good combinations. MVGRL [95] and GCA [89] leverage the domain knowledge of network science and adopt network centrality to perform GDAs. Note that none of the above methods consider optimizing augmentations. In contrast, our principle AD-GCL provides theoretical guiding principles to optimize augmentations. Very recently, JOAO [220] adopts a bi-level optimization framework sharing some high-level ideas with our adversarial training strategy but has several differences: 1) the GDA search space in JOAO is set as different types of augmentation with uniform perturbation, such as uniform edge/node dropping while we allow augmentation with non-uniform perturbation. 2) JOAO relaxes the GDA combinatorial search problem into continuous space via Jensen's inequality and adopts projected gradient descent to optimize. Ours, instead, adopts Bayesian modeling plus reparameterization tricks to optimize. The performance comparison between AD-GCL and JOAO for the tasks investigated in Sec. 5.5 is given in Appendix 5.5.5.

Tian et al. [221] has recently proposed the InfoMin principle that shares some ideas with AD-GCL but there are several fundamental differences. Theoretically, InfoMin needs the downstream tasks to supervise the augmentation. Rephrased in our notation, the optimal augmentation $T_{IM}(G)$ given by InfoMin (called the sweet spot in [221]) needs to satisfy $I(t_{IM}(G);Y) = I(G;Y)$ and $I(t_{IM}(G);G|Y) = 0$, $t_{IM}(G) \sim T_{IM}(G)$, neither of which are possible without the downstream-task knowledge. Instead, our Theorem 5.2.1 provides more reasonable arguments and creatively suggests using regularization to control the tradeoff. Empirically, InfoMin is applied to CNNs while AD-GCL is applied to GNNs. AD-GCL needs to handle the above challenges due to irregular graph structures and the limited expressive power of GNNs [52], [53], which InfoMin does not consider.

5.5 Experimental Evaluation

This section is devoted to the empirical evaluation of the proposed instantiation of our AD-GCL principle. Our initial focus is on unsupervised learning which is followed by analysis of the effects of regularization. We further apply AD-GCL to transfer and semi-supervised learning settings.

5.5.1 Datasets

A wide variety of datasets from different domains for a range of graph property prediction tasks are used for our experiments. Here, we summarize and point out the specific experiment setting for which they are used.

- Table 5.1 shows the datasets for chemical molecular property prediction which are from Open Graph Benchmark (OGB) [203] and ZINC-10K [222]. These are used in the unsupervised learning setting for both classification and regression tasks. We are the first one to considering using regression tasks and the corresponding datasets in the evaluation of self-supervised GNN.

- Table 5.2 shows the datasets which contain biochemical and social networks. These are taken from the TU Benchmark Datasets [223]. We use them for graph classification tasks in both unsupervised and semi-supervised learning settings.

- Table 5.3 shows the datasets consisting of biological interactions and chemical molecules from [83]. These datasets are used for graph classification in the transfer learning setting.

Table 5.1. Summary of chemical molecular properties datasets used for unsupervised learning experiments. Datasets obtained from OGB [203] and [222]

Name	#Graphs	Avg #Nodes	Avg #Edges	#Tasks	Task Type	Metric
ogbg-molesol	1,128	13.3	13.7	1	Regression	RMSE
ogbg-mollipo	4,200	27.0	29.5	1	Regression	RMSE
ogbg-molfreesolv	642	8.7	8.4	1	Regression	RMSE
ogbg-molbace	1,513	34.1	36.9	1	Binary Class.	ROC-AUC
ogbg-molbbbp	2,039	24.1	26.0	1	Binary Class.	ROC-AUC
ogbg-molclintox	1,477	26.2	27.9	2	Binary Class.	ROC-AUC
ogbg-moltox21	7,831	18.6	19.3	12	Binary Class.	ROC-AUC
ogbg-molsider	1,427	33.6	35.4	27	Binary Class.	ROC-AUC
ZINC-10K	12,000	23.16	49.83	1	Regression	MAE

Table 5.2. Summary of biochemical and social networks from TU Benchmark Dataset [223] used for unsupervised and semi-supervised learning experiments. The evaluation metric for all these datasets is Accuracy.

Dataset	#Graphs	Avg. #Nodes	Avg. #Edges	#Classes
Biochemical Molecules				
NCI1	4,110	29.87	32.30	2
PROTEINS	1,113	39.06	72.82	2
MUTAG	188	17.93	19.79	2
DD	1,178	284.32	715.66	2
Social Networks				
COLLAB	5,000	74.5	2457.78	3
REDDIT-BINARY	2,000	429.6	497.75	2
REDDIT-MULTI-5K	4,999	508.8	594.87	5
IMDB-BINARY	1,000	19.8	96.53	2
IMDB-MULTI	1,500	13.0	65.94	3

5.5.2 Unsupervised Learning Settting

Evaluation Protocol. In this setting, an encoder (specifically GIN [204]) is trained with different self-supervised methods to learn graph representations, which are then evaluated by feeding these representations to make prediction for the downstream tasks. We use datasets from Open Graph Benchmark (OGB) [203], TU Dataset [223] and ZINC [222] for graph-level property classification and regression. All methods (ours and baselines) are first trained with the corresponding self-supervised objective and then evaluated with a linear classifier/regressor. We follow [214] and adopt a linear evaluation protocol. Specifically, once the encoder provides representations, a Ridge regressor (+ L2) and Logistic (+ L2) classifier is trained on top and evaluated for regression and classification tasks respectively. Both methods are implemented using sklearn [205] and uses LBFGS [224] or LibLinear [225] solvers. Finally, the lone hyper-parameter of the downstream linear model i.e. L2 regularization strength is grid searched among $\{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$ on the validation set for every single representation evaluation.

Table 5.3. Summary of biological interaction and chemical molecule datasets from [83]. Used for graph classification in transfer learning experiments. The evaluation metric is ROC-AUC.

Dataset	Utilization	#Graphs	Avg. #Nodes	Avg. Degree
Protein-Protein Interaction Networks				
PPI-306K	Pre-Training	306,925	39.82	729.62
PPI	Finetuning	88,000	49.35	890.77
Chemical Molecules				
ZINC-2M	Pre-Training	2,000,000	26.62	57.72
BBBP	Finetuning	2,039	24.06	51.90
Tox21	Finetuning	7,831	18.57	38.58
SIDER	Finetuning	1,427	33.64	70.71
ClinTox	Finetuning	1,477	26.15	55.76
BACE	Finetuning	1,513	34.08	73.71
HIV	Finetuning	41,127	25.51	54.93
MUV	Finetuning	93,087	24.23	52.55
ToxCast	Finetuning	8,576	18.78	38.52

For the Open Graph Benchmark Datasets (ogbg-mol*), we directly use the processed data in Pytorch Geometric format which is available online [1]. The processed data includes train/val/test that follow a scaffolding split. More details are present in the OGB paper [203]. Additionally, we make use of the evaluators written by authors for standardizing the evaluation. The evaluation metric varies depending on the task at hand. For regression tasks it is RMSE (root mean square error) and for classification it is ROC-AUC (%).

For the ZINC-10K dataset [222], we use the processed data in Pytorch Geometric format that is made available online[2] by the authors. We use the same train/val/test splits that are provided. We follow the authors and adopt MAE (mean absolute error) as the test metric.

For the TU Datasets [223], we obtain the data from Pytorch Geometric Library [3] and follow the conventional 10-Fold evaluation. Following standard protocol, we adopt Accuracy (%) as the test metric. All our experiments are performed 10 times with different random

[1]↑https://ogb.stanford.edu/docs/graphprop/
[2]↑https://github.com/graphdeeplearning/benchmarking-gnns/tree/master/data
[3]↑https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html

seeds and we report mean and standard deviation of the corresponding test metric for each dataset.

Hyper-parameters. The encoder used for ours and baselines is GIN [204]. The encoder is fixed and not tuned while performing self-supervised learning (i.e. embedding dimension, number of layers, pooling type) for all the methods to keep the comparison fair. The reasoning is that any performance difference we witness should only be attributed to the self-supervised objective and not to the encoder design. Details of encoder for specific datasets.

- OBG - *emb dim = 300, num gnn layers = 5, pooling = add, skip connections = None, dropout = 0.5, batch size = 32*
- ZINC-10K - *emb dim = 100, num gnn layers = 5, pooling = add, skip connections = None, dropout = 0.5, batch size = 64*
- TU Datasets - *emb dim = 32, num gnn layers = 5, pooling = add, skip connections = None, dropout = 0.5, batch size = 32*

The optimization of AD-GCL is performed using Adam and the learning rates for the encoder and the augmenter in AD-GCL are tuned among $\{0.01, 0.005, 0.001\}$. We find that asymmetric learning rates for encoder and augmenter tend to make the training non-stable. Thus, for stability we adopt a learning rate of 0.001 for all the datasets and experiments. The number of training epochs are chosen among $\{20, 50, 80, 100, 150\}$ using the validation set.

Results. We consider two types of AD-GCL, where one is with a fixed regularization weight $\lambda_{\text{reg}} = 5$ (Eq.5.15), termed AD-GCL-FIX, and another is with λ_{reg} tuned over the validation set among $\{0.1, 0.3, 0.5, 1.0, 2.0, 5.0, 10.0\}$, termed AD-GCL-OPT. AD-GCL-FIX assumes any information from the downstream task as unavailable while AD-GCL-OPT assumes the augmentation search space has some weak information from the downstream task. A full range of analysis on how λ_{reg} impacts AD-GCL will be investigated in Sec. 5.6. We compare AD-GCL with three unsupervised/self-supervised learning baselines for graph-level tasks, which include randomly initialized untrained GIN (RU-GIN) [204], InfoGraph [84]

and GraphCL [94]. Previous works [84], [94] show that they generally outperform graph kernels [26], [226], [227] and network embedding methods [99], [100], [228], [229].

We also adopt GCL with GDA based on non-adversarial edge dropping (NAD-GCL) for ablation study. NAD-GCL drops the edges of a graph uniformly at random. We consider NAD-GCL-FIX and NAD-GCL-OPT with different edge drop ratios. NAD-GCL-GCL adopts the edge drop ratio of AD-GCL-FIX at the saddle point of the optimization (Eq.5.15) while NAD-GCL-OPT optimally tunes the edge drop ratio over the validation datasets to match AD-GCL-OPT. We also adopt fully supervised GIN (F-GIN) to provide an anchor of the performance. We stress that all methods adopt GIN [204] as the encoder. Except F-GIN, all methods adopt a downstream *linear* classifier or regressor with the same hyper-parameters for fair comparison. Adopting *linear models* was suggested by [105], which explicitly attributes any performance gain/drop to the quality of learnt representations.

Tables 5.4 show the results for unsupervised graph level property prediction in social and chemical domains respectively. We witness the big performance gain of AD-GCL as opposed to all baselines across all the datasets. Note GraphCL utilizes extensive evaluation to select the best combination of augmentions over a broad GDA family including node-dropping, edge dropping and subgraph sampling. Our results indicate that such extensive evaluation may not be necessary while optimizing the augmentation strategy in an adversarial way is greatly beneficial.

We stress that edge dropping is not cherry picked as the search space of augmentation strategies. Other search spaces may even achieve better performance, while an extensive investigation is left for the future work.

Moreover, AD-GCL also clearly improves upon the performance against its non-adversarial counterparts (NAD-GCL) across all the datasets, which further demonstrates stable and significant advantages of the AD-GCL principle. Essentially, the input-graph-dependent augmentation learnt by AD-GCL yields much benefit.

Finally, we compare AD-GCL-FIX with AD-GCL-OPT. Interestingly, two methods achieve comparable results though AD-GCL-OPT is sometimes better. This observation implies that the AD-GCL principle may be robust to the choice of λ_{reg} and thus motivates the analysis in the next subsection. Moreover, weak information from the downstream tasks indeed

Table 5.4. Unsupervised learning performance for (TOP) biochemical and social network classification in TU datasets [223] (Averaged accuracy ± std. over 10 runs) and (BOTTOM) chemical molecules property prediction in OGB datasets [203] (mean ± std. over 10 runs). **Bold**/**Bold*** indicats our methods outperform baselines with $\geq 0.5/\geq 2$ std respectively. Fully supervised (F-GIN) results are shown **only** for placing GRL methods in perspective. Ablation-study (AB-S) results do not count as baselines.

	Dataset	NCI1	PROTEINS	MUTAG	DD	COLLAB	RDT-B	RDT-M5K	IMDB-B	IMDB-M
	F-GIN	78.27 ± 1.35	72.39 ± 2.76	90.41 ± 4.61	74.87 ± 3.56	74.82 ± 0.92	86.79 ± 2.04	53.28 ± 3.17	71.83 ± 1.93	48.46 ± 2.31
Baselines	RU-GIN [204]	62.98 ± 0.10	69.03 ± 0.33	87.61 ± 0.39	74.22 ± 0.30	63.08 ± 0.10	58.97 ± 0.13	27.52 ± 0.61	51.86 ± 0.33	32.81 ± 0.57
Baselines	InfoGraph [84]	68.13 ± 0.59	72.57 ± 0.65	87.71 ± 1.77	75.23 ± 0.39	70.35 ± 0.64	78.79 ± 2.14	51.11 ± 0.55	71.11 ± 0.88	48.66 ± 0.67
Baselines	GraphCL [94]	68.54 ± 0.55	72.86 ± 1.01	88.29 ± 1.31	74.70 ± 0.70	71.26 ± 0.55	82.63 ± 0.99	53.05 ± 0.40	70.80 ± 0.77	48.49 ± 0.63
AB-S	NAD-GCL-FIX	69.23 ± 0.60	72.81 ± 0.71	88.58 ± 1.58	74.55 ± 0.55	71.56 ± 0.58	83.41 ± 0.66	52.72 ± 0.71	70.94 ± 0.77	48.33 ± 0.47
AB-S	NAD-GCL-OPT	69.30 ± 0.32	73.18 ± 0.71	89.05 ± 1.06	74.55 ± 0.55	72.04 ± 0.67	83.74 ± 0.76	53.43 ± 0.26	71.94 ± 0.59	49.01 ± 0.93
Ours	AD-GCL-FIX	**69.67 ± 0.51***	**73.59 ± 0.65**	**89.25 ± 1.45**	74.49 ± 0.52	**73.32 ± 0.61***	**85.52 ± 0.79***	53.00 ± 0.82	**71.57 ± 1.01**	**49.04 ± 0.53**
Ours	AD-GCL-OPT	**69.67 ± 0.51***	**73.81 ± 0.46***	**89.70 ± 1.03**	75.10 ± 0.39	**73.32 ± 0.61***	**85.52 ± 0.79***	**54.93 ± 0.43***	**72.33 ± 0.56***	**49.89 ± 0.66***

	Task	Regression (Downstream Classifier - Linear Regression + L2)				Classification (Downstream Classifier - Logistic Regression + L2)				
	Dataset	molesol	mollipo	molfreesolv	ZINC-10K	molbace	molbbbp	molclintox	moltox21	molsider
	Metric	RMSE (shared) (↓)			MAE (↓)	ROC-AUC % (shared) (↑)				
	F-GIN	1.173 ± 0.057	0.757 ± 0.018	2.755 ± 0.349	0.254 ± 0.005	72.97 ± 4.00	68.17 ± 1.48	88.14 ± 2.51	74.91 ± 0.51	57.60 ± 1.40
Baselines	RU-GIN [204]	1.706 ± 0.180	1.075 ± 0.022	7.526 ± 2.119	0.809 ± 0.022	75.07 ± 2.23	64.48 ± 2.46	72.29 ± 4.15	71.53 ± 0.74	62.29 ± 1.12
Baselines	InfoGraph [84]	1.344 ± 0.178	1.005 ± 0.023	10.005 ± 4.819	0.890 ± 0.017	74.74 ± 3.64	66.33 ± 2.79	64.50 ± 5.32	69.74 ± 0.57	60.54 ± 0.90
Baselines	GraphCL [94]	1.272 ± 0.089	0.910 ± 0.016	7.679 ± 2.748	0.627 ± 0.013	74.32 ± 2.70	68.22 ± 1.89	74.92 ± 4.42	72.40 ± 1.01	61.76 ± 1.11
AB-S	NAD-GCL-FIX	1.392 ± 0.065	0.952 ± 0.024	5.840 ± 0.877	0.609 ± 0.010	73.60 ± 2.73	66.12 ± 1.80	73.32 ± 3.66	71.65 ± 0.94	60.41 ± 1.48
AB-S	NAD-GCL-OPT	1.242 ± 0.096	0.897 ± 0.022	5.840 ± 0.877	0.609 ± 0.010	73.69 ± 3.67	67.70 ± 1.78	74.40 ± 4.92	71.65 ± 0.94	61.14 ± 1.43
Ours	AD-GCL-FIX	**1.217 ± 0.087**	**0.842 ± 0.028***	**5.150 ± 0.624***	**0.578 ± 0.012***	**76.37 ± 2.03**	68.24 ± 1.47	**80.77 ± 3.92**	71.42 ± 0.73	**63.19 ± 0.95**
Ours	AD-GCL-OPT	**1.136 ± 0.050***	**0.812 ± 0.020***	**4.145 ± 0.369***	**0.544 ± 0.004***	**77.27 ± 2.56**	**69.54 ± 1.92**	**80.77 ± 3.92**	**72.92 ± 0.86**	**63.19 ± 0.95**

help with controlling the search space and further betters the performance. We also list the optimal λ_{reg}'s of AD-GCL-OPT for different datasets in Section 5.6.2 for the purpose of comparison and reproduction.

On the Choice of Downstream Classifier. We find that the choice of the downstream classifier can significantly affect the evaluation of the self-supervised representations. InfoGraph [84] and GraphCL [94] adopt a non-linear SVM model as the downstream classifier. Such a non-linear model is more powerful than the linear model we adopt and thus causes some performance gap between the results showed in Table 5.4 (TOP) and (BOTTOM) and their original results (listed in Table 5.5). We argue that using a non-linear SVM model as the downstream classifier is unfair, because the performance of even a randomly initialized untrained GIN (RU-GIN) is significantly improved (comparing results from Table 5.4 (TOP)

to Table 5.5). Therefore, we argue for adopting a linear classifier protocol as suggested by [105]. That having been said, our methods (both AD-GCL-FIX and AD-GCL-OPT) still performs significantly better than baselines in most cases, even when a non-linear SVM classifer is adopted, as shown in Table 5.5. Several relative gains are there no matter whether the downstream classifier is a simple linear model (Tables 5.4) or a non-linear SVM model (Table 5.5). AD-GCL methods significantly outperform InfoGraph in 5 over 8 datasets and GraphCL in 6 over 8 datasets. This further provides the evidence for the effectiveness of our method.

Table 5.5. Unsupervised Learning results on TU Datasets using a non-linear SVM classifier as done in GraphCL [94]. Averaged Accuracy (%) ± std. over 10 runs. This is different from the linear classifier used to show results in Tables 5.4 (TOP) and (BOTTOM).

	NCI1	PROTEINS	DD	MUTAG	COLLAB	RDT-B	RDT-M5K	IMDB-B
RU-GIN	65.40 ± 0.17	72.73 ± 0.51	75.67 ± 0.29	87.39 ± 1.09	65.29 ± 0.16	76.86 ± 0.25	48.48 ± 0.28	69.37 ± 0.37
InfoGraph	76.20 ± 1.06	74.44 ± 0.31	72.85 ± 1.78	89.01 ± 1.13	70.65 ± 1.13	82.50 ± 1.42	53.46 ± 1.03	73.03 ± 0.87
GraphCL	77.87 ± 0.41	74.39 ± 0.45	78.62 ± 0.40	86.80 ± 1.34	71.36 ± 1.15	89.53 ± 0.84	55.99 ± 0.28	71.14 ± 0.44
AD-GCL-FIX	75.77 ± 0.50	**75.04 ± 0.48**	75.38 ± 0.41	88.62 ± 1.27	**74.79 ± 0.41***	**92.06 ± 0.42***	**56.24 ± 0.39**	71.49 ± 0.98
AD-GCL-OPT	75.86 ± 0.62	**75.04 ± 0.48**	75.73 ± 0.51	88.62 ± 1.27	**74.89 ± 0.90***	**92.35 ± 0.42***	**56.24 ± 0.39**	71.49 ± 0.98

In our evaluation, we also observe several further benefits of using a downstream linear model in practice, would like to list them here. First, linear classifiers are much faster to train/converge in practice, especially for the large-scaled datasets or large embedding dimensions, which is good for practical usage. We observe that non-linear SVM classifiers induce a rather slow convergence, when applying to those several OGB datasets where the embedding dimensions are 300 (Table 5.4 (BOTTOM)). Second, compared to the linear model, the non-liner SVM may introduce additional hyper-parameters which not only need further effort to be tuned but also weaken the effect of the self-training procedure on the downstream performance.

5.5.3 Transfer Learning Setting

Evaluation Protocol. We follow the same evaluation protocol as done in [83]. In this setting, self-supervised methods are trained on the pre-train dataset and later used to be

Table 5.6. Transfer learning performance for chemical molecules property prediction (mean ROC-AUC ± std. over 10 runs). **Bold** indicates our methods outperform baselines with ≥ 0.5 std..

Pre-Train Dataset	ZINC 2M								PPI-306K
Fine-Tune Dataset	BBBP	Tox21	SIDER	ClinTox	BACE	HIV	MUV	ToxCast	PPI
No Pre-Train	65.8 ± 4.5	74.0 ± 0.8	57.3 ± 1.6	58.0 ± 4.4	70.1 ± 5.4	75.3 ± 1.9	71.8 ± 2.5	63.4 ± 0.6	64.8 ± 1.0
EdgePred [83]	67.3 ± 2.4	76.0 ± 0.6	60.4 ± 0.7	64.1 ± 3.7	79.9 ± 0.9	76.3 ± 1.0	74.1 ± 2.1	64.1± 0.6	65.7 ± 1.3
AttrMasking [83]	64.3 ± 2.8	76.7 ± 0.4	61.0 ± 0.7	71.8 ± 4.1	79.3 ± 1.6	77.2 ± 1.1	74.7 ± 1.4	64.2 ± 0.5	65.2 ± 1.6
ContextPred [83]	68.0 ± 2.0	75.7 ± 0.7	60.9 ± 0.6	65.9 ± 3.8	79.6 ± 1.2	77.3 ± 1.0	75.8 ± 1.7	63.9 ± 0.6	64.4 ± 1.3
InfoGraph [84]	68.8 ± 0.8	75.3 ± 0.5	58.4 ± 0.8	69.9 ±3.0	75.9 ± 1.6	76.0 ± 0.7	75.3 ± 2.5	62.7 ± 0.4	64.1 ± 1.5
GraphCL [94]	69.68 ± 0.67	73.87 ± 0.66	60.53 ± 0.88	75.99 ± 2.65	75.38 ± 1.44	78.47 ± 1.22	69.8 ± 2.66	62.40 ± 0.57	67.88 ± 0.85
AD-GCL-FIX	70.01 ±1.07	76.54 ± 0.82	**63.28 ± 0.79**	**79.78 ± 3.52**	78.51 ± 0.80	78.28 ± 0.97	72.30 ± 1.61	63.07 ± 0.72	**68.83 ± 1.26**
Our Ranks	1	2	1	1	4	2	5	5	1

test regarding transferability. In the testing procedure, the models are fine-tuned on multiple datasets and evaluated by the labels of these datasets. We adopt the GIN encoder used in [83] with the same settings for fair comparison. All reported values for baseline methods are taken directly from [83] and [94]. For the fine-tuning, the encoder has an additional linear graph prediction layer on top which is used to map the representations to the task labels. This is trained end-to-end using gradient descent (Adam).

Hyper-parameters. Due to the large pre-train dataset size and multiple fine-tune datasets finding optimal λ_{reg} for each of them can become time consuming. Instead we use a fixed $\lambda_{\text{reg}} = 5.0$ as it provides reasonable performance. The learning rate is also fixed to 0.001 and is symmetric for both the encoder and augmenter during self-supervision on the pre-train dataset. The number of training epochs for pre-training is chosen among $\{20, 50, 80, 100\}$ based on the validation performance on the fine-tune datasets. The same learning setting for fine-tuning is used by following [94].

Results. We evaluate the GNN encoders trained by AD-GCL on transfer learning to predict chemical molecule properties and biological protein functions. We follow the setting in [83] and use the same datasets: GNNs are pre-trained on one dataset using self-supervised learning and later fine-tuned on another dataset to test out-of-distribution performance. Here, we only consider AD-GCL-FIX as AD-GCL-OPT is only expected to have better performance. We adopt baselines including no pre-trained GIN (*i.e.,* without self-supervised training on the first dataset and with only fine-tuning), InfoGraph [84], GraphCL [94], three different

pre-train strategies in [83] including edge prediction, node attribute masking and context prediction that utilize edge, node and subgraph context respectively.

According to Table 5.6, AD-GCL-FIX significantly outperforms baselines in 3 out of 9 datasets and achieves a mean rank of 2.4 across these 9 datasets which is better than all baselines. Note that although AD-GCL only achieves 5th on some datasets, AD-GCL still significantly outperforms InfoGraph [84] and GraphCL [94], both of which are strong GNN self-training baselines. In contrast to InfoGraph [84] and GraphCL [94], AD-GCL achieves some performance much closer to those baselines (EdgePred, AttrMasking and ContextPred) based on domain knowledge and extensive evaluation in [83]. This is rather significant as our method utilizes only edge dropping GDA, which again shows the effectiveness of the AD-GCL principle.

5.5.4 Semi-Supervised Learning Setting

Evaluation Protocol. We follow the protocol as mentioned in [94]. In this setting, the self-supervised methods are pre-trained and later fine-tuned with 10% true label supervision on the same dataset. The representations generated by the methods are finally evaluated using 10-fold evaluation. All reported values for baseline methods are taken directly from [94]. For fine-tuning, the encoder has an additional linear graph prediction layer on top which is used to map the representations to the task labels. This is trained end-to-end by using gradient descent (Adam).

Hyper-parameters. For the pre-training our model, a fixed $\lambda_{\text{reg}} = 5.0$ and learning rate of 0.001 for both encoder and augmenter is used. The epochs are selected among $\{20, 50, 80, 100\}$ and finally for fine-tuning with 10% label supervision, default parameters from [94] are used.

Results. We evaluate AD-GCL on semi-supervised learning for graph classification on the benchmark TU datasets [223]. Again, we only consider AD-GCL-FIX and compare it with several baselines in [94]: 1) no pre-trained GCN, which is directly trained by the 10% labels from scratch, 2) SS-GCN-A, a baseline that introduces more labelled data by creating random augmentations and then gets trained from scratch, 3) a predictive method GAE [86]

Table 5.7. Semi-supervised learning performance with 10% labels on TU datasets [223] (10-Fold Accuracy (%)± std over 5 runs). **Bold**/**Bold*** indicate our methods outperform baselines with $\geq$ 0.5 std/ $\geq$ 2 std respectively.

Dataset	NCI1	PROTEINS	DD	COLLAB	RDT-B	RDT-M5K
No Pre-Train	73.72 ± 0.24	70.40 ± 1.54	73.56 ± 0.41	73.71± 0.27	86.63 ± 0.27	51.33 ± 0.44
SS-GCN-A	73.59 ± 0.32	70.29 ± 0.64	74.30 ± 0.81	74.19 ± 0.13	87.74 ± 0.39	52.01 ± 0.20
GAE [86]	74.36 ± 0.24	70.51 ± 0.17	74.54 ± 0.68	75.09 ± 0.19	87.69 ± 0.40	53.58 ± 0.13
InfoGraph [84]	74.86 ± 0.26	72.27 ± 0.40	75.78 ± 0.34	73.76 ± 0.29	88.66 ± 0.95	53.61 ± 0.31
GraphCL [94]	74.63 ± 0.25	74.17 ± 0.34	76.17 ± 1.37	74.23 ± 0.21	89.11 ± 0.19	52.55 ± 0.45
AD-GCL-FIX	**75.18 ± 0.31**	73.96 ± 0.47	**77.91 ± 0.73***	**75.82 ± 0.26***	**90.10 ± 0.15***	53.49 ± 0.28
Our Ranks	1	2	1	1	1	3

that utilizes adjacency reconstruction in the pre-training phase, and GCL methods, 4) InfoGraph [84] and 5) GraphCL [94]. Note that here we have to keep the encoder architecture same and thus AD-GCL-FIX adopts GCN as the encoder. Table 5.7 shows the results. AD-GCL-FIX significantly outperforms baselines in 3 out of 6 datasets and achieves a mean rank of 1.5 across these 6 datasets, which again demonstrates the strength of AD-GCL.

5.5.5 Additional Baseline Comparisons for AD-GCL

We first clarify the different mechanisms that JOAO [220] and AD-GCL adopt. JOAO selects augmentation families from a pool $\mathcal{A}$ = NodeDrop, Subgraph,EdgePert, AttrMask,Identical and defines a uniform prior on them for their inner optimization over all possible augmentation family pairs. (See Section 3.2 and See Eq. 7,8 in [220]). An important distinction is that JOAO still adopts uniformly random augmentations and the inner optimization only searches over different pairs of uniform augmentations. Whereas, AD-GCL adopts non-uniformly random augmentations, which essentially corresponds to a much larger search space.

Complexity wise, JOAO is more expensive than AD-GCL as, they utilize projected gradient descent to fully optimize the inner optimization step over all possible augmentations $\mathcal{A}$. This is a factor k more expensive than AD-GCL. The factor k in JOAO is currently $|\mathcal{A}|^2 = 4^2 = 16$. This makes it slow to train while still having a restricted search space compared to AD-GCL which on the other hand is both faster and looks at a larger search

space for a given augmentation family. In our experiments on a single GPU, JOAO took 3.2 hrs for training on COLLAB whereas AD-GCL only took 14.4 mins (0.24 hrs). Moreover, we derive the min-max principle in a more principled way by illustrating its connection to graph information bottleneck (Theorem 5.2.1), which explains the fundamental reason and benefits of optimizing graph augmentation strategies.

Experimental Comparison. We provide comparison between JOAO and AD-GCL in unsupervised learning setting with the standard non-linear downstream classifier setting in Table 5.8. This is done following [220] for fair comparison. Now, we provide the comparison between JOAO and AD-GCL using a linear evaluation protocol for unsupervised setting in Table 5.9. Specifically, a linear SVM head is used for evaluating the representations learned by the 2 methods for the downstream task. The regularization hyper-parameter of the linear svm is grid-searched among 0.001, 0.01,0.1,1,10,100,1000. We re-run the code provided by authors of JOAO (available at https://github.com/Shen-Lab/GraphCL_Automated) with the default parameters for 5 times each with different seeds. The only change done is to the embedding evaluation code to include linear svm as the final prediction head. For all the TU datasets used here, standard 10-Fold evaluation is used to report classification accuracy (%).

The results in the above table further indicate that AD-GCL performs better than JOAO in 6 of the 8 TU benchmark datasets. The gap in performance is even more clear compared to the non-linear evaluation setting as shown previously in Table 5.8. Again, we reiterate that the improved performance gains are due to AD-GCL's search of non-uniformly random augmentations.

Table 5.8. Unsupervised learning showing Averaged Accuracy (%) ± std. with a non linear SVM downstream classifier and same standard setup as used in [220]. The results for JOAO and JOAOv2 are taken from [220].

Dataset	NCI1	PROTEINS	DD	MUTAG	COLLAB	RDT-B	RDT-M5K	IMDB-B
JOAO	78.07±0.47	74.55±0.41	77.32±0.54	87.35±1.02	69.50±0.36	85.29±1.35	55.74±0.63	70.21±3.08
JOAOv2	78.36±0.53	74.07±1.10	77.40±1.15	87.67±0.79	69.33±0.34	86.42±1.45	56.03±0.27	70.83±0.25
AD-GCL-FIX	75.77±0.50	75.04±0.48	75.38±0.41	88.62±1.27	74.79±0.41	92.06±0.42	56.24±0.39	71.49±0.98

Table 5.9. Unsupervised learning showing Averaged Accuracy (%) ± std. with a linear downstream classifier. JOAOv2 results using linear evaluation is obtained by us using code provided by the authors.

Dataset	NCI1	PROTEINS	DD	MUTAG	COLLAB	RDT-B	RDT-M5K	IMDB-B
JOAOv2 (FIX-gamma=0.1)	72.99±0.75	71.25±0.85	66.91±1.75	85.20±1.64	70.40±2.21	78.35±1.38	45.57±2.86	71.60±0.86
AD-GCL-FIX	69.67±0.51	73.59±0.65	74.49±0.52	89.25±1.45	73.32±0.61	85.52±0.79	53.00±0.82	71.57±1.01

Table 5.10. Transfer learning results showing mean ROC-AUC ± std. Pre-Training done using ZINC 2M (used for first 8 fine-tune datasets) and PPI-306K (for the last PPI fine-tune dataset). The results for JOAO and JOAOv2 are taken from [220]. The experimental setting follows [220].

Fine-Tune Dataset	BBBP	Tox21	SIDER	ClinTox	BACE	HIV	MUV	ToxCast	PPI
JOAO	70.22±0.98	74.98±0.29	59.97±0.79	81.32±2.49	77.34±0.48	76.73±1.23	71.66±1.43	62.94±0.48	64.43±1.38
JOAOv2	71.39±0.92	74.27±0.62	60.49±0.74	80.97±1.64	75.49±1.27	77.51±1.17	73.67±1.00	63.16±0.45	63.94±1.59
AD-GCL-FIX	70.01±1.07	76.54±0.82	63.28±0.79	79.78±3.52	78.51±0.80	78.28±0.97	72.30±1.61	63.07±0.72	68.83±1.26

Table 5.11. Semi-supervised Learning with 10% label rate showing 10-Fold Accuracy (%). The results for JOAO and JOAOv2 are taken from [220]. The experimental setting follows [220].

Dataset	NCI1	PROTEINS	DD	COLLAB	RDT-B	RDT-M5K
JOAO	74.48±0.27	72.13±0.92	75.69±0.67	75.30±0.32	88.14±0.25	52.83±0.54
JOAOv2	74.86±0.39	73.31±0.48	75.81±0.73	75.53±0.18	88.79±0.65	52.71±0.28
AD-GCL-FIX	75.18±0.31	73.96±0.47	77.91±0.73	75.82±0.26	90.10±0.15	53.49±0.28

More comparison on transfer learning and semi-supervised learning is put in Table 5.10 and Table 5.10 respectively, where the experimental settings follow Sec. 5.5. For transfer learning, AD-GCL outperforms JOAO in 7 among 9 datasets, JOAOv2 in 5 among 9 datasets. For semi-supervised learning, AD-GCL outperforms both of them in all 6 datasets.

5.6 Detailed Analysis of Regularizing AD-GCL

Here, we study how different λ_{reg}'s impact the expected edge drop ratio of AD-GCL at the saddle point of Eq.5.15 and further impact the model performance on the validation datasets. The main hyper-parameter for our method AD-GCL is the regularization strength λ_{reg}. Detailed sensitivity analysis is provided in Figures 5.3, 5.5 and 5.4. For the method AD-GCL-OPT, we tune λ_{reg} over the validation set among $\{0.1, 0.3, 0.5, 1.0, 2.0, 5.0, 10.0\}$.

For the ablation study, i.e. NAD-GCL-OPT the random edge drop ratio is tuned over the validation set among $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$.

5.6.1 Key Takeaways

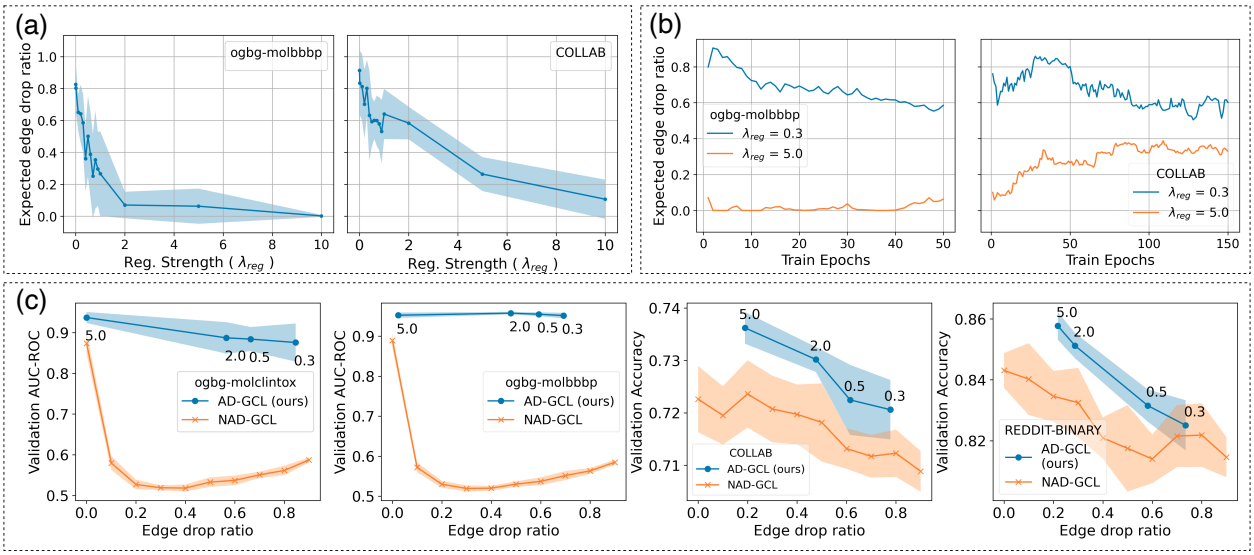

Figure 5.3. (a) λ_{reg} *v.s.* expected edge drop ratio $\mathbb{E}_{\mathcal{G}}[\sum_e \omega_e/|E|]$ (measured at saddle point of Eq.5.15). (b) Training dynamics of expected drop ratio for λ_{reg}. (c) Validation performance for graph classification *v.s.* edge drop ratio. Compare AD-GCL and GCL with non-adversarial edge dropping. The markers on AD-GCL's performance curves show the λ_{reg} used.

As shown in Figure 5.3(a), a large λ_{reg} tends to yield a small expected edge drop ratio at the convergent point, which matches our expectation. λ_{reg} ranging from 0.1 to 10.0 corresponds to dropping almost everything (80% edges) to nothing (<10% edges). The validation performance in Figure 5.3(c) is out of our expectation. We find that for classification tasks, the performance of the encoder is extremely robust to different choices of λ_{reg}'s when trained w.r.t. the AD-GCL principle, though the edge drop ratios at the saddle point are very different. However, the non-adversarial counterpart NAD-GCL is sensitive to different edge drop ratios, especially on the molecule dataset (e.g., ogbg-molclitox, ogbg-molbbbp). We actually observe the similar issue of NAD-GCL across all molecule datasets (See Section 5.6.5). More interesting aspects of our results appear at the extreme cases. When $\lambda_{\text{reg}} \geq 5.0$, the

convergent edge drop ratio is close to 0, which means no edge dropping, but AD-GCL still significantly outperforms naive GCL with small edge drop ratio. When $\lambda_{\text{reg}} = 0.3$, the convergent edge drop ratio is greater than 0.6, which means dropping more than half of the edges, but AD-GCL still keeps reasonable performance. We suspect that such benefit comes from the training dynamics of AD-GCL (examples as shown in Figure 5.3(b)). Particularly, optimizing augmentations allows for non-uniform edge-dropping probability. During the optimization procedure, AD-GCL pushes high drop probability on redundant edges while low drop probability on critical edges, which allows the encoder to differentiate redundant and critical information. This cannot be fully explained by the final convergent edge drop ratio and motivates future investigation of AD-GCL from a more in-depth theoretical perspective.

5.6.2 Optimal Regularization Strength Values

Table 5.12. Optimal λ_{reg} for AD-GCL on validation set that are used for reporting test performance in Tables 5.4 (TOP) and (BOTTOM).

	ogbg-molesol	ogbg-mollipo	ogbg-molfreesolv	ZINC-10K	ogbg-molbace	ogbg-molbbbp	ogbg-molclintox	ogbg-moltox21	ogbg-molsider
AD-GCL-OPT	0.4	0.1	0.3	0.8	10.0	10.0	5.0	10.0	5.0
	COLLAB	RDT-B	RDT-M5K	IMDB-B	IMDB-M	NCI1	PROTEINS	MUTAG	DD
AD-GCL-OPT	5.0	5.0	10.0	2.0	10.0	5.0	1.0	10.0	10.0

Table 5.12 shows the optimal λ_{reg} on the validation set that are used to report test performance in Tables 5.4 (both TOP and BOTTOM).

5.6.3 Effects of Regularization on Graph Classification Tasks

Figure 5.4 shows the complete validation set performance for different edge drop ratios. AD-GCL is compared to a non-adversarial random edge dropping GCL (NAD-GCL). We choose λ_{reg}'s that result in an expected edge drop ratio (measured at saddle point of Eq. 5.15) value to match the random drop ratio used for NAD-GCL.

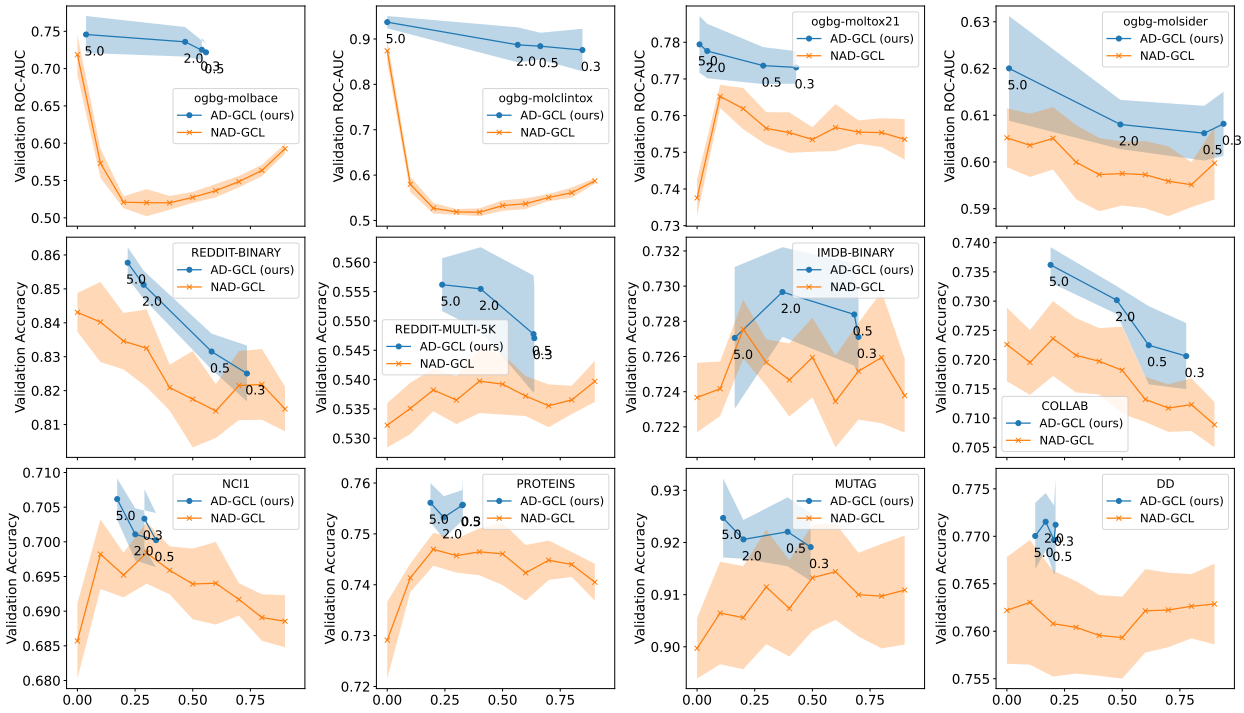

Figure 5.4. Validation performance for graph classification *v.s.* edge drop ratio. Comparing AD-GCL and GCL with non-adversarial random edge dropping. The markers on AD-GCL's performance curves show the λ_{reg} used. Note here that higher validation metric is better.

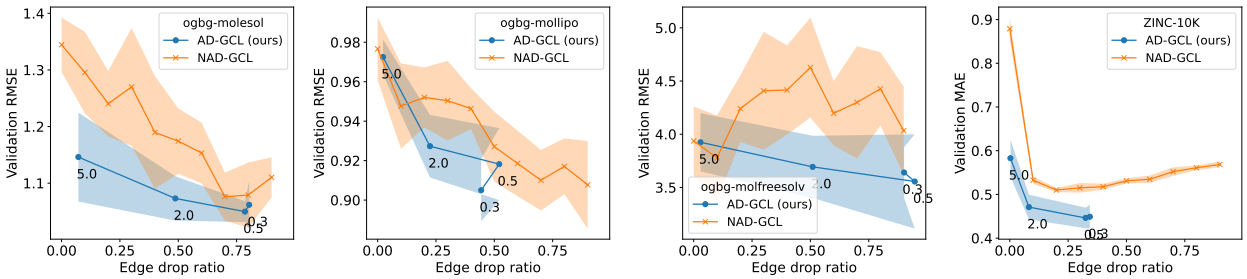

Figure 5.5. Validation performance for graph regression *v.s.* edge drop ratio. Comparing AD-GCL and GCL with non-adversarial random edge dropping. The markers on AD-GCL's performance curves show the λ_{reg} used. Note here that lower validation metric is better.

5.6.4 Effects of Regularization on Graph Regression Tasks

Subplots in the topmost row of Figure 5.5 shows the validation performance for different λ_{reg}'s in AD-GCL and random edge drop ratios in NAD-GCL for regression tasks. These observations show an interesting phenomenon that is different from what we observe in classification tasks: for AD-GCL, small λ_{reg} (which in-turn lead to large expected edge drop ratio) results in better performance. A similar trend can be observed even for NAD-GCL, where large random edge drop ratios results in better performance. However, AD-GCL is still uniformly better that NAD-GCL in that regard. We reason that, regression tasks (different from classification tasks) are more sensitive to node level information rather than structural fingerprints and thus, the edge dropping GDA family might not be the most apt GDA family. Modelling different learnable GDA families is left for future work and these observations motivate such steps.

5.6.5 Effects of Regularization on Edge-Drop Ratio

Figure 5.6 shows how different regularization strengths (λ_{reg}) affects the expected edge drop ratio for multiple datasets. These results further provides us evidence that indeed, λ_{reg} and the expected edge drop ratio are inversely related in accordance with our design objective and thus provides us with a way of controlling the space of augmentations for our learnable edge dropping GDA.

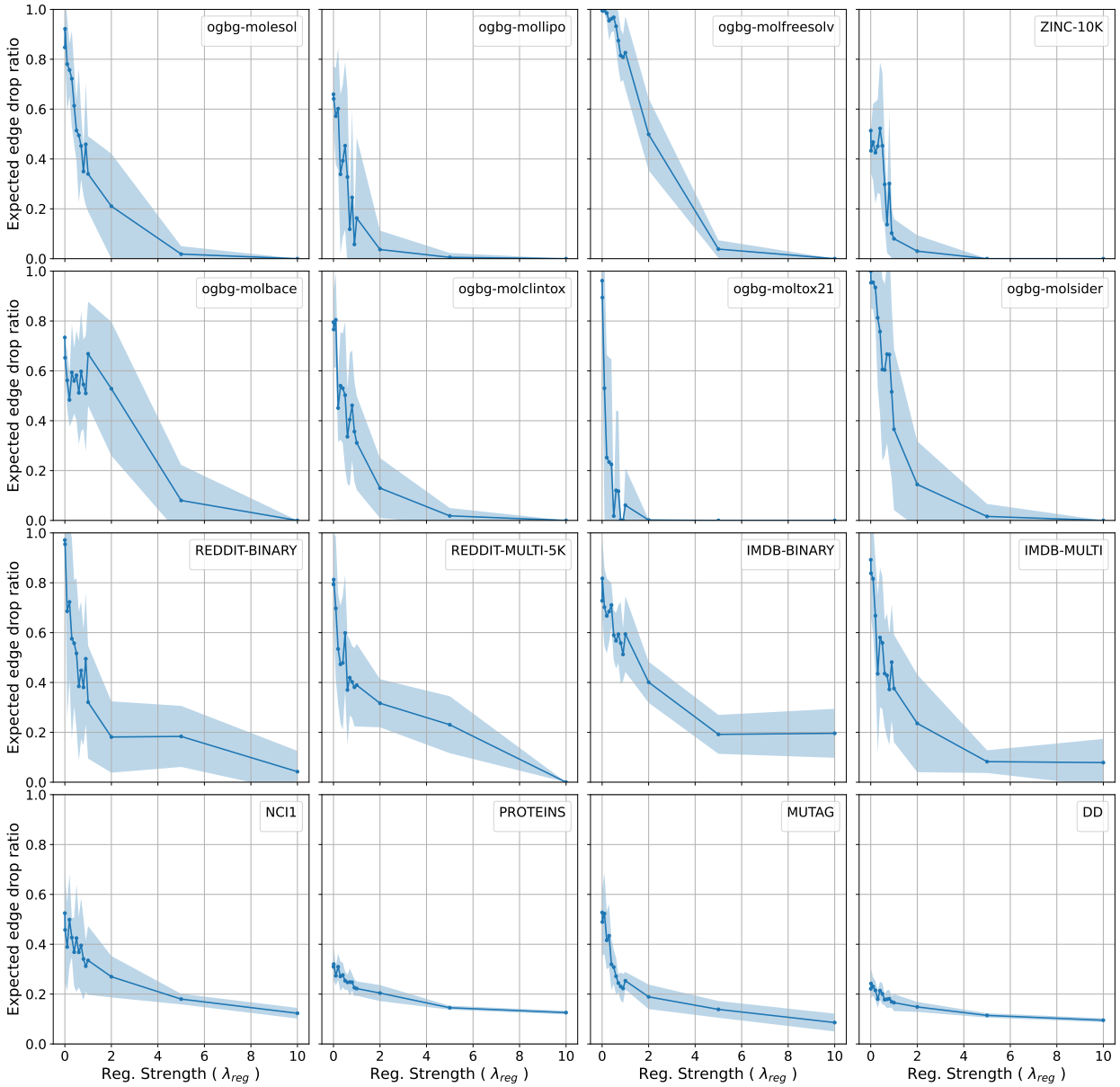

Figure 5.6. λ_{reg} *v.s.* expected edge drop ratio $\mathbb{E}_{\mathcal{G}}[\sum_e \omega_e/|E|]$ (measured at saddle point of Eq.5.15).

5.6.6 Training Dynamics of Regularized AD-GCL

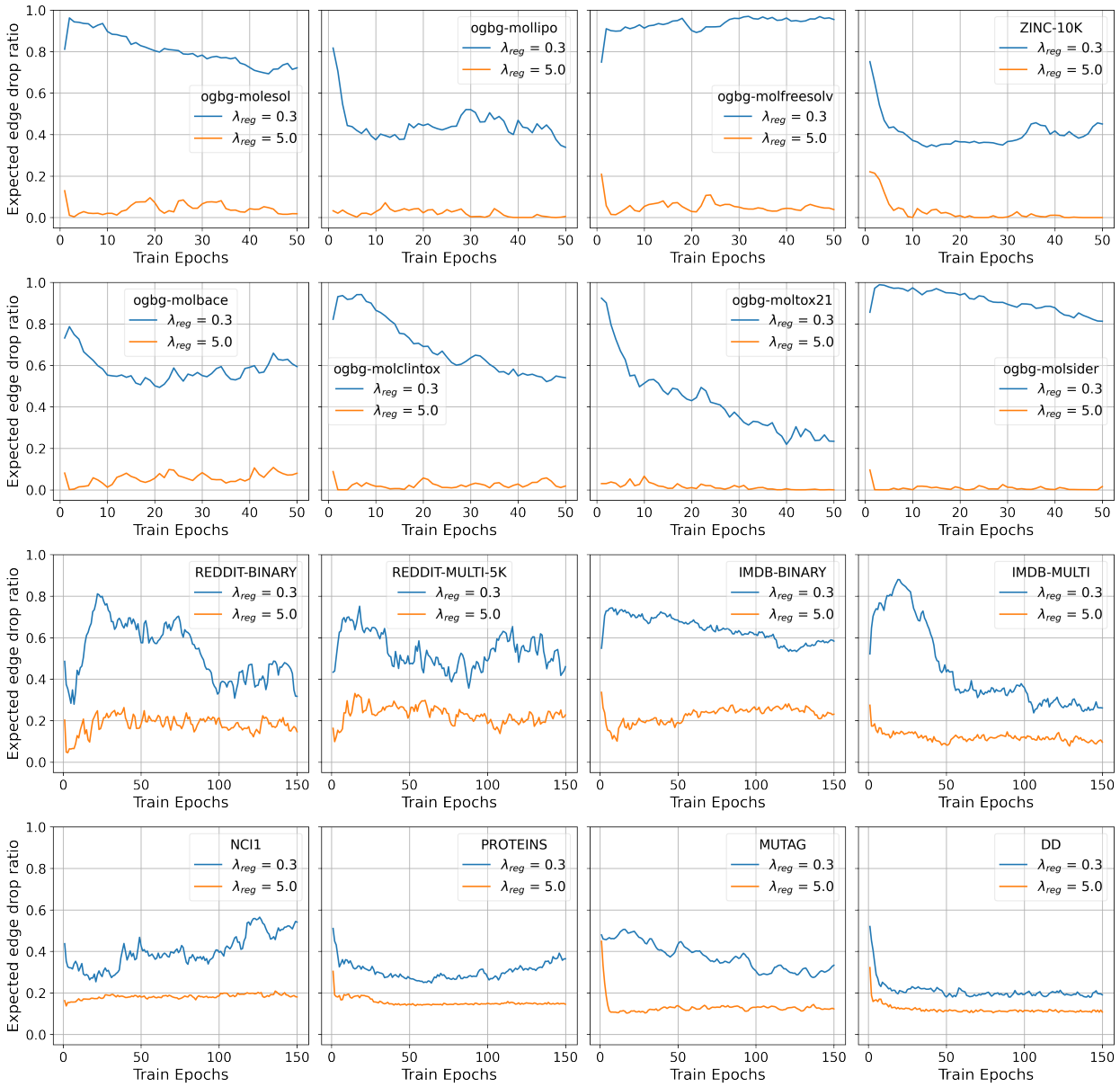

Figure 5.7. Training dynamics of expected edge drop ratio for λ_{reg}.

Figure 5.7 further provides additional plots of the training dynamics of expected edge drop ratio for different λ_{reg} values.

5.7 Discussion

In this work we have developed a theoretically motivated, novel principle: *AD-GCL* that goes a step beyond the conventional InfoMax objective for self-supervised learning of GNNs. The optimal GNN encoders that are agnostic to the downstream tasks are the ones that capture the minimal sufficient information to identify each graph in the dataset. To achieve this goal, AD-GCL suggests to better graph contrastive learning via optimizing graph augmentations in an adversarial way. Following this principle, we developed a practical instantiation based on learnable edge dropping. We have extensively analyzed and demonstrated the benefits of AD-GCL and its instantiation with real-world datasets for graph property prediction in unsupervised, transfer and semi-supervised learning settings.

We stress on the fact that self-supervised methods come with a fundamental set of limitations as they don't have access to the downstream task information. Specifically for contrastive learning, the design of contrastive pairs (done through augmentations) plays a major role as it guides the encooder to selectively capture certain invariances with the hope that it can be beneficial to downstream tasks. Biases could creep in during the design of such augmentations that can be detrimental to the downstream tasks and risk learning of sub-optimal or non-robust representations of input data. Our work helps to alleviate some of the issues of hand designed augmentation techniques and provides a novel principle that can aid in the design of learnable augmentations. It also motivates further research into the understanding the inherent biases of family of augmentations and how they affect the downstream tasks. Finally, self-supervised graph representation learning has a lot of implications in terms of either fairness, robustness or privacy for the various fields that have been increasing adopting these methods.

6. FEDERATED SELF-SUPERVISED GRAPH LEARNING

The widespread adoption of Graph neural networks (GNNs), a class of powerful encoders for graph representation learning [42], [45], [46], [141] have shown enormous potential for downstream applications in a variety of domains spanning social, physical and biochemical sciences [9], [14], [39], [40]. While GNNs have been extensively studied in both supervised [45]–[47], [53], [77], [80], [204] and self-supervised [93]–[97], [230] settings, the bulk of the work falls under a traditional data-centralized training regime. With heightened data security concerns, privacy, and compliance regulations in key GNN application domains such as social networks and healthcare, increasingly, vast amounts of such graph data is siloed away or held behind strict data boundary constraints [231], [232]. Therefore, there is great need for understanding and developing decentralized training processes for GNNs.

Federated learning (FL) [114], [233] has risen as a widely popular distributed learning approach that brings model training processes to the training data held at the clients, thereby avoiding transfer of raw client data. The benefits of FL are two fold, first, it is seen as a key ingredient in enabling privacy-preserving model learning in cross-geo or cross-silo scenarios [116]. Second, when certain participating clients have scarce training data or lack diverse distributions, FL enables them to potentiality leverage the power of data from others—thereby helping them improve performance on their own local tasks [233].

Recent research efforts have looked at applying federated learning algorithms to graph structured data [117]–[119], [123], [234], [235]. However, several interesting and real-world graph data characteristics are not taken into consideration: (1) **Label deficiency** - current methods assume that training labels (node / graph level) for the corresponding tasks [1] are readily available at the clients and a global model is trained end-to-end in a federated fashion. However, in many cross-silo applications, clients might have very little or no labeled data points. It is a well known that annotating labels of node / graph data takes a lot of time and resources [83], [128], e.g., difficulties in obtaining explicit user feedback in social network applications and costly *in vitro* experiments for biological networks. Moreover, certain clients may be unwilling to share labels due to competition or other regulatory reasons.

[1] ↑e.g., classification / regression problems at node or whole graph level.

(2) **Downstream task heterogeneity** - it is reasonable to assume that while clients may share the same graph data domain, the downstream tasks may be client-dependent and vary significantly across clients. It is also reasonable to expect that some clients may have new downstream tasks added at a later point, where a model supervised by previous tasks may be ineffective.

With these observations, we propose a realistic and unexplored problem setting for FedGRL: *Participating clients have a shared space of graph-structured data, though the distributions may different across clients. And, clients have the access to vast amounts of unlabeled data. Additionally, they may have very different local downstream tasks with very few private labeled data points.* Fundamentally, our problem setting asks if one can leverage unlabeled data across clients to learn a shared graph representation (akin to "knowledge transfer") which can then be further personalized to perform well in the local downstream tasks at each client. In a data centralized training regime, a number of works that utilize GNN pre-training [83], [90] and self-supervision [88], [94], [96], [230] have shown the benefits of such approaches in dealing with label deficiency and transfer learning scenarios which motivate us to explore and utilize them for the proposed FedGRL problem setting.

In this chapter, we propose a novel FedGRL formulation (called FedSGL) based on model interpolation where we aim to learn a shared global model that is optimized collaboratively using a self-supervised objective and gets downstream task supervision through local client models. We provide a specific instantiation of our general formulation using BGRL [88] a SoTA self-supervised graph representation learning method and we empirically verify its effectiveness through realistic cross-slio datasets: (1) we adapt the Twitch Gamer Network which naturally simulates a cross-geo scenario and show that our formulation can provide consistent and avg. 6.1% gains over traditional supervised federated learning objectives and on avg. 1.7% gains compared to individual client specific self-supervised training, (2) we construct and introduce a new cross-silo dataset called Amazon Co-purchase Networks that have both the characteristics of the motivated problem setting and (3) we adapt the OGB Molecule datasets [92] to a cross-silo setting to test FedSGL in graph classification tasks. We firstly show how standard supervised federated objectives can result in negative gains (on avg. -4.16%) compared to individual client specific supervised training, due to the increased data

heterogeneity and limited label availability. Then we experimentally verify the effectiveness of our method and witness on avg. 11.5% gains over traditional supervised federated learning and on avg. 1.9% gains over individually trained self-supervised models. For OGB molecule cross-silo dataset, we witness up to avg. +19.3% gains across eight different clients for graph classification task compared to supervised federated learning. Both experimental results point to the effectiveness of our proposed formulation.

The remainder of this chapter is organized as follows, in Sec. 6.1 we review relevant work related to FL for graph structured data, self-supervised techniques for GNNs and finally some recent work on tackling label deficiency with FL. In Sec. 6.2 we introduce notation and some preliminaries, later in Sec. 6.3, we provide a detailed problem setup, introduce our formulation and its instantiations. Later in Sec. 6.4 we provide detailed experimental setup and finally present experimental results in Sec 6.5.

6.1 Related Work

The broad fields of designing GNNs for graph representation learning (GRL) get detailed coverage in recent surveys [39], [40], [141]. We refer the reader to [114], [115], [131] for a an overview of FL methods.

6.1.1 Federated Learning for Graphs

FedGRL is a new research topic and current works have considered the following two main problem formulations.

First, for node-level tasks (predicting node labels), there are three sub categories based on the degree of overlap in graph nodes across clients: (1) No node overlap between client graphs. Here, each client maintains a GNN model which is trained on the local node labels and the server aggregates the parameters of the client GNN models and communicates it back in every federation round [235]–[237]. ASFGNN [236] additionally tackles the non-IID data issue using split based learning and FedGraph [237] focuses on efficiency and utilizes a privacy preserving cross-client GNN convolution operation. FedSage [235] considers a slightly different formulation, wherein each client has access to disjoint subgraphs of some global

graph. They utilize GraphSage [46] and train it with label information and further propose to train a missing neighbor generator to deal with missing links across local subgraphs. (2) Partial node overlap across clients. Here, each participating client holds subgraphs which may have overlapping nodes with other clients graphs. GraphFL [234] considers this scenario and utilizes a meta-learning based federated learning algorithm to personalize client models to downstream tasks. [238] considers overlapping nodes in local client knowledge graphs and utilize them to translate knowledge embedding across clients. (3) Complete node overlap across clients. Here all clients hold the same set of nodes, they upload node embeddings instead of model parameters to the server for FL aggregation. Existing works focus on the vertically partitioned citation network data [239], [240]. Note that all the above problem settings are different from ours in motivation as we focus on label deficiency and downstream task heterogeneity.

Secondly, for graph-level tasks (predicting graph labels), each client has a local set of labeled graphs and the goal is to learn one global model or personalized local models using federation. This problem setting is fundamentally similar to other federated learning settings widely considered in vision and language domains. One needs to replace the previous linear/DNN encoder into a graph kernel/GNN encoder to handle the graph data modality. [117] creates a benchmark towards this end. The issues of client data non-IID ness carry over to the graph domain as well and [123] utilizes client clustering to aggregate model parameters.

6.1.2 Self-Supervised Learning for GNNs

Early SSL techniques for GNNs adopted the edge-reconstruction principle, where the edges of the input graph are expected to be reconstructed based on the output of GNNs [46], [86]. Recently, contrastive methods [214] effective on images have been successfully adapted to self-supervised GNN training. The main focus of these methods are in designing contrastive pairs for irregular graph structures unlike images and thus becomes more challenging. Some works use different parts of a graph to build contrastive pairs, including nodes v.s. whole graphs [84], [93], nodes v.s. nodes [241], nodes v.s. subgraphs [219]. Other

works have also employed graph data augmentation like edge perturbation, node dropping, subgraph sampling for building contrastive pairs and have been commonly applied for both node and graph level representation learning [94], [95], [230], [242]. However, due to the need for costly negative sampling to generate contrastive pairs, large negative examples and batch sizes, these contrastive based methods suffer from scalability and efficiency issues when applied to large scale graphs [88]. Recent efforts have adapted Siamese representation learning techniques [243] that do not rely on negative examples, but contrast representations from GNN encoders on different augmentations of a graph for e.g., BGRL [88], SelfGNN [244] and DGB [245]. It is important to note that our general self-supervised FedGRL formulation can be instantiated using any of the above SSL techniques developed for GNNs.

6.1.3 Label Deficiency and FL

There are also some very recent developments in handling the label deficiency issue within FL mainly for vision domains that are relevant to our work [246]–[250]. FedCA [246] and FedU [247] are based on direct extension of self-supervised methods SimCLR [214] and BYOL [251] to the federated regime respectively. FedEMA [248] extends FedU to further address the non-IID data issue by controlling the divergence between global and local models dynamically using exponential moving average. Orchestra [250] also employs an SSL based method during federation and addresses the non-IID problem using client clustering. SSFL [249] utilizes SimSiam [243], a siamese SSL method in conjunction to both standard and personalized FL algorithms viz., FedAvg [233], perFedAvg [130], Ditto [252]. While these methods have utilized and shown the benefits of using unlabeled data within federated learning for image domain, they have not shown the experimental efficacy of their methods for real world datasets with downstream task heterogeneity across clients, which heterogeneity graph learning tasks often contain.

6.2 Preliminaries

We consider a graph $G = (V, E)$ with node set $V = \{v_1, \dots v_N\}$ and edge set $E \subseteq V \times V$. Additionally, we use node feature matrix $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ and adjacency matrix $\boldsymbol{A} \in \{0, 1\}^{N \times N}$

where, $N = |V|$ is the number of nodes in the graph, F is the node feature dimension and $\boldsymbol{A}_{i,j} = 1$ iff $(v_i, v_j) \in E$.

6.2.1 Graph Neural Networks

A GNN encoder $f(\boldsymbol{X}, \boldsymbol{A})$ or equivalently[2] $f(G)$, receives the node features and graph structure to produce node representations $\boldsymbol{H} \in \mathbb{R}^{N \times F'}$ of dimension $F' \ll F$ which is defined using a layer-wise propagation rule for e.g., GCN [45] defined as follows:

$$\boldsymbol{H}_{(l+1)} = \sigma\left(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}_{(l)}W_{(l)}\right) \tag{6.1}$$

where, $\tilde{\boldsymbol{A}} = \boldsymbol{A} + I$ is the adjacency matrix with added self loops, $\tilde{\boldsymbol{D}}$ is the degree matrix with $\tilde{\boldsymbol{D}}_{ii} = \sum_j \tilde{\boldsymbol{A}}_{ij}$. $W_{(l)}$ is trainable weight matrix for layer l and $\boldsymbol{H}_{(l)}$ is the intermediate l^{th} layer node representation matrix with $\boldsymbol{H}_{(0)} = \boldsymbol{X}$. We omit the layer index and use $\boldsymbol{H} = \boldsymbol{H}_{(l+1)}$ to denote the final node representations produced by the l-layer GNN. These representations can be used for downstream tasks, such as node classification.

6.2.2 Federated Optimization

In standard federated optimization, we are given a set of clients C. A client $c \in C$ holds n_c amount of private data drawn from distribution $\mathcal{D}_c$ and the goal is to learn $f_c : \mathcal{X}_c \to \mathcal{Y}_c$ for each c and the overall distributed optimization is of the form [233]:

$$\min_{\{f_c\}} \sum_{c \in \mathcal{C}} \frac{n_c}{n} \mathcal{L}_c(f_c) \tag{6.2}$$

where, $n = \cup_{c \in \mathcal{C}} n_c$ is total amount of data. For each client c, the expected loss over its data distribution is,

$$\mathcal{L}_c(f_c) = \mathbb{E}_{(x_c, y_c) \sim \mathcal{D}_c}[\ell_c(f_c, x_c, y_c)], \tag{6.3}$$

where, (x_c, y_c) represents data and labels in client c and $\ell_c(f, x_c, y_c)$ is the loss function for e.g., cross-entropy loss for classification problems.

[2] ↑later we omit $\boldsymbol{X}$ and $\boldsymbol{A}$ and simply denote the input as a graph G for brevity.

6.3 Method

In this section, we describe the problem setting for self-supervised FedGRL, provide a general framework and then introduce a specific instantiation.

6.3.1 Problem Setting

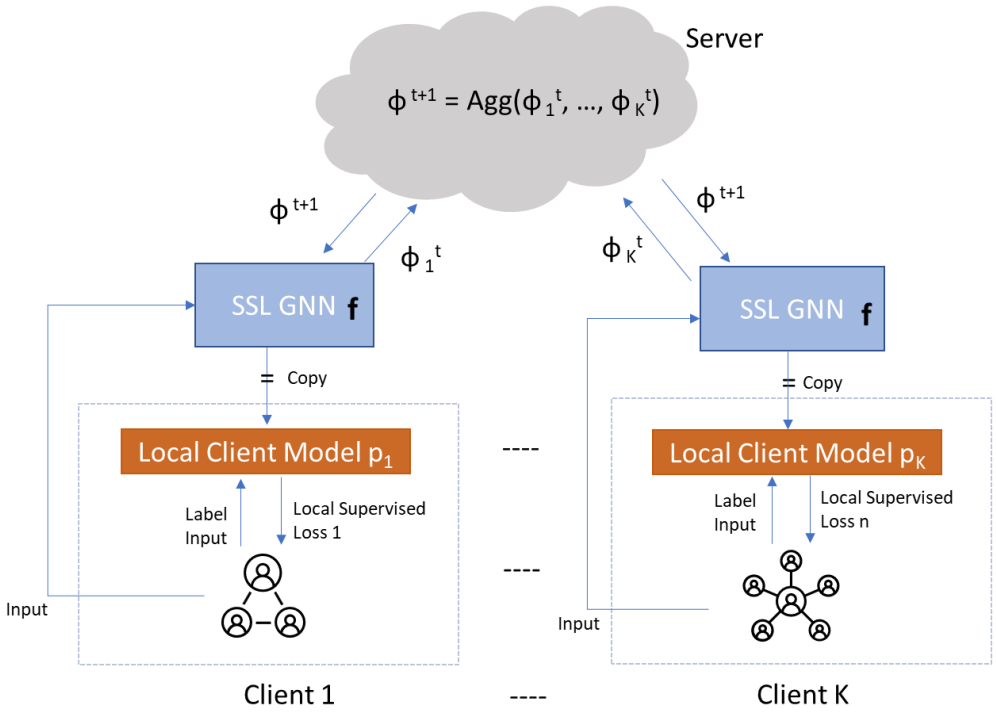

Figure 6.1. Overview of the proposed general formulation for self-supervised FedGRL. Here, we learn a shared global GNN model f using a self-supervised (SSL) objective, collaboratively based on federated learning. With the copy gate, f is only trained on unlabeled data through SSL and local client models p_c are further trained individually on top of f with label supervision. This forms our first learning protocol (**LP1**) . Without the copy gate, in addition to SSL objective, f can further receive label supervision from client tasks and this forms the second learning protocol (LP2).

Each client $c \in \mathcal{C}$ holds a graph $G_c = (V_c, E_c)$ and has a downstream task T_c if one exists. Let $N_c = |V_c|$ be the number of nodes in client graph. The node set V_c is partitioned into labeled and unlabeled nodes as, $V_c = V_c^l \cup V_c^u$. If a certain client has no downstream task, then $V_c^l = \emptyset$. A client c with task T_c has access to a space of node label $\mathcal{Y}_c$ of size

m_c. Different clients may have entirely different tasks and thus $\mathcal{Y}_c$'s may not overlap across clients. Each node in V_c^l gets a label from $\mathcal{Y}_c$ to form its corresponding set of node labels Y_c. The task T_c is then to infer the labels for unlabeled nodes in V_c^u. Note that there could be multiple tasks for each client but it will not change the foundation of the problem. So, we simply consider one task per client.

There are several differences compared to the traditional FL setting: (1) Label scarcity i.e., for each client c, $|V_c^l|$ could be very small or even zero. (2) Total labeled data may be far less than unlabeled data i.e., $\sum_c |V_c^l| \ll \sum_c |V_c^u|$ (3) Poor data quality i.e., for some clients $|V_c^l| + |V_c^u|$ may also be small. (4) Downstream task heterogeneity i.e., across clients, the class label domain $\mathcal{Y}_c$'s could be very different.

6.3.2 General Formulation

Directly optimizing $\{f_c\}_{c \in C}$ using the standard FL optimization in Eq. 6.2 and 6.3 with labels, may not achieve good results as $\sum_c |V_c^l| \ll \sum_c |V_c^u|$. So, instead, we propose to utilize a model interpolation based formulation where a global shared model and client specific local models are simultaneously optimized using self-supervision and label supervision if downstream task labels are available. Specifically, we set $f_c = p_c \circ f$, where p_c is the local client model, f is the global shared model. We then propose to empirically solve the following distributed objective:

$$\min_{\{p_c\}, f} \sum_{c \in \mathcal{C}} \Big[\mathcal{L}_c(p_c, f, G_c, Y_c) \mathbb{1}_{T_c \text{ exists}} + \lambda_c \tilde{\mathcal{L}}_c(f, G_c) \Big] \tag{6.4}$$

where, the first term is the objective supervised by labels and the second term is the self-supervised objective that doesn't utilize label information, λ_c controls the amount of model interpolation. Note that f is shared while p_c is not shared. It is important to point out that, by setting each λ_c to 0.0, the objective in Eq. 6.4 falls back to standard supervised FL optimization. An overview of the formulation is shown in Fig 6.1. From the general formulation in Eq. 6.4, we propose two practical learning protocols (LPs).

- **LP1**: The first supervised objective term is ignored, each λ_c is set to 1.0 and a global model f is learnt in a federated self-supervised fashion using the objective in the second term. Notice that here, labeled data is not used to train f during federation. Once such a f is trained, each client c can further train a task specific local model p_c themselves again by **freezing** or **finetuning** on top of f.
- **LP2**: Both terms are used with a non zero λ_c during federation. Both p_c and f get label data supervision if available through the first term. Additionally, f also gets trained using the self-supervised objective through the second term.

We will **only** consider the first learning protocol **(LP1)** for further experimentation in Sec 6.5 and leave the study of **LP2** for future work.

6.3.3 Self-Supervised Objective

While in theory, the self-supervised loss term $\tilde{\mathcal{L}}_c(\cdot)$ in Eq 6.4 can be instantiated by using any self-supervised objectives, for practical reasons, more considerations are needed. Methods like GRACE [242] and GCA [89] learn node representations by maximizing the agreement between the same node pairs from two augmented versions of a graph while simultaneously minimizing the agreement between every other node pair. This leads to a scalability issue on large graphs as they require costly (time and space complexity quadratic in the number of nodes) negative sampling for building contrastive pairs [88]. Moreover, in practice these methods rely on a large number of such negatives to be effective. We adopt BGRL [88] which is a scalable and SoTA for node level self-supervised learning of GNNs. BGRL scales linearly with the the number of nodes in the input graph as it does not require contrasting negative node pairs. Fig. 6.2 shows an overview of the approach.

Instantiating using BGRL. We describe the method for a single client and omit the client subscript c for brevity. Specifically, BGRL first produces two alternate but correlated views of the input graph G: $G^1 = \mathcal{T}^1(G)$ and $G^2 = \mathcal{T}^2(G)$, by using stochastic graph augmentations $\mathcal{T}^1$ and $\mathcal{T}^2$ respectively. Examples include node feature masking and edge masking. BGRL, then utilizes two GNN encoders, an online $\omega(\cdot)$ and a target $\tau(\cdot)$ encoder. The online encoder produces online node representations for the first graph $\boldsymbol{H}^1 = \omega(G^1)$ and

similarly, the target encoder produces target node representations of the second augmented graph as, $\boldsymbol{H}^2 = \tau(G^2)$. Then, the online node representations are fed to a node level head $s(\cdot)$ that outputs predictions of the target node representations as $\boldsymbol{Z}^1 = s(\boldsymbol{H}^1)$. Then, the self-supervised BGRL objective is as follows,

$$\tilde{\mathcal{L}}(\omega, s, \tau) = -\frac{2}{N} \sum_{i=0}^{N-1} \frac{\boldsymbol{Z}_i^1 \ \boldsymbol{H}_i^{2^\top}}{\|\boldsymbol{Z}_i^1\| \|\boldsymbol{H}_i^2\|} \tag{6.5}$$

where N is the number of nodes in G. It is important to note that gradients of the above objective are taken only w.r.t the parameters of $\omega(\cdot)$ and $s(\cdot)$ and only they get optimized by gradient descent. The parameters of $\tau(\cdot)$ are updated using an exponential moving average of the parameters of $\omega(\cdot)$ [88].

During federation, parameters of online GNN encoder $\omega(\cdot)$ and node level head $s(\cdot)$ are both shared across clients and the server aggregates them in each federation round and sends it back to the clients. We refer to the global shared model $f = \omega$ and is used for the supervised objectives and further downstream tasks. The target GNN encoder $\tau(\cdot)$ is an auxiliary unit and is only used locally within the client for the purpose of self-supervised learning and discarded once training is finished.

6.3.4 Task Specific Supervision

The first loss term $\mathcal{L}_c(\cdot)$ utilizes supervision from the node labels if available. Standard cross-entropy loss over all labeled nodes can be utilized as commonly employed in semi-supervised GNN node classification [45]. For completeness it takes the following form,

$$\mathcal{L}(p_c, f, G_c, Y_c) = -\sum_{i \in V_c^l} \sum_{j=1}^{m_c} Y_{c(ij)} \ln \boldsymbol{Z}_{c(ij)} \tag{6.6}$$

where, $\boldsymbol{Z}_c = \text{softmax}(p_c \circ f(G_c))$ and $\boldsymbol{Z}_c \in \mathbb{R}^{N_c \times m_c}$.

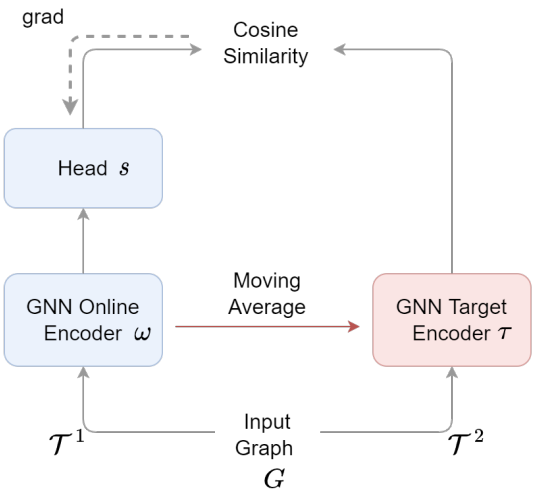

Figure 6.2. Overview of BGRL [88]. Here, two correlated views of input G are obtained using augmentations $\mathcal{T}^1$ and $\mathcal{T}^2$. Further, two GNN encoders ω and τ are used to obtain node representations. The node level head s is tasked at using the representations from ω to predict the representations obtained from τ using a cosine similarity metric. Gradients flow only through s and ω and τ's parameters are updated using an exponential moving average of ω's parameters.

6.4 Experimental Setup

In this section, we first introduce the real-world datasets apt for our proposed problem setting. We then describe the baseline methods, experiment protocol, model architecture and hyper-parameters.

6.4.1 Datasets

Twitch Gamer Networks. These are a set of six user-user social network of gamers who stream in certain countries [162]. Nodes are users and edges represent friendships. Node features are 128-dimensional vectors based on games liked and played, streaming habits and location. These networks are naturally formed in different geographies and share the same feature space which allows us consider them as a cross-silo scenario for our problem setting of federated "knowledge transfer". Additionally, each network also has a binary node

classification task of predicting if a user uses explicit language. This enables us to empirically evaluate the learnt representations. The statistics of these networks are provided in Table 6.1.

Table 6.1. Statistics of Twitch Gamer Networks. Each network from a geography represents a client graph in our experiments.

	twitch-DE	twitch-EN	twitch-ES	twitch-FR	twitch-PT	twitch-RU
#nodes	9,498	7,126	4,648	6,549	1,912	4,385
#edges	153,138	35,324	59,382	112,666	31,299	37,304
density	0.003	0.002	0.006	0.005	0.017	0.004

Amazon Co-purchase Networks. We construct a set of six co-purchase networks from raw Amazon reviews[3] and product meta data [253]. Specifically, we first consider six top level product segments viz. computer, photo, phone, tool, guitar and art. For each segment we construct a co-purchase network where nodes represent various products in the chosen segment and edges signify a frequent co-purchase (using the "also buy" signal in the raw product meta data). Each of these networks is held privately by six different clients simulating a cross-silo scenario. We then build a common review vocabulary of all products considered across all the six networks and use 1000-dimensional bag-of-words product review encoding as node features. This ensures all nodes across graphs share the same node feature space to make federated "knowledge transfer" meaningful and apt to our problem setting. The downstream task for each network is to classify the products (nodes) to fine-grained sub-segments. Again, this construction naturally simulates downstream task heterogeneity as each graph has a different class label domain. The statistics of these networks are provided in Table 6.2.

Table 6.2. Statistics of Amazon Co-purchase Networks. Refer to Fig. 6.3 for detailed label distributions.

	computer	photo	phone	tool	guitar	art
#nodes	10,055	4,705	16,683	4,827	2,506	6,610
#edges	87,512	28,818	113,760	43,458	10,342	90,678
#classes	9	8	7	6	5	8

[3]↑https://nijianmo.github.io/amazon/index.html

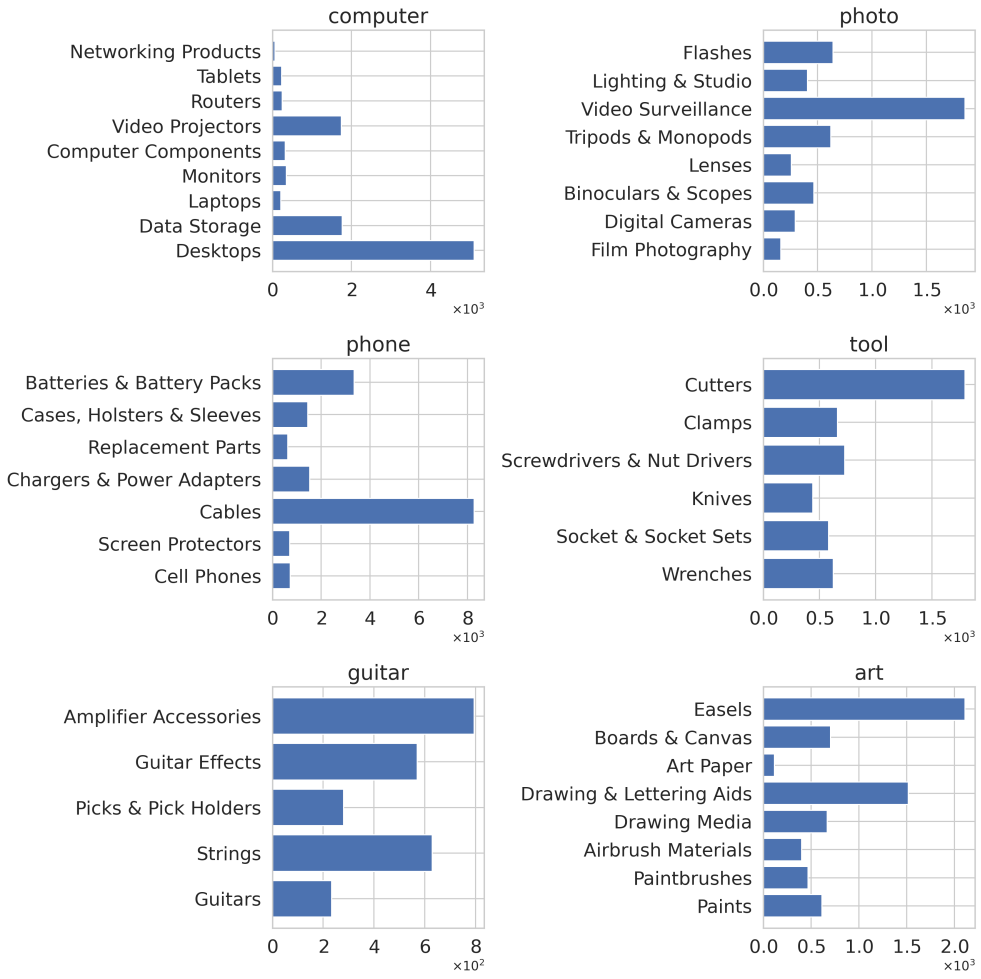

Figure 6.3. Class label distribution for Amazon Co-purchase Networks

OGB Cross-Silo Molecule Datasets For graph level classification tasks we adopt eight molecule datasets from open graph benchmark (OGB) [92]. Each of the datasets contains a set a bio-chemical molecules represented as nodes and edges with attributes. To simulate cross-silo graph level representation learning we distribute each of the eight datasets to eight different clients respectively. It is important to note that all datasets share the same 9-dimensional node feature space consisting of information such as atomic number, chirality and other atom features etc. Each silo consists of a novel domain multi-class binary graph classification task and naturally simulates extreme client task heterogeneity. We simulate the level of label scarcity with 8/10/10 and 80/10/10 scaffold train/val/test splitting strategy. Detailed dataset statistics are provided in Table 6.3. It is clear from the table that each client

has different set of molecules and downstream tasks simulating extreme data heterogeneity among clients.

Table 6.3. Cross-silo molecule datasets for (multi-task) binary graph classification.

	bbbp	bace	tox21	toxcast	sider	clintox	muv	hiv
#Graphs	2,039	1,513	7,831	8,576	1,427	1,477	93,087	41,127
Avg. #Nodes	24.06	34.08	18.57	18.78	33.64	26.15	24.23	25.51
Avg. Degree	51.90	73.71	38.58	38.52	70.71	55.76	52.55	54.93
#Tasks	1	1	12	617	27	2	17	1

6.4.2 Baselines

For baselines, we consider: (1) **NoFed-Sup** which trains a GNN model end-to-end with label supervision, for each client individually without federation. (2) **NoFed-Self-Freeze** which first trains a GNN model using self-supervision and then performs linear evaluation after freezing the GNN node representations, for each client individually without federation. (3) **NoFed-Self-Finetune** which first trains a GNN model using self-supervision and then performs end-to-end task specific fine-tuning with a MLP task head on top, for each client individually without federation. (4) **Fed-Sup** trains a local GNN encoder + local MLP task head with end-to-end label supervision using federated learning. Only the parameters of the local GNN models are shared across clients and their parameters are aggregated in the server via FedAvg [233] and sent back after every communication round. We choose the **No-Fed** class of baselines to experiment whether FL can bring improvements to each client through collaborative training. The **Freeze** and **Finetune** baselines allow us to experiment with the best downstream task evaluation strategy. The **Fed-Sup** baseline is the traditional FL technique that utilizes label information across clients. This baseline allows us to experiment if the formulation (and method) we propose to utilize self-supervision signal within FL brings improvements to each client.

6.4.3 Experiment Protocol

For twitch network experiments, we use a 60/20/20 training, validation and test node random split for evaluation and for amazon network experiments, we use a 10/10/80 random split. Both experiments utilize F1-Micro score as the metric for node classification performance and we report average score results over 10 data splits. For OGB cross-silo molecule dataset, we use ROC-AUC metric for graph classification performance and we report average metric score over 10 data splits.

6.4.4 Parameter Settings

For all baselines and our methods, we employ a 2-layer GCN [45] encoder with hidden dimensions tuned from $\{64, 128, 256\}$ and batch normalization. For supervised baselines **NoFed-Sup** and **Fed-Sup** we use the AdamW [166] optimizer with learning rate of 0.01 and weight decay of 1e-3. For **NoFed-Self** baselines and our **Fed-Self** methods we use the AdamW optimizer with a learning rate of 1e-4 annealed using a cosine scheduler as prescribed in [88] and weight decay of 1e-5. Additionally, the hidden dimension of the MLP predictor model $s(\cdot)$ in BGRL is set as 512. The exponential moving average decay rate in BGRL is initialized as 0.99 and gradually increased to 1.0 using cosine schedule. We perform a small random grid search (20 max trials) over node feature and edge masking probabilities from $\{0.1, 0.2, \ldots, 0.8, 0.9\}$ for both augmentations $\mathcal{T}^1$ and $\mathcal{T}^2$. For all **Fed** methods the number of local epochs is set to be 1 and the number of communication rounds for **Fed-Sup** is set to 500 and **FedSGL** is set to 10,000 with 1,000 warmup rounds. For **Freeze** linear evaluation, we employ a ℓ_2-regularized logistic regression classifier and perform regularization strength search over $\{2^{-10}, 2^{-8}, 2^{-6}, \ldots, 2^{6}, 2^{8}, 2^{10}\}$. For **Finetune** evaluation, we employ a task specific MLP head of 128 dimensions and perform optimization using AdamW for 100 steps with a learning rate of 0.01 and weight decay of 1e-3. For graph level classification tasks (i.e., OGB cross-silo molecule dataset), we instantiate FedSGL with GraphCL [94] for the self-supervised objective with default hyper-parameters. GraphCL is a contrastive objective that was given detailed introduced in Chapter 5.

6.5 Experimental Analysis

In this section, we experimentally compare our instantiated methods against baselines for twitch and amazon networks.

6.5.1 Twitch Network Experiments

Table 6.4. Node classification task on cross-silo Twitch Gamer Networks. Avg F1-Micro Scores over 10 splits and std. dev.

	twitch-DE	twitch-EN	twitch-ES	twitch-FR	twitch-PT	twitch-RU
NoFed (RandInit-GNN)	0.502 ± 0.103	0.501 ± 0.047	0.496 ± 0.198	0.510 ± 0.131	0.533 ± 0.156	0.507 ± 0.248
NoFed-Sup	0.673 ± 0.010	0.584 ± 0.014	0.725 ± 0.014	0.626 ± 0.016	0.678 ± 0.023	0.751 ± 0.015
NoFed-Self-Freeze	0.685 ± 0.009	0.617 ± 0.012	0.731 ± 0.010	0.626 ± 0.015	0.696 ± 0.021	0.752 ± 0.009
NoFed-Self-Finetune	0.703 ± 0.012	**0.665 ± 0.023**	0.746 ± 0.015	0.635 ± 0.018	0.710 ± 0.026	**0.762 ± 0.011**
Fed-Sup (FedAvg)	0.677 ± 0.009	0.600 ± 0.009	0.723 ± 0.013	0.627 ± 0.018	0.683 ± 0.013	0.743 ± 0.015
FedSGL-Freeze (ours)	0.686 ± 0.007	0.606 ± 0.012	0.733 ± 0.007	0.626 ± 0.014	0.699 ± 0.022	0.752 ± 0.009
FedSGL-Finetune (ours)	**0.706 ± 0.013**	0.657 ± 0.024	**0.745 ± 0.013**	**0.636 ± 0.021**	**0.712 ± 0.024**	0.761 ± 0.014

Table 6.4 shows the results across all Twitch clients graphs. Firstly, it is clear that both NoFed-Self-Freeze and NoFed-Self-Finetune perform better than the NoFed-Sup baseline for all the clients. This shows the usefulness of the self-supervised objective as it can make use of the available unlabeled data in each client albeit individually. Secondly, we witness that fine-tuning based evaluation (NoFed-Self-Finetune) leads to the best gain of 4.75% averaged across all clients over No-Fed-Sup. The individual client gains are shown in Fig. 6.4(a). Thirdly, Fed-Sup is only marginally better (0.45 % avg. gain) than No-Fed-Sup, indicating that a supervised federated baseline is not effective in of properly utilizing all the client label information. In fact, for certain clients we see negative gains as shown in Fig 6.4(b). Fourthly, our method **Fed-Self-Finetune** provides a gain of on avg. 6.13% across all clients over Fed-Sup and Fig 6.4(c) shows individual client gains we witness. This provides the evidence of the benefits in using self-supervised objective for learning the global model during federation. We reason that the heterogeneity across these client graphs is due to the fact that they are naturally formed in different geographies with diverse friendship cultures and language-use patterns. In such a scenario, standard supervised FedAvg suffers (Fed-Sup) due to the reliance on labels alone to learn representations, whereas utilizing unlabeled data

and performing self-supervised FedAvg can learn shared patterns and provide more robust representations for each client (Fed-Self-Finetune). Finally, our method **Fed-Self-Finetune** provides consistent gains over individually trained No-Fed-Self-Finetune baseline and overall we witness an avg. 1.73% gain across all clients (see also Fig 6.4(d)). This further shows that we can indeed utilize unlabeled data collaboratively using federation and justifies for formulation. This can also be attributed statistically, as there is an increase in effective unlabeled data which the shared GNN encoder model can leverage.

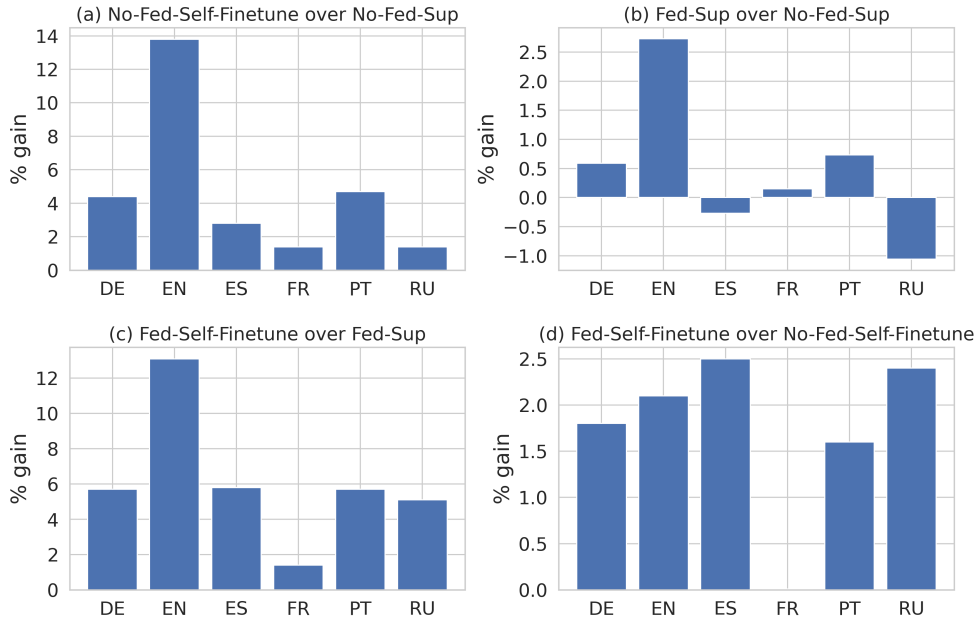

Figure 6.4. Performance gains in Twitch network experiments

6.5.2 Amazon Network Experiments

Table 6.5 shows the results across all Amazon client graphs. Firstly, No-Fed-Self-Freeze is clearly superior to No-Fed-Sup, which shows that self-supervised learning is effective for these graphs with an avg., gain of 4.3% across all clients and Fig 6.5(a) shows the gains individually. Note that, the number of train node labels for amazon graphs are extremely limited—we consider only 10% of all nodes as train nodes. In this case, No-Fed-Sup suffers from over-fitting, where as No-Fed-Self is able to utilize the unlabeled nodes effectively—

Table 6.5. Node classification task on cross-silo Amazon Co-purchase Networks. Avg. F1-Micro Scores over 10 splits and std. dev.

	computer	photo	phone	guitar	tool	art
NoFed (RandInit-GNN)	0.687 ± 0.005	0.709 ± 0.008	0.673 ± 0.005	0.737 ± 0.014	0.622 ± 0.008	0.653 ± 0.009
NoFed-Sup	0.787 ± 0.009	0.801 ± 0.008	0.759 ± 0.007	0.866 ± 0.012	0.764 ± 0.012	0.741 ± 0.010
NoFed-Self-Freeze	0.837 ± 0.004	0.841 ± 0.007	**0.778 ± 0.003**	0.882 ± 0.011	0.800 ± 0.007	0.784 ± 0.006
NoFed-Self-Finetune	0.789 ± 0.012	0.792 ± 0.014	0.757 ± 0.012	0.845 ± 0.026	0.758 ± 0.018	0.735 ± 0.015
Fed-Sup (FedAvg)	0.769 ± 0.003	0.754 ± 0.009	0.741 ± 0.004	0.807 ± 0.019	0.722 ± 0.010	0.704 ± 0.013
FedSGL-Freeze (ours)	**0.849 ± 0.005**	**0.868 ± 0.009**	0.776 ± 0.005	**0.897 ± 0.011**	**0.818 ± 0.010**	**0.807 ± 0.005**
FedSGL-Finetune (ours)	0.786 ± 0.009	0.791 ± 0.012	0.753 ± 0.005	0.849 ± 0.019	0.764 ± 0.016	0.738 ± 0.015

thereby justifying the SSL objective. Secondly, No-Fed-Self-Finetune is worse than No-Fed-Self-Freeze, indicating that fine-tuning is not very effective for Amazon networks which is contrary to what we observed in Twitch. We attribute this again to limited train labels and fine-tuning is still ineffective. Thirdly, we interestingly find that Fed-Sup suffers very badly across all clients and we witness avg. gain of -4.16% compared to No-Fed-Sup across clients. Fig 6.5(b) further shows it individually. This is again different to what we observed in twitch where the same comparison (Fed-Sup vs No-Fed-Sup) did not lead to this kind of severe performance degradation. We reason that because of the increased downstream task heterogeneity in Amazon compared to Twitch, a global model learnt using label supervision alone will be incapable to perform well. Moreover, The non-IID ness among Amazon clients is significantly greater than among Twitch clients and standard supervised FedAvg (Fed-Sup) falls short. Fourthly, our method **FedSGL-Freeze** provides significant increase in performance compared to baselines across all clients due to the effective and collaborative use of unlabeled data. Specifically, we witness avg. 11.5% gains across clients over Fed-Sup which shows how our formulation can be effectively used when there is high task heterogeneity across clients. Fig 6.5(c) further shows the gains individually. We also witness an avg. gain of 1.9% over NoFed-Self-Freeze and further shows that our formulation can indeed leverage unlabeled data across clients using federation to improve downstream performance locally.

6.5.3 OGB Cross-Silo Molecule Datasets

Table 6.6 shows the results for graph classification task on the OBG cross-silo molecule dataset. We use FedAvg [233] and personalized federated methods such as FedProx [131]

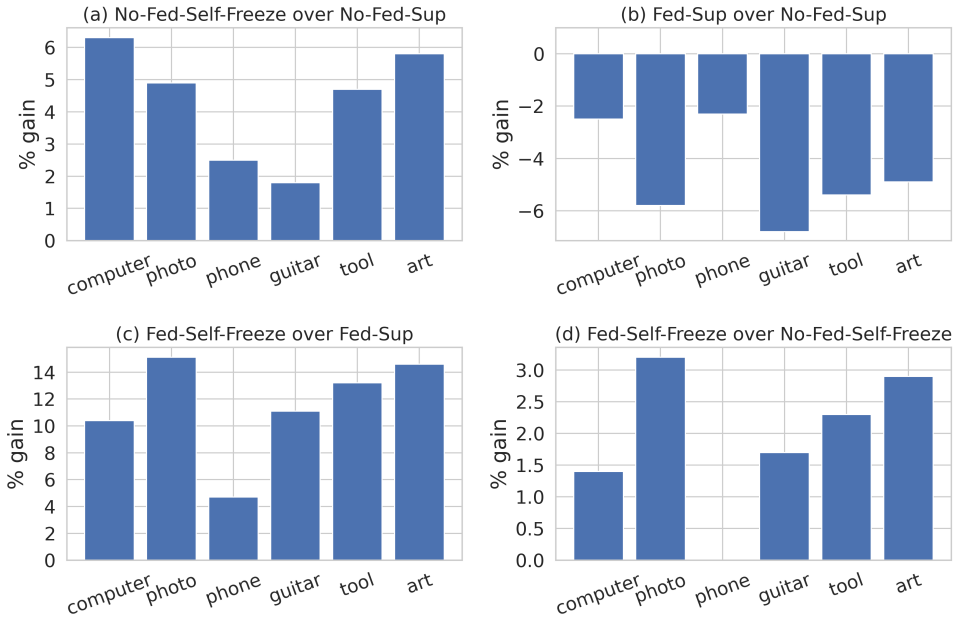

Figure 6.5. Performance gains in Amazon network experiments

which uses proximal regularization technique and GCFL+ [123] which utilizes client clustering. For the case with train/val/test split of 80/10/10 which has sufficient train label availability, it is clear that supervised FL (i.e. FedAvg) performs worse than NoFed-Sup with avg. gains of -8.6%. The best personalized method (among FedProx and GCFL+) also performs poorly compared to NoFed-Sup with avg. gains of -8.3%. Next, client specific self-supervision and further task fine-tuning i.e., NoFed-Self-Finetune is on par or better than client specific supervised model without self-supervision i.e., NoFed-Sup with an avg. gain of $+4.1\%$. This shows the usefulness of self-supervision. Our method FedSGL-Finetune results in best performance with avg. gain of $+5.4\%$ over NoFed-Self-Finetune which means using the features of other clients is always beneficial even if clients hold data for very different prediction tasks. Also, we witness an avg. gain of $+19.3\%$ over the best supervised FL baseline (among FedAvg, FedProx and GCFL+) which shows how our method can tackle the heterogeneous label bias problem in supervised FL methods. In the label scarcity case where we use a train/val/test split of 8/10/10, it is evident that FedSGL-Finetune results in avg. gain of $+4.1\%$ over NoFed-Self-Finetune and an avg. gain of $+11.6\%$ over the best

Table 6.6. Graph classification task across cross-silo clients. Avg. ROC-AUC Scores over 10 splits and std. dev.

Split		bbbp (1)	bace (1)	tox21 (12)	toxcast (617)	sider (27)	clintox (2)	muv (17)	hiv (1)
80/10/10	NoFed-Sup	67.66 ± 2.42	71.00 ± 3.59	74.11 ± 0.53	62.25 ± 0.56	57.30 ± 2.44	60.95 ± 3.08	71.04 ± 2.57	75.25 ± 1.40
	FedAvg	57.98 ± 3.41	58.18 ± 6.34	67.83 ± 1.72	56.84 ± 1.23	52.21 ± 1.99	60.28 ± 6.16	69.84 ± 4.14	69.61 ± 2.85
	FedProx	55.49 ± 3.20	59.38 ± 6.21	66.87 ± 1.74	56.40 ± 1.68	52.67 ± 2.53	57.11 ± 8.64	71.81 ± 4.21	69.45 ± 3.94
	GCFL+	55.23 ± 4.60	60.13 ± 8.12	67.72 ± 1.10	56.57 ± 1.57	53.51 ± 3.49	58.45 ± 5.17	69.86 ± 5.11	69.77 ± 2.21
	NoFed-Self-Freeze	64.85 ± 1.05	57.27 ± 0.36	68.98 ± 0.24	59.37 ± 0.37	58.75 ± 0.93	50.76 ± 1.39	71.64 ± 1.48	67.31 ± 0.62
	NoFed-Self-Finetune	71.05 ± 1.14	73.01 ± 1.76	74.82 ± 0.31	62.91 ± 0.24	59.76 ± 0.49	68.53 ± 1.37	75.92 ± 1.90	76.18 ± 0.81
	FedSGL-Freeze	61.03 ± 1.97	56.89 ± 1.03	69.64 ± 0.57	58.86 ± 0.48	60.45 ± 0.63	60.79 ± 1.98	73.35 ± 1.59	68.20 ± 0.97
	FedSGL-Finetune	72.40 ± 1.35	80.47 ± 1.61	75.23 ± 0.94	63.72 ± 0.83	61.53 ± 1.02	84.30 ± 1.87	76.79 ± 1.81	77.91 ± 1.03
8/10/10	NoFed-Sup	57.80 ± 2.19	60.59 ± 4.23	67.74 ± 1.14	57.09 ± 0.64	49.97 ± 1.84	48.85 ± 5.53	63.23 ± 2.13	69.27 ± 3.22
	FedAvg	57.66 ± 4.40	51.55 ± 12.11	65.45 ± 0.92	53.31 ± 1.15	51.22 ± 1.69	54.03 ± 4.76	61.27 ± 3.31	65.27 ± 3.31
	FedProx	54.81 ± 3.84	56.50 ± 8.64	65.22 ± 1.52	55.12 ± 2.03	50.41 ± 1.56	49.34 ± 5.40	62.08 ± 4.26	68.97 ± 3.08
	GCFL+	57.81 ± 4.35	57.61 ± 8.83	65.04 ± 1.60	52.81 ± 1.30	50.49 ± 2.25	52.48 ± 5.77	60.38 ± 2.95	67.34 ± 2.93
	NoFed-Self-Freeze	61.54 ± 0.52	53.46 ± 1.20	65.07 ± 0.61	55.62 ± 0.23	51.08 ± 0.75	42.57 ± 2.38	64.65 ± 1.08	64.12 ± 0.51
	NoFed-Self-Finetune	61.42 ± 1.75	72.03 ± 1.98	68.07 ± 0.82	57.71 ± 0.41	51.56 ± 0.73	56.70 ± 1.91	65.95 ± 2.33	72.65 ± 0.92
	FedSGL-Freeze	58.09 ± 0.76	56.35 ± 1.76	67.41 ± 0.98	55.46 ± 0.56	53.51 ± 0.65	53.00 ± 1.63	67.18 ± 0.72	66.22 ± 1.13
	FedSGL-Finetune	63.38 ± 1.09	79.55 ± 1.07	69.07 ± 0.34	58.81 ± 0.17	54.89 ± 0.55	58.52 ± 1.08	68.81 ± 0.23	73.89 ± 0.87

supervised FL (among FedAvg, FedProx, GCFL+) method. This again shows that in the low label availability regime coupled with heterogeneous task scenario, our method FedSGL is able to extract common features across clients and provides a uniform advantage over local non-federated self-supervised methods and federated supervised methods.

6.6 Discussion

Motivated by two key characteristics of real-world graph data when used in cross-silo settings: (1) Label deficiency and (2) Downstream task heterogeneity, we proposed a novel problem setting that has not been considered by current FedGRL setups. We raise a natural research question of how one can leverage vast amounts of unlabeled graph data and collaboratively learn a shared model using federated learning that is effective at solving downstream client tasks. We provide a general formulation based on model interpolation where a shared global model is both self-supervised and gets supervision with available local task labels through a local client model. We provided two learning protocols based on this general formulation and provided specific instantiations using BGRL for the self-supervised objective. To empirically verify one of our learning protocol and its instantiation, we adopted a real world Twitch Gamer Networks to simulate a cross-geo FedGRL application and we witnessed

on avg. 6.1% gains over traditional supervised federated objectives. To further incorporate the task heterogeneity characteristic, we constructed a new cross-silo dataset called Amazon Co-purchase Networks which have varying downstream label domains. We showed how standard supervised federated objectives can result in negative gains (on avg. -4.16%) compared to individual client specific training, due to the increased data heterogeneity and finally justified our formulation which results in on avg. 11.5% gains over traditional supervised federated learning and on avg. 1.9% gains over individually trained self-supervised models. We witness up to avg. +19.3% gains in OBG cross-silo molecule dataset for graph classification which shows how FedSGL outperforms personalized FL methods even when they have task supervision. Additionally, FedSGL is able to extract shared features across clients and provides gains over local non-federated SSL under label scarcity.

Several extensions to our work can be considered. While we have experimented with learning protocol **LP1** in this chapter, more experiments are required to fully realize the potential benefits of the formulations, especially **LP2**. It is also significantly more challenging as the shared global model may be biased towards local client labels due to supervision and techniques from personalized federated learning literature may have to be adapted, instead of the standard FedAvg algorithm we currently employ. Further work can also be undertaken to explore the best server aggregation protocol, experiments with varying client participation ratios and how it affects self-supervised based federation can also be further explored. Moreover, our methodology of constructing the amazon co-purchase networks can be extended to include more clients for future bench-marking. We stress on the fact that for FedGRL, realistic datasets are a major gap in current works and more effort is required in this front.

7. SUMMARY AND DISCUSSIONS

In this chapter, we summarize the contributions of the two thrusts presented in this dissertation and discuss areas of future work.

7.1 Contributions

In Chapter 3, motivated by the observation that nodes in real-world networks exhibit diverse mixing behaviour, we conducted an in depth investigation into the relation between node mixing and SoTA GNN performance in node label prediction tasks. We first proposed the use of a local assortativity metric which offers a fine grained measure of node level mixing compared to a conventional global measurement of network assortativity (e.g., assortativity coefficient). Using real-world networks, we empirically demonstrated that the predictive performance of GNNs are highly correlated with our local assortativity measure. We attribute this limiting behaviour to the over reliance of proximity information in constructing node representations—in part due to the smoothing effects of message passing in GNNs. We empirically identify that for nodes with high local assortativity, proximity information from its surroundings suffices to infer its own label whereas, for nodes with low local assortativity, structural similarity is key to help distinguish or infer their labels. We then proposed a graph transformation algorithm that transforms the original graph into a computation graph accounting for both node level structural and proximity information. The resulting computation graph is shown to experimentally improve assortativity owing to the use of structural regularities in the input graph. We further developed a practical version of the transformation algorithm that reduces the quadratic complexity of structural and proximal similarity calculation to log-linear in the the number of nodes in the graph. With extensive experiments on various real world networks from different domains we showed that our novel GNN (WRGAT) which adaptively chooses structure and proximity information in the transformed computation graph provides superior node classification performance compared to baseline GNN methods that operate on the original graph.

In Chapter 4, we proposed an expressive and efficient approach to learn link representations in temporal graphs for the task of ranking links. First, we developed a novel temporal

graph encoder model that incorporates a *Labeling Trick* followed by parametrized diffusion using GNNs to learn structural and temporal link representations. We theoretically showed the benefits of increased expressivity due to the use of our labeling trick in the encoder compared to previous models. We also theoretically showed that our encoder is capable of inductive link ranking due to the use of (1) our parametrized diffusion model with permutation equivariance property and (2) our timestamp encoding scheme which is time shift invariant. Instead of optimizing the encoder parameters with a conventional point-wise loss as employed by previous methods, we proposed to use a list-wise ranking loss function which better suits the goals of the evaluation i.e. ranking links. To implement the list-wise loss in practice, we proposed a subgraph sampling technique which selects an effective set of candidates for ranking. Our labeling approach and diffusion over the sampled subgraph is amenable to joint inference over candidates and we performed a detailed complexity analysis to show the benefits in terms of scalable inference over candidates of our approach in comparison to baseline methods. We conducted a detailed empirical study using real-world temporal network datasets to demonstrate increased ranking performance of our approach in both trasductive and inductive learning settings. To validate our theoretical claims of increased expressivity, we conducted ablations with and without the labeling trick and empirically demonstrated performance gains in link ranking tasks. Lastly, we provided detailed run time analysis to empirically demonstrate the practical efficiency of our approach in comparison to baselines.

In Chapter 5, we developed a principled learning mechanism to train graph neural networks in a self-supervised fashion. Our initial empirical study found that existing InfoMax priciple based methods often risk capturing redundant graph features which may be irrelevant to the downstream tasks. We proposed an adversarial InfoMax framework (*AD-GCL*), inspired by the graph information bottleneck theory with a goal of letting the graph encoder capture minimal information that is sufficient to identify each graph. We provided theoretical guarantees on the upper bound of irrelevant information captured by our framework and a lower bound of mutual information between learnt representations and downstream task labels. Inspired by our theoretical characterization of AD-GCL, we developed a practical instantiation via learnable edge dropping augmentations and a min-max contrastive objective.

To parameterize the learnable augmentation model, we developed a method that utilizes the Gumbel-Max reparametrization trick. Inspired by our theory, we further proposed a novel regularization term that controls the amount of perturbation in the min-max objective. Using real-world graph datasets from different domains, we performed extensive experiments in unsupervised learning setting to demonstrate the significant gains of our learning mechanism compared to baselines. We performed an in dept empirical study to analyze our min-max objective and demonstrated stable training dynamics. Lastly, we empirically tested our principle in semi-supervised and transfer learning settings to show improved performance gains compared to baselines.

In Chapter 6, motivated by two key characteristics of real-world graph data in cross-silo settings: (1) downstream task heterogeneity (2) label deficiency, we proposed a novel problem setting that has not been considered by current federated graph representation learning setups. We provided a general formulation based on model interpolation where a shared global model is both self-supervised and gets supervision with available local task labels through a local client model. We provided a learning framework (called FedSGL) based on this general formulation and provided specific instantiations using two self-supervised objective, BGRL and GraphCL for node and graph representation learning respectively. To empirically verify FedSGL and its instantiations, we adopted six real-world Twitch Gamer Networks to simulate a cross-geo application and we witnessed uniform gains over traditional supervised federated objectives. To further incorporate the task heterogeneity characteristic, we constructed a new cross-silo dataset called Amazon Co-purchase Networks which have varying downstream label domains. For graph level tasks, we adopted eight OGB molecule benchmark datasets to simulate a cross-silo application scenario. Overall, we showed how standard supervised federated objectives can result in negative gains compared to individual client specific training, due to the increased data heterogeneity and finally justified our formulation which results in significant gains over traditional supervised federated learning and uniform gains over locally trained self-supervised models.

The contributions of this dissertations are categorized into the following aspects. Chapter numbers are provided in parenthesis at the end of each bullet for ease of reference.

7.1.1 Theoretical

- Defining the notion of *GNN Troublesome Center Node* to show the node ambiguity problem and failure of GNN encoders to provide structural and temporal link representations for temporal graphs. Analytically quantifying the number of such troublesome center nodes in temporal graphs which are problematic for conventional GNNs. (Chapter 4)

- Analyzing the expressive power of our TGRank encoder in terms of distinguishing non-isomorphic temporal links. Proving that conventional GNN based encoders will fail to distinguish a lot of *GNN Troublesome Center Nodes* while, TGRank with labelling strategy can distinguish them—leading to enhanced expressivity. (Chapter 4)

- Proving the inductive ranking ability of TGRank which guarantees that for any two isomorphic temporal queries, their ranking prediction scores obtained from TGRank will be identical. (Chapter 4)

- Complexity analysis to show that as the size of the candidate set grows, other baseline methods will be much less efficient than our TGRank, which amortizes computation over a joint set rather than compute representations independently for each pair (source+one candidate) during link ranking. (Chapter 4)

- Defining graph data augmentation families and presenting mutual information bounds for AD-GCL principle. Upper bound of irrelevant information captured by our framework and a lower bound of mutual information between learnt representations and downstream task labels. (Chapter 5)

7.1.2 Algorithmic / Models

- Development of a graph transformation algorithm that transforms graphs which exhibit diverse mixing patterns into a highly assortative computation graph for running message passing GNNs. Our computation graph construction contains weighted *structural edges* that brings far away disassortative nodes closer and reflects different levels

of structural similarity between nodes. Simultaneously, models assortative node mixing behavior with the inclusion of weighted *proximity edges* that reflect different levels of proximal similarity between nodes. (Chapter 3)

- Efficient and practical computation graph construction to reduce the costly $\mathcal{O}(n^2)$ structural and proximity similarity calculation to $\mathcal{O}(n \log n)$ w.r.t the number of nodes n in graph. (Chapter 3)
- Development of a parametrized relational GNN model (WRGAT) that can adaptively choose between proximity and structural edges in our computation graph to boost node classification performance in the entire spectrum of local assortativity. (Chapter 3)
- Development of TGRank encoder model for learning expressive temporal link representations. (Chapter 4)
- Development of a subgraph sampling technique to generate effective candidate sets and efficiently train the TGRank encoder with a list-wise ranking loss. (Chapter 4)
- Development of a practical algorithmic instantiation of our AD-GCL principle to train GNNs with self-supervision using a novel min-max contrastive learning objective. (Chapter 5)
- For the instantiation of AD-GCL, development of a parametrized learnable edge dropping model using Gumbel-Max reparametrization trick and a novel regularization term to control the amount of perturbation. (Chapter 5)
- Development of FedSGL learning framework which trains GNNs in a federated fashion with self-supervised objectives. (Chapter 6)
- Concrete algorithmic instantiations of FedSGL with different combinations of (a) non-personalized/personalized FL algorithms with (b) commonly used self-supervised objectives, paired with (c) different local fine-tuning strategies. (Chapter 6)

7.1.3 Empirical / Applications

- Proposing the use of a local assortativity metric to measure diverse mixing behaviour in real-world networks and an in dept study of its effects on message passing GNN performance for the task of node label prediction. (Chapter 3)
- Experimental performance evaluation of our WRGAT encoder model against seven baseline methods, detailed ablations and parameter sensitivity analysis of our graph transformation algorithm for the task of semi-supervised node classification on six hyperlinked web-page networks, three citation networks, three air-traffic networks and a BGP internet network—all of which are real-world networks with diverse node mixing patterns. (Chapter 3)
- Experimental performance evaluation of TGRank and comparisons against relevant baselines for the task of temporal link ranking using six real-world temporal graphs. Extending prior methods to utilize list-wise loss for apt performance comparisons. (Chapter 4)
- Empirical evaluation of TGRank in both transductive and inductive learning settings, ablation study to probe into the critical components of TGRank, parameter sensitivity analysis and empirical run time analysis. (Chapter 4)
- Empirical analysis highlighting the drawbacks of traditional graph contrastive learning methods and showing that task irrelevant information can be sufficient for holding the InfoMax principle. (Chapter 5)
- Experimental evaluation of our instantiation of proposed AD-GCL principle and comparisons against baselines methods using eighteen different graph datasets from social and biochemistry domains for the task of unsupervised graph property prediction. (Chapter 5)
- Empirical analysis of regularizing the AD-GCL principle and showing stable training dynamics of our proposed min-max contrastive objective.

- Empirical performance evaluation of graph representations obtained from GNNs trained using AD-GCL principle and comparisons to baselines using six different datasets for semi-supervised graph property prediction. (Chapter 5)
- Transfer learning performance evaluation of graph representations obtained from GNNs trained using AD-GCL principle and other baselines on two large scale pre-train graph datasets and downstream fine-tuning on nine datasets for graph property prediction tasks. (Chapter 5)
- Construction and introduction of a new dataset of co-purchase networks based on different sales departments in Amazon, to test FedSGL over node classification tasks. (Chapter 6)
- Empirical study of proposed FedSGL framework. Experiments on a variety of real-world cross silo-ed graph datasets for node and graph classification tasks. Comparisons with local self-supervised baselines, supervised personalized and non-personalized federated methods. (Chapter 6)

7.2 Future Work

There are a number of interesting directions for future work that extend the ideas and techniques presented in this dissertation. We will outline immediate areas of exploration for each chapter and discuss broader avenues of future work.

In Chapter 3, we empirically observed correlation between node level local assortativity and GNN performance for node classification. Some recent works have begun to theoretically study the connection between a global network metric called *homophily* [64] to GNN representation quality [254], [255]. A possible extension is to provide theoretical arguments relating our fine grained node level assortativity and GNN node representation quality. While our generic toolkit can be used to empirically analyze future GNN encoders from the perspective of node mixing behaviour, it can also aid in further theoretical understanding of the node feature smoothing operation of message passing GNNs [56], [57]. From the technical encoder design standpoint, while our adaptive structure and proximity edge selection is

learnable using an attention mechanism, the underlying graph transformation algorithm is predefined and not learnable. Future work can build upon this to make the transformation process completely automatic and possibly fused with the encoder taking local assortativity into account. Our empirical study also motivates the curation of real-world networks with diverse mixing patterns to systematically benchmark GNN performance.

In Chapter 4, we proposed to incorporate a list-wise encoder training objective for the task of temporal link ranking. For efficiency reasons, we proposed a sampling algorithm to generate candidate sets for ranking training. Future work can be directed towards providing theoretical arguments to the effectiveness of the proposed candidate set generation process. Empirically, we witnessed increased gains in link ranking performance when our list-wise objective was used instead of the conventional point-wise objective. Immediate future work can be directed towards incorporating other learning to rank objectives and systematically studying their trade-offs in terms of efficiency and performance.

In Chapter 5, we showed the drawbacks of InfoMax principle based contrastive learning where redundant and task irrelevant information may be captured in GNN graph representations. We then instantiated a learnable edge dropping augmentation process within contrastive learning based on our proposed AD-GCL principle. We showed that when the augmentation process is optimized adversarially, GNN representations may learn the minimum and sufficient information for the downstream task. Immediate future work can look at incorporating other learnable augmentation families like node dropping, subgraph masking or attriburte masking. A broader goal would be to incorporate the AD-GCL principle to non-contrastive self-supervised methods such as BYOL [251] and momentum contrast [256]. There is growing body of work to understand the effectiveness of self-supervised learning mostly for vision and language domains [257]–[265]. Extending them to the graph domain to further understand the benefits of AD-GCL can be considered as future work. As an extension, the learnable edge dropping augmentation can also be used to provide explanations as to which edges are most useful when learning representations.

In Chapter 6, we developed a federated self-supervised graph learning framework (FedSGL) for both cross-silo node and graph level representation learning. An extension for future work is to consider cross-silo link representation learning. Further, theoretical convergence analysis

of FedSGL can be explored as future work. Another direction is a theoretical characterization of how FedSGL tackles heterogeneous label bias which supervised federated learning suffers.

REFERENCES

[1] M. Newman, *Networks*. Oxford university press, 2018.

[2] A.-L. Barabási *et al.*, *Network Science*. Cambridge University Press, 2016.

[3] *Cisco annual internet report*, https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report/index.html, Online; accessed 18 April 2022.

[4] J.-F. Rual, K. Venkatesan, T. Hao, *et al.*, "Towards a proteome-scale map of the human protein–protein interaction network," *Nature*, vol. 437, no. 7062, pp. 1173–1178, 2005.

[5] T. Rolland, M. Taşan, B. Charloteaux, *et al.*, "A proteome-scale map of the human interactome network," *Cell*, vol. 159, no. 5, pp. 1212–1226, 2014.

[6] L. K. Scheffer, C. S. Xu, M. Januszewski, *et al.*, "A connectome and analysis of the adult drosophila central brain," *Elife*, vol. 9, e57443, 2020.

[7] *CERN Open Data*, http://opendata.cern.ch/.

[8] A. J. Larkoski, I. Moult, and B. Nachman, "Jet substructure at the large hadron collider: A review of recent advances in theory and machine learning," *Physics Reports*, vol. 841, pp. 1–63, 2020.

[9] J. Shlomi, P. Battaglia, and J.-R. Vlimant, "Graph neural networks in particle physics," *Machine Learning: Science and Technology*, vol. 2, no. 2, p. 021 001, 2020.

[10] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 974–983.

[11] X. L. Dong, X. He, A. Kan, *et al.*, "Autoknow: Self-driving knowledge collection for products of thousands of types," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 2724–2734.

[12] S. Wu, F. Sun, W. Zhang, and B. Cui, "Graph neural networks in recommender systems: A survey," *arXiv preprint arXiv:2011.02260*, 2020.

[13] T. Gaudelet, B. Day, A. R. Jamasb, *et al.*, "Utilizing graph machine learning within drug discovery and development," *Briefings in bioinformatics*, vol. 22, no. 6, bbab159, 2021.

[14] A. W. Senior, R. Evans, J. Jumper, *et al.*, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.

[15] M. Zitnik, M. Agrawal, and J. Leskovec, "Modeling polypharmacy side effects with graph convolutional networks," *Bioinformatics*, vol. 34, no. 13, pp. i457–i466, 2018.

[16] V. Fung, J. Zhang, E. Juarez, and B. G. Sumpter, "Benchmarking graph neural networks for materials chemistry," *npj Computational Materials*, vol. 7, no. 1, pp. 1–8, 2021.

[17] O. Wieder, S. Kohlbacher, M. Kuenemann, *et al.*, "A compact review of molecular property prediction with graph neural networks," *Drug Discovery Today: Technologies*, vol. 37, pp. 1–12, 2020.

[18] J. M. Stokes, K. Yang, K. Swanson, *et al.*, "A deep learning approach to antibiotic discovery," *Cell*, vol. 180, no. 4, pp. 688–702, 2020.

[19] C. L. Zitnick, L. Chanussot, A. Das, *et al.*, "An introduction to electrocatalyst design using machine learning for renewable energy storage," *arXiv preprint arXiv:2010.09435*, 2020.

[20] B. Gallagher and T. Eliassi-Rad, "Leveraging label-independent features for classification in sparsely labeled networks: An empirical study," in *International Workshop on Social Network Mining and Analysis*, Springer, 2008, pp. 1–19.

[21] L. A. Adamic and E. Adar, "Friends and neighbors on the web," *Social networks*, vol. 25, no. 3, pp. 211–230, 2003.

[22] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[23] K. Henderson, B. Gallagher, L. Li, *et al.*, "It's who you know: Graph mining using recursive structural features," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 663–671.

[24] N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield, "Efficient graphlet counting for large networks," in *2015 IEEE International Conference on Data Mining*, IEEE, 2015, pp. 1–10.

[25] L. Backstrom and J. Leskovec, "Supervised random walks: Predicting and recommending links in social networks," in *Proceedings of the fourth ACM international conference on Web search and data mining*, 2011, pp. 635–644.

[26] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Applied Network Science*, vol. 5, no. 1, pp. 1–42, 2020.

[27] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[30] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[31] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[32] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[33] A. van den Oord, S. Dieleman, H. Zen, *et al.*, "Wavenet: A generative model for raw audio," in *Arxiv*, 2016. [Online]. Available: https://arxiv.org/abs/1609.03499.

[34] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[35] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *ICML*, JMLR. org, 2017.

[36] J. Jumper, R. Evans, A. Pritzel, *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[37] A. Ramesh, M. Pavlov, G. Goh, *et al.*, "Zero-shot text-to-image generation," in *International Conference on Machine Learning*, PMLR, 2021, pp. 8821–8831.

[38] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[39] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artifical Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

[40] L. Wu, P. Cui, J. Pei, and L. Zhao, *Graph Neural Networks: Foundations, Frontiers, and Applications*. Singapore: Springer Singapore, 2022, p. 725.

[41] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE international joint conference on neural networks*, vol. 2, 2005, pp. 729–734.

[42] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[43] K. Hornik, M. Stinchcombe, H. White, *et al.*, "Multilayer feedforward networks are universal approximators.," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[44] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[45] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[46] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[47] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *International Conference on Learning Representations*, 2018.

[48] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, *et al.*, "Convolutional networks on graphs for learning molecular fingerprints," in *NeurIPS*, vol. 2015, 2015, pp. 2224–2232.

[49] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," *ICLR*, 2018.

[50] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Transactions On Graphics (tog)*, 2019.

[51] B. Weisfeiler and A. Leman, "A reduction of a graph to a canonical form and an algebra arising during this reduction," *Nauchno-Technicheskaya Informatsia*, 1968.

[52] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *ICLR*, 2018.

[53] C. Morris, M. Ritzert, M. Fey, *et al.*, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4602–4609.

[54] R. Sato, "A survey on the expressive power of graph neural networks," *arXiv preprint arXiv:2003.04078*, 2020.

[55] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *International Conference on Learning Representations*, 2014.

[56] H. NT and T. Maehara, "Revisiting graph neural networks: All we have is low-pass filters," *arXiv preprint arXiv:1905.09550*, 2019.

[57] G. Fu, Y. Hou, J. Zhang, K. Ma, B. F. Kamhoua, and J. Cheng, "Understanding graph neural networks from graph signal denoising perspectives," *arXiv preprint arXiv:2006.04386*, 2020.

[58] Y. Min, F. Wenkel, and G. Wolf, "Scattering gcn: Overcoming oversmoothness in graph convolutional networks," *arXiv preprint arXiv:2003.08414*, 2020.

[59] X. Dong, D. Thanou, L. Toni, M. Bronstein, and P. Frossard, "Graph signal processing for machine learning: A review and new perspectives," *arXiv preprint arXiv:2007.16061*, 2020.

[60] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," *arXiv preprint arXiv:1801.07606*, 2018.

[61] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997*, 2018.

[62] K. Oono and T. Suzuki, "Graph neural networks exponentially lose expressive power for node classification," in *International Conference on Learning Representations*, 2020.

[63] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 3438–3445.

[64] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annual review of sociology*, 2001.

[65] M. E. Newman, "Mixing patterns in networks," *Physical review E*, vol. 67, no. 2, p. 026 126, 2003.

[66] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *KDD*, 2019, pp. 1269–1278.

[67] Z. Fan, Z. Liu, J. Zhang, Y. Xiong, L. Zheng, and P. S. Yu, "Continuous-time sequential recommendation with temporal graph collaborative transformer," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 433–442.

[68] Y. Wang, Y.-Y. Chang, Y. Liu, J. Leskovec, and P. Li, "Inductive representation learning in temporal networks via causal anonymous walks," in *ICLR*, 2020.

[69] Y. Wang, P. Li, C. Bai, and J. Leskovec, "Tedic: Neural modeling of behavioral patterns in dynamic social interaction networks," in *WWW*, 2021, pp. 693–705.

[70] A. Pareja, G. Domeniconi, J. Chen, *et al.*, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," in *AAAI*, vol. 34, 2020, pp. 5363–5370.

[71] A. Z. Wang, R. Ying, P. Li, N. Rao, K. Subbian, and J. Leskovec, "Bipartite dynamic representations for abuse detection," in *KDD*, 2021, pp. 3638–3648.

[72] P. Holme and J. Saramäki, "Temporal networks," *Physics reports*, vol. 519, no. 3, pp. 97–125, 2012.

[73] P. W. Battaglia, J. B. Hamrick, V. Bapst, *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[74] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," in *ICLR*, 2020.

[75] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *arXiv preprint arXiv:2006.10637*, 2020.

[76] B. Srinivasan and B. Ribeiro, "On the equivalence between node embeddings and structural graph representations," in *International Conference on Learning Representations*, 2020.

[77] P. Li, Y. Wang, H. Wang, and J. Leskovec, "Distance encoding: Design provably more powerful neural networks for graph representation learning," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[78] M. Zhang, P. Li, Y. Xia, K. Wang, and L. Jin, "Labeling trick: A theory of using graph neural networks for multi-node representation learning," *NeurIPS*, vol. 34, 2021.

[79] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *ICLR*, 2019.

[80] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *International Conference on Machine Learning*, PMLR, 2016, pp. 2702–2711.

[81] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[82] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *the AAAI Conference on Artificial Intelligence*, 2018, pp. 4438–4445.

[83] W. Hu, B. Liu, J. Gomes, *et al.*, "Strategies for pre-training graph neural networks," in *International Conference on Learning Representations*, 2019.

[84] F.-Y. Sun, J. Hoffmann, V. Verma, and J. Tang, "Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization," *arXiv preprint arXiv:1908.01000*, 2019.

[85] H. G. Vogel, *Drug discovery and evaluation: pharmacological assays*. Springer Science & Business Media, 2002.

[86] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[87] S. Zhang, Z. Hu, A. Subramonian, and Y. Sun, "Motif-driven contrastive learning of graph representations," *arXiv preprint arXiv:2012.12533*, 2020.

[88] S. Thakoor, C. Tallec, M. G. Azar, R. Munos, P. Veličković, and M. Valko, "Bootstrapped representation learning on graphs," in *ICLR 2021 Workshop on Geometrical and Topological Representation Learning*, 2021.

[89] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Graph contrastive learning with adaptive augmentation," in *Proceedings of the Web Conference 2021*, 2021, pp. 2069–2080.

[90] J. Qiu, Q. Chen, Y. Dong, *et al.*, "Gcc: Graph contrastive coding for graph neural network pre-training," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1150–1160.

[91] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," in *International Conference on Machine Learning*, PMLR, 2019, pp. 2434–2444.

[92] Y. Hu and Y. Zhang, "Graph contrastive learning with local and global mutual information maximization," in *2020 The 8th International Conference on Information Technology: IoT and Smart City*, 2020, pp. 74–78.

[93] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," *arXiv preprint arXiv:1809.10341*, 2018.

[94] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5812–5823, 2020.

[95] K. Hassani and A. H. Khasahmadi, "Contrastive multi-view representation learning on graphs," in *International Conference on Machine Learning*, PMLR, 2020, pp. 4116–4126.

[96] Y. Xie, Z. Xu, J. Zhang, Z. Wang, and S. Ji, "Self-supervised learning of graph neural networks: A unified review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[97] Y. Liu, M. Jin, S. Pan, *et al.*, "Graph self-supervised learning: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[98] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[99] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014, pp. 701–710.

[100] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

[101] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "Struc2vec: Learning node representations from structural identity," in *KDD*, 2017, pp. 385–394.

[102] K. Henderson, B. Gallagher, T. Eliassi-Rad, *et al.*, "Rolx: Structural role extraction & mining in large graphs," in *KDD*, 2012, pp. 1231–1239.

[103] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec, "Learning structural node embeddings via diffusion wavelets," in *KDD*, 2018, pp. 1320–1329.

[104] R. Linsker, "Self-organization in a perceptual network," *Computer*, vol. 21, no. 3, pp. 105–117, 1988.

[105] M. Tschannen, J. Djolonga, P. K. Rubenstein, S. Gelly, and M. Lucic, "On mutual information maximization for representation learning," in *International Conference on Learning Representations*, 2020.

[106] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," *arXiv preprint physics/0004057*, 2000.

[107] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle," in *2015 IEEE Information Theory Workshop (ITW)*, IEEE, 2015.

[108] Z. Goldfeld and Y. Polyanskiy, "The information bottleneck problem and its applications in machine learning," *IEEE Journal on Selected Areas in Information Theory*, 2020.

[109] A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy, "Deep variational information bottleneck," *arXiv preprint arXiv:1612.00410*, 2016.

[110] X. B. Peng, A. Kanazawa, S. Toyer, P. Abbeel, and S. Levine, "Variational discriminator bottleneck: Improving imitation learning, inverse rl, and gans by constraining information flow," *arXiv preprint arXiv:1810.00821*, 2018.

[111] I. Higgins, L. Matthey, A. Pal, *et al.*, "Beta-vae: Learning basic visual concepts with a constrained variational framework.," in *International Conference on Learning Representations*, 2017.

[112] T. Wu, H. Ren, P. Li, and J. Leskovec, "Graph information bottleneck," in *Advances in Neural Information Processing Systems*, 2020.

[113] J. Yu, T. Xu, Y. Rong, Y. Bian, J. Huang, and R. He, "Recognizing predictive substructures with subgraph information bottleneck," *International Conference on Learning Representations*, 2021.

[114] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[115] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[116] K. Bonawitz, P. Kairouz, B. McMahan, and D. Ramage, "Federated learning and privacy: Building privacy-preserving systems for machine learning and data science on decentralized data," *Queue*, vol. 19, no. 5, pp. 87–114, 2021.

[117] C. He, K. Balasubramanian, E. Ceyani, *et al.*, "Fedgraphnn: A federated learning system and benchmark for graph neural networks," *arXiv preprint arXiv:2104.07145*, 2021.

[118] X. Fu, B. Zhang, Y. Dong, C. Chen, and J. Li, "Federated graph machine learning: A survey of concepts, techniques, and applications," *arXiv e-prints*, arXiv–2207, 2022.

[119] R. Liu and H. Yu, "Federated graph neural networks: Overview, techniques and challenges," *arXiv preprint arXiv:2202.07256*, 2022.

[120] Z. Liu, L. Yang, Z. Fan, H. Peng, and P. S. Yu, "Federated social recommendation with graph neural network," *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2021.

[121] C. Wu, F. Wu, L. Lyu, T. Qi, Y. Huang, and X. Xie, "A federated graph neural network framework for privacy-preserving personalization," *Nature Communications*, vol. 13, no. 1, pp. 1–10, 2022.

[122] D. Manu, Y. Sheng, J. Yang, *et al.*, "Fl-disco: Federated generative adversarial network for graph-based molecule drug discovery: Special session paper," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, IEEE, 2021, pp. 1–7.

[123] H. Xie, J. Ma, L. Xiong, and C. Yang, "Federated graph classification over non-iid graphs," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18 839–18 852, 2021.

[124] W. Zhu, A. White, and J. Luo, "Federated learning of molecular properties in a heterogeneous setting," *arXiv preprint arXiv:2109.07258*, 2021.

[125] T. Suzumura, Y. Zhou, N. Baracaldo, *et al.*, "Towards federated graph learning for collaborative financial crimes detection," *NeurIPS Workshops*, 2019.

[126] H. C. Bayram and I. Rekik, "A federated multigraph integration approach for connectional brain template learning," in *International Workshop on Multimodal Learning for Clinical Decision Support*, Springer, 2021, pp. 36–47.

[127] L. Peng, N. Wang, N. Dvornek, X. Zhu, and X. Li, "Fedni: Federated graph learning with network inpainting for population-based disease prediction," *IEEE Transactions on Medical Imaging*, 2022.

[128] M. Zitnik, R. Sosič, and J. Leskovec, "Prioritizing network communities," *Nature communications*, vol. 9, no. 1, pp. 1–9, 2018.

[129] Y. Jiang, J. Konečnỳ, K. Rush, and S. Kannan, "Improving federated learning personalization via model agnostic meta learning," *arXiv preprint arXiv:1909.12488*, 2019.

[130] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning: A meta-learning approach," *arXiv preprint arXiv:2002.07948*, 2020.

[131] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[132] C. T Dinh, N. Tran, and T. D. Nguyen, "Personalized federated learning with moreau envelopes," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[133] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 4427–4437.

[134] F. Hanzely and P. Richtárik, "Federated learning of a mixture of global and local models," *arXiv preprint arXiv:2002.05516*, 2020.

[135] F. Sattler, K.-R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 8, pp. 3710–3722, 2020.

[136] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 586–19 597, 2020.

[137] G. Cheng, K. Chadha, and J. Duchi, "Fine-tuning is fine in federated learning," *arXiv preprint arXiv:2108.07313*, 2021.

[138] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai, "Fedavg with fine tuning: Local updates lead to representation learning," *arXiv preprint arXiv:2205.13692*, 2022.

[139] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, "On the convergence of federated optimization in heterogeneous networks," *arXiv preprint arXiv:1812.06127*, vol. 3, p. 3, 2018.

[140] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE TKDE*, 2020.

[141] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, "Machine learning on graphs: A model and comprehensive taxonomy," *Journal of Machine Learning Research*, vol. 23, no. 89, pp. 1–64, 2022.

[142] L. Babai, "Groups, graphs, algorithms: The graph isomorphism problem," in *Proc. ICM*, World Scientific, vol. 3, 2018, pp. 3303–3320.

[143] H. A. Helfgott, J. Bajpai, and D. Dona, "Graph isomorphisms in quasi-polynomial time," *arXiv preprint arXiv:1710.04574*, 2017.

[144] L. Peel, J.-C. Delvenne, and R. Lambiotte, "Multiscale mixing patterns in networks," *PNAS*, 2018.

[145] G. T. Cantwell and M. Newman, "Mixing patterns and individual differences in networks," *Physical Review E*, 2019.

[146] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," *ICLR*, 2020.

[147] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, "Beyond homophily in graph neural networks: Current limitations and effective designs," *NeurIPS*, 2020.

[148] M. Liu, Z. Wang, and S. Ji, "Non-local graph neural networks," *arXiv preprint arXiv:2005.14612*, 2020.

[149] Y. Hou, J. Zhang, J. Cheng, *et al.*, "Measuring and improving the use of graph information in graph neural networks," in *ICLR*, 2019.

[150] E. Chien, J. Peng, P. Li, and O. Milenkovic, "Adaptive universal generalized pagerank graph neural network," *arXiv preprint arXiv:2006.07988*, 2020.

[151] P. Li, E. Chien, and O. Milenkovic, "Optimizing generalized pagerank methods for seed-expansion community detection," *Advances in Neural Information Processing Systems*, vol. 32, pp. 11 705–11 716, 2019.

[152] L. Chen, Z. Chen, and J. Bruna, "On graph neural networks versus graph-augmented mlps," *arXiv preprint arXiv:2010.15116*, 2020.

[153] B. Y. Weisfeiler and A. Leman, "Reduction of a graph to a canonical form and an algebra arising during this reduction (in russian)," *Nauchno-Technicheskaya Informatsia*, 1968.

[154] J. Klicpera, S. Weißenberger, and S. Günnemann, "Diffusion improves graph learning," in *NeurIPS*, 2019, pp. 13 333–13 345.

[155] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," *International Conference on Machine Learning*, pp. 5453–5462, 2018.

[156] S. Abu-El-Haija, B. Perozzi, A. Kapoor, *et al.*, "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," *ICML*, pp. 21–29, 2019.

[157] P. Boldi, "Totalrank: Ranking without damping," in *Special interest tracks and posters of the 14th international conference on World Wide Web*, ACM, 2005, pp. 898–899.

[158] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intell. Data Anal.*, 2007.

[159] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*, 2018.

[160] G. Namata, B. London, L. Getoor, B. Huang, and U. EDU, "Query-driven active surveying for collective classification," in *10th International Workshop on Mining and Learning with Graphs*, 2012.

[161] M. Luckie, B. Huffaker, A. Dhamdhere, V. Giotsas, and K. Claffy, "As relationships, customer cones, and validation," in *Proceedings of the 2013 conference on Internet measurement conference*, 2013.

[162] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-Scale Attributed Node Embedding," *Journal of Complex Networks*, vol. 9, no. 2, 2021.

[163] J. Tang, J. Sun, C. Wang, and Z. Yang, "Social influence analysis in large-scale networks," in *KDD*, 2009.

[164] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, p. 93, 2008.

[165] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NeurIPS*, 2016.

[166] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[167] X. Wang, D. Lyu, M. Li, *et al.*, "Apan: Asynchronous propagation attention network for real-time temporal graph embedding," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2628–2638.

[168] C. J. Burges, "From ranknet to lambdarank to lambdamart: An overview," *Learning*, vol. 11, no. 23-581, p. 81, 2010.

[169] D. Yin, Y. Hu, J. Tang, *et al.*, "Ranking relevance in yahoo search," in *KDD*, 2016, pp. 323–332.

[170] F. Manessi, A. Rozza, and M. Manzo, "Dynamic graph convolutional networks," *Pattern Recognition*, vol. 97, p. 107 000, 2020.

[171] P. Goyal, S. R. Chhetri, and A. Canedo, "Dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowledge-Based Systems*, vol. 187, p. 104 816, 2020.

[172] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dysat: Deep neural representation learning on dynamic graphs via self-attention networks," in *WSDM*, 2020, pp. 519–527.

[173] J. You, T. Du, and J. Leskovec, "Roland: Graph learning framework for dynamic graphs," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 2358–2366.

[174] S. M. Kazemi, R. Goel, K. Jain, *et al.*, "Representation learning for dynamic graphs: A survey.," *JMLR*, vol. 21, no. 70, pp. 1–73, 2020.

[175] J. Skardinga, B. Gabrys, and K. Musial, "Foundations and modelling of dynamic networks using dynamic graph neural networks: A survey," *IEEE Access*, 2021.

[176] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *WWW*, 2018, pp. 969–976.

[177] N. Bastas, T. Semertzidis, A. Axenopoulos, and P. Daras, "Evolve2vec: Learning network representations using temporal unfolding," in *International Conference on Multimedia Modeling*, Springer, 2019, pp. 447–458.

[178] Y. Ma, Z. Guo, Z. Ren, J. Tang, and D. Yin, "Streaming graph neural networks," in *SIGIR*, 2020, pp. 719–728.

[179] R. Trivedi, H. Dai, Y. Wang, and L. Song, "Know-evolve: Deep temporal reasoning for dynamic knowledge graphs," in *ICML*, PMLR, 2017, pp. 3462–3471.

[180] S. Mahdavi, S. Khoshraftar, and A. An, "Dynnode2vec: Scalable dynamic network embedding," in *IEEE Big Data*, IEEE, 2018, pp. 3762–3765.

[181] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic network embedding by modeling triadic closure process," in *AAAI*, vol. 32, 2018.

[182] L. Du, Y. Wang, G. Song, Z. Lu, and J. Wang, "Dynamic network embedding: An extended approach for skip-gram based network embedding.," in *IJCAI*, vol. 2018, 2018, pp. 2086–2092.

[183] W. Song, Z. Xiao, Y. Wang, L. Charlin, M. Zhang, and J. Tang, "Session-based social recommendation via dynamic graph attention networks," in *WSDM*, 2019, pp. 555–563.

[184] M. Zhang, S. Wu, X. Yu, Q. Liu, and L. Wang, "Dynamic graph neural networks for sequential recommendation," *arXiv preprint arXiv:2104.07368*, 2021.

[185] J. Chang, C. Gao, Y. Zheng, *et al.*, "Sequential recommendation with graph neural networks," in *SIGIR*, 2021, pp. 378–387.

[186] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *ICDM*, IEEE, 2018, pp. 197–206.

[187] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," Stanford InfoLab, Tech. Rep., 1999.

[188] G. Jeh and J. Widom, "Scaling personalized web search," in *the International Conference on World Wide Web*, Acm, 2003, pp. 271–279.

[189] L. Lovász *et al.*, "Random walks on graphs: A survey," *Combinatorics, Paul erdos is eighty*, vol. 2, no. 1, pp. 1–46, 1993.

[190] S. Mangan and U. Alon, "Structure and function of the feed-forward loop network motif," *PNAS*, vol. 100, no. 21, pp. 11 980–11 985, 2003.

[191] J. You, J. Gomes-Selman, R. Ying, and J. Leskovec, "Identity-aware graph neural networks," in *AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 10 737–10 745.

[192] Z. Zhu, Z. Zhang, L.-P. Xhonneux, and J. Tang, "Neural bellman-ford networks: A general graph neural network framework for link prediction," *arXiv preprint arXiv:2106.06935*, 2021.

[193] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Advances in Neural Information Processing Systems*, vol. 31, 2018, pp. 5165–5175.

[194] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *NeurIPS*, vol. 20, 2007.

[195] Y. Liu, J. Ma, and P. Li, "Neural predicting higher-order patterns in temporal networks," in *TheWebConf (WWW)*, 2022.

[196] S. Bochner, "A theorem on Fourier-Stieltjes integrals," *Collected Papers of Salomon Bochner*, vol. 2, 1992.

[197] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Self-attention with functional time representation learning," *NeurIPS*, vol. 32, pp. 15 915–15 925, 2019.

[198] S. M. Kazemi, R. Goel, S. Eghbali, *et al.*, "Time2vec: Learning a vector representation of time," *arXiv preprint arXiv:1907.05321*, 2019.

[199] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: From pairwise approach to listwise approach," in *ICML*, 2007, pp. 129–136.

[200] J. Kunegis, "Konect: The koblenz network collection," in *WWW*, 2013, pp. 1343–1350.

[201] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Engineering Bulletin*, vol. 40, no. 3, pp. 52–74, 2017.

[202] T. M. Cover, *Elements of information theory*. John Wiley & Sons, 1999.

[203] W. Hu, M. Fey, M. Zitnik, *et al.*, "Open graph benchmark: Datasets for machine learning on graphs," *arXiv preprint arXiv:2005.00687*, 2020.

[204] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" In *International Conference on Learning Representations*, 2019.

[205] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[206] E. N. Gilbert, "Random graphs," *The Annals of Mathematical Statistics*, vol. 30, no. 4, pp. 1141–1144, 1959.

[207] P. Erdős and A. Rényi, "On random graphs i.," *Publ. Math. Debrecen*, vol. 6, pp. 290–297, 1959.

[208] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *International Conference on Learning Representations*, 2017.

[209] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *International Conference on Learning Representations*, 2017.

[210] D. Luo, W. Cheng, D. Xu, *et al.*, "Parameterized explainer for graph neural network," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[211] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[212] Y. Tian, D. Krishnan, and P. Isola, "Contrastive multiview coding," *arXiv preprint arXiv:1906.05849*, 2019.

[213] B. Poole, S. Ozair, A. Van Den Oord, A. Alemi, and G. Tucker, "On variational bounds of mutual information," in *International Conference on Machine Learning*, 2019.

[214] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, PMLR, 2020, pp. 1597–1607.

[215] S. Becker and G. E. Hinton, "Self-organizing neural network that discovers surfaces in random-dot stereograms," *Nature*, vol. 355, no. 6356, pp. 161–163, 1992.

[216] O. Henaff, "Data-efficient image recognition with contrastive predictive coding," in *International Conference on Machine Learning*, PMLR, 2020, pp. 4182–4192.

[217] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, *et al.*, "Learning deep representations by mutual information estimation and maximization," in *International Conference on Learning Representations*, 2019.

[218] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. Hinton, "Big self-supervised models are strong semi-supervised learners," *arXiv preprint arXiv:2006.10029*, 2020.

[219] Y. Jiao, Y. Xiong, J. Zhang, Y. Zhang, T. Zhang, and Y. Zhu, "Sub-graph contrast for scalable self-supervised graph representation learning," in *2020 IEEE international conference on data mining (ICDM)*, IEEE, 2020, pp. 222–231.

[220] Y. You, T. Chen, Y. Shen, and Z. Wang, "Graph contrastive learning automated," *arXiv preprint arXiv:2106.07594*, 2021.

[221] Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola, "What makes for good views for contrastive learning?" In *Advances in Neural Information Processing Systems*, 2020.

[222] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.

[223] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tu-dataset: A collection of benchmark datasets for learning with graphs," in *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.

[224] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 23, no. 4, pp. 550–560, 1997.

[225] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[226] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2015, pp. 1365–1374.

[227] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[228] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "Graph2vec: Learning distributed representations of graphs," *arXiv preprint arXiv:1707.05005*, 2017.

[229] B. Adhikari, Y. Zhang, N. Ramakrishnan, and B. A. Prakash, "Sub2vec: Feature learning for subgraphs," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, 2018, pp. 170–182.

[230] S. Suresh, P. Li, C. Hao, and J. Neville, "Adversarial graph augmentation to improve graph contrastive learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 920–15 933, 2021.

[231] *General data protection regulation*, https://gdpr-info.eu/, [Online; accessed 17-August-2022], Sep. 2019.

[232] B. Custers, A. M. Sears, F. Dechesne, I. Georgieva, T. Tani, and S. Van der Hof, *EU personal data protection in policy and practice*. Springer, 2019.

[233] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, PMLR, 2017, pp. 1273–1282.

[234] B. Wang, A. Li, H. Li, and Y. Chen, "Graphfl: A federated learning framework for semi-supervised node classification on graphs," *arXiv preprint arXiv:2012.04187*, 2020.

[235] K. Zhang, C. Yang, X. Li, L. Sun, and S. M. Yiu, "Subgraph federated learning with missing neighbor generation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 6671–6682, 2021.

[236] L. Zheng, J. Zhou, C. Chen, B. Wu, L. Wang, and B. Zhang, "Asfgnn: Automated separated-federated graph neural network," *Peer-to-Peer Networking and Applications*, vol. 14, no. 3, pp. 1692–1704, 2021.

[237] F. Chen, P. Li, T. Miyazaki, and C. Wu, "Fedgraph: Federated graph learning with intelligent sampling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 8, pp. 1775–1786, 2021.

[238] H. Peng, H. Li, Y. Song, V. Zheng, and J. Li, "Differentially private federated knowledge graphs embedding," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 1416–1425.

[239] J. Zhou, C. Chen, L. Zheng, *et al.*, "Vertically federated graph neural network for privacy-preserving node classification," *arXiv preprint arXiv:2005.11903*, 2020.

[240] X. Ni, X. Xu, L. Lyu, C. Meng, and W. Wang, "A vertical federated learning framework for graph convolutional network," *arXiv preprint arXiv:2106.11593*, 2021.

[241] Z. Peng, W. Huang, M. Luo, *et al.*, "Graph representation learning via graphical mutual information maximization," in *Proceedings of The Web Conference 2020*, 2020, pp. 259–270.

[242] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Deep graph contrastive representation learning," *arXiv preprint arXiv:2006.04131*, 2020.

[243] X. Chen and K. He, "Exploring simple siamese representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 15 750–15 758.

[244] Z. T. Kefato and S. Girdzijauskas, "Self-supervised graph neural networks without explicit negative sampling," *arXiv preprint arXiv:2103.14958*, 2021.

[245] F. Che, G. Yang, D. Zhang, J. Tao, and T. Liu, "Self-supervised graph representation learning via bootstrapping," *Neurocomputing*, vol. 456, pp. 88–96, 2021.

[246] F. Zhang, K. Kuang, Z. You, *et al.*, "Federated unsupervised representation learning," *arXiv preprint arXiv:2010.08982*, 2020.

[247] W. Zhuang, X. Gan, Y. Wen, S. Zhang, and S. Yi, "Collaborative unsupervised visual representation learning from decentralized data," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 4912–4921.

[248] W. Zhuang, Y. Wen, and S. Zhang, "Divergence-aware federated self-supervised learning," *ICLR*, 2022.

[249] C. He, Z. Yang, E. Mushtaq, S. Lee, M. Soltanolkotabi, and S. Avestimehr, "Ssfl: Tackling label deficiency in federated learning via personalized self-supervision," *arXiv preprint arXiv:2110.02470*, 2021.

[250] E. S. Lubana, C. I. Tang, F. Kawsar, R. P. Dick, and A. Mathur, "Orchestra: Unsupervised federated learning via globally consistent clustering," *arXiv preprint arXiv:2205.11506*, 2022.

[251] J.-B. Grill, F. Strub, F. Altché, *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning," *Advances in neural information processing systems*, vol. 33, pp. 21 271–21 284, 2020.

[252] T. Li, S. Hu, A. Beirami, and V. Smith, "Ditto: Fair and robust federated learning through personalization," in *International Conference on Machine Learning*, PMLR, 2021, pp. 6357–6368.

[253] J. Ni, J. Li, and J. McAuley, "Justifying recommendations using distantly-labeled reviews and fine-grained aspects," in *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, 2019, pp. 188–197.

[254] X. Zheng, Y. Liu, S. Pan, M. Zhang, D. Jin, and P. S. Yu, "Graph neural networks for graphs with heterophily: A survey," *arXiv preprint arXiv:2202.07082*, 2022.

[255] Y. Ma, X. Liu, N. Shah, and J. Tang, "Is homophily a necessity for graph neural networks?" *arXiv preprint arXiv:2106.06134*, 2021.

[256] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9729–9738.

[257] J. Z. HaoChen, C. Wei, A. Gaidon, and T. Ma, "Provable guarantees for self-supervised deep learning with spectral contrastive loss," *Advances in Neural Information Processing Systems*, vol. 34, pp. 5000–5011, 2021.

[258] J. Z. HaoChen, C. Wei, A. Kumar, and T. Ma, "Beyond separability: Analyzing the linear transferability of contrastive representations to related subpopulations," *arXiv preprint arXiv:2204.02683*, 2022.

[259] J. Z. HaoChen and T. Ma, "A theoretical study of inductive biases in contrastive learning," *arXiv preprint arXiv:2211.14699*, 2022.

[260] A. Kumar, A. Raghunathan, R. Jones, T. Ma, and P. Liang, "Fine-tuning can distort pretrained features and underperform out-of-distribution," *arXiv preprint arXiv:2202.10054*, 2022.

[261] C. Wei, K. Shen, Y. Chen, and T. Ma, "Theoretical analysis of self-training with deep networks on unlabeled data," *arXiv preprint arXiv:2010.03622*, 2020.

[262] Q. Garrido, Y. Chen, A. Bardes, L. Najman, and Y. Lecun, "On the duality between contrastive and non-contrastive self-supervised learning," *arXiv preprint arXiv:2206.02574*, 2022.

[263] N. Saunshi, O. Plevrakis, S. Arora, M. Khodak, and H. Khandeparkar, "A theoretical analysis of contrastive unsupervised representation learning," in *International Conference on Machine Learning*, PMLR, 2019, pp. 5628–5637.

[264] X. Liu, F. Zhang, Z. Hou, *et al.*, "Self-supervised learning: Generative or contrastive," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[265] K. Shen, R. M. Jones, A. Kumar, *et al.*, "Connect, not collapse: Explaining contrastive learning for unsupervised domain adaptation," in *International Conference on Machine Learning*, PMLR, 2022, pp. 19 847–19 878.

VITA

Susheel Suresh received his Bachelor of Engineering in computer science from P.E.S Institute of Technology, Bangalore, India in 2017. After a brief stint as a member of technical staff at Adobe Systems, Bangalore, he moved to the computer science department at Purdue University to pursue graduate studies during the Fall of 2018. He worked under Professor Jennifer Neville on building models and representation learning mechanisms for graph structured data. He has published several works at various machine learning, data mining and artificial intelligence conferences and/or workshops viz. NeurIPS, KDD, ICDM, CIKM and AAAI. He also has several research internship experiences both in industry/national labs viz. Indian Institute of Technology Bombay, INRIA Paris-Saclay, Microsoft Corporation, Redmond and Microsoft Research, Redmond.