GRAPH REPRESENTATION LEARNING FOR UNSUPERVISED AND SEMI-SUPERVISED LEARNING TASKS

by

Mengyue Hang

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 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Jennifer Neville, Chair

Department of Computer Science and Statistics

Dr. Bruno Ribeiro

Department of Computer Science

Dr. Yexiang Xue

Department of Computer Science

Dr. Ming Yin

Department of Computer Science

Approved by:

Dr. Kihong Park

ACKNOWLEDGMENTS

First I would like to express my deepest gratitude to my advisor, Jennifer Neville, for being the ideal advisor and a great role model. Her enthusiasm, her inspiration, and her endless support helped make this daunting task an enjoyable experience. During my time at Purdue, Jan has imparted innumerable lessons, from practical instructions on research, writing, and presentation, to insightful guidance on academia and industry – which transformed me from a fresh graduate student to an independent researcher and an experienced ML practitioner. I would not be where I am today without her.

I would also like to thank Bruno Ribeiro and Yexiang Xue for being on my PhD committee, Alex Pothen and Ming Yin for being external members on my preliminary and final exams respectively. Exposure to their supplementary views helped me broden and deepen my sense of research. Moreover, I greatly appreciate the guidance and assistance that Bruno has offered, particularly in the topics of representation learning and GNNs.

I thank Carolina Barcenas, Bill Campbell, Jack Kearney, Paul Bennett, Tobias Schnabel, Longqi Yang, Haoming Chen, Chongxi Bao, for hosting my internships at Visa, Amazon, Microsoft and Facebook. Throughout these internship experiences, I developed an appreciation for applied machine learning research, with a focus on developing efficient, explainable approaches to solving real-world problems with massive datasets.

I would like to thank all my lab mates: Jean Lin, Jason Meng, Jiasen Yang, Ellen Lai, Hogun Park, Gui Gomes, Mahak Goindani, Giselle Zeno, and all the others – for always providing their invaluable input and comments in practice talks, lab meetings, and extended research discussions.

Next, I would like to express my appreciation to my friends, who have helped me maintain my sanity throughout all the challenges in graduate school. To Tamalika Mukherjee, Linjie Li and Yanjun Wang, who have offered me their continuous love and support along the way – even at great distances. Thank you my friends Zitao Li, Yue Xing, Zixun Yu, Ruying Bao, Ran Xin. This journey wouldn't have been the same without you.

To my parents, I cannot thank them enough for providing me with unfailing support and continuous encouragement throughout my years of study. And to my boyfriend, for being my strength, my joy, and my happiness.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Graph representation learning and Graph Neural Network (GNNs) models provide flexible tools for modeling and representing relational data (graphs) in various application domains. Specifically, node embedding methods provide continuous representations for vertices that has proved to be quite useful for prediction tasks, and Graph Neural Networks (GNNs) have recently been used for semi-supervised node and graph classification tasks with great success.

However, most node embedding methods for unsupervised tasks consider a simple, sparse graph, and are mostly optimized to encode aspects of the network structure (typically local connectivity) with random walks. And GNNs model dependencies among the attributes of nearby neighboring nodes rather than dependencies among observed node labels, which makes it not expressive enough for semi-supervised node classification tasks.

This thesis will investigate methods to address these limitations, including:

(1) For heterogeneous graphs: Development of a method for dense(r), heterogeneous graphs that incorporates global statistics into the negative sampling procedure with applications in recommendation tasks; (2) For capturing long-range role equivalence: Formalized notions of representation-based equivalence w.r.t regular/automorphic equivalence in a single graph or multiple graph samples, which is employed in a embedding-based models to capture long-range equivalence patterns that reflect topological roles; (3) For collective classification: Since GNNs model dependencies among the attributes of nearby neighboring nodes rather than dependencies among observed node labels, we develop an add-on collective learning framework to GNNs that provably boosts their expressiveness for node classification tasks, beyond that of an *optimal* WL-GNN, utilizing self-supervised learning and Monte Carlo sampled embeddings to incorporate node labels during inductive learning for semi-supervised node classification tasks.

1. INTRODUCTION

Machine learning on graphs is an important and ubiquitous task with a variety of applications ranging from social networks, recommendation systems, knowledge graphs, and drug discovery. The central problem in this domain is finding a way to represent, or encode high-dimensional graph structure into low-dimensional feature vectors so that it can be easily exploited by machine learning models for downstream tasks. Among the graph representation learning models, graph embedding and Graph Neural Networks are two powerful types of models for tackling mostly unsupervised and semi-supervised tasks respectively, and there has a surge of approaches that seek to develop more powerful representation learning methods under various settings.

In this thesis we consider a graph representation learning problem, where the objective is to learn node representations for a given graph that encodes the structural information, node attributes and partial labels if available. Specifically, if given a graph G with vertex set V ($N = |V|$) and adjacency matrix $\boldsymbol{A}$, the task is to learn a node representation $\boldsymbol{Z}_v$ for each node $v \in V$, which are then evaluated with graph tasks such as node classification, i.e. predict $P(\boldsymbol{Y}|\boldsymbol{Z})$ where $\boldsymbol{Y}$ is node labels. For example, in a citation network, nodes can be publications and edges are citations between them, then the task is to learn a vector representation for each publication and use the representation to predict its research area. Depending on the problem setting, the input graph G might have different specifications, and in this thesis we consider (1) graphs with multiple node types, (2) graphs with labels $\boldsymbol{Y}$ reflecting topological roles, (3) multiple graphs with varying connectivity structure, (4) graphs with partial labels and attributes $\boldsymbol{X}$.

We argue that previous graph representation learning methods fail to consider the characteristics of these specific settings which are common in real-world applications. For example, spatio-temporal social networks are weighted, heterogeneous graphs, and the node activity level in a transportation network is independent of its position in the graph, but closely related to its topological role; and Graph Neural Networks applied to partially labeled graphs doesn't consider joint label distribution ($P(\boldsymbol{Y}|G)$) with independent inference for node classification problem.

There are several deficiencies when applying these methods to certain graphs with more complicated signals.

- **Biased negative sampling due to heterogeneity of graph structure** Instead of computing an expensive $|V||E|$ objective, most node embedding models adopt a negative sampling approach to make the optimization tractable, which might be biased when there is large variations of graph structures. In such cases, negative sampling allows a node to choose its true neighbor as a negative sample, especially when high-degree nodes are present in graphs with increased density, e.g. user-building interaction graphs with density of 20% in some spatio-temporal social networks.

- **Failure to capture long-range structural similarity** Much of the work in graph embedding based on random walks has been designed to preserve node proximity (e.g. [1], [2], [3]) explicitly or implicitly. This works for detecting communities especially in networks with high degree of homophily. However, it fails to identify nodes with topologically similar network neighborhoods that reside in potentially distant parts of the network. For example, in a protein-protein interaction network, two proteins with the same functionality might not interact with other or have common neighbors, but may be embedded in similar subgraphs.

- **Incapability of modeling multiple graphs with varying connectivity** Previous graph representation learning methods mostly consider a single graph, however, in reality we are often given a set of graphs with varying connectivity. In this case, each single graph might be a noisy indication of regular equivalence, but the set of graph structures encodes the equivalence notion. For example, given a set of temporal graphs, two nodes might not be exactly equivalent at each timestamp, but their behaviors across all timestamps are regularly equivalent. Existing methods for sets of graphs mainly consider the dynamics, thus fail to capture the behaviors across the graphs. Therefore, the notion of regular equivalence, and regular equivalence in graph embedding, should be extended to sets of graphs.

- **Ignoring label dependence in partially-labeled graphs** When tackling semi-supervised node classification problems, existing GNNs perform independent inference, which means it considers a conditional independence assumption (i.e. factoring $P(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{A})$ as $\prod_{v \in V} P(\boldsymbol{Y}_v|\boldsymbol{X}, \boldsymbol{A})$), while collective classification methods used in Statistical Relational Learning (SRL) [4] are designed to model the joint distribution of $P(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{A})$. Collective classification can be unnecessary with most expressive node representation, thus the conditional independence assumption of GNNs wouldn't matter. But as shown in previous works ([5]–[8]), GNNs are not most expressive. Therefore, the joint distribution of $P(\boldsymbol{Y}|\boldsymbol{X})$ contains more signals than what is captured by the GNN.

1.1 Problem Statement

This thesis aims to understand the unique challenges of applying node embedding and graph neural networks to unlabeled and partially-labeled graphs with complex signals, and use this understanding to (1) motivate the development of new models that are more suitable or more powerful than existing methods to capture more signals in graphs, and (2) improve the model performance for various down-stream prediction tasks under both unsupervised and semi-supervised settings. Therefore, we consider the following three central research questions and the extended sub-questions.

1. *How to customize node embedding models to take into account characteristics of real-world graphs for link prediction tasks, specifically weighted heterogeneous graphs?* When given a dense(r) weighted graph with heterogeneous node types, the negative sampling objective of existing node embedding models (for homogeneous and heterogeneous graphs) fails by allowing the model to choose neighboring nodes as negative samples, which also causes over-shrinkage of embeddings of high degree nodes. How to adapt node embedding models to capture these more complex signals in graphs and how to evaluate the efficacy of these unsupervised models via downstream tasks are all important questions to consider.

2. *How to capture long-range structural equivalence patterns, in single graphs and sets of graphs?* The notions of structural and regular equivalence are often used to reason about

the roles and relationships among nodes. However, these formulations typically focus on discrete roles which limits its usage for real-world applications. Node embeddings provide a continuous representation that has recently proved to be quite useful for prediction tasks, but most of the current node embedding models fail to consider high-order graph structure. How to learn node embeddings that capture the information of node roles remains an open question. In addition, it is natural to extend these concepts to reflect the sets of graph structures and their relations, and to consider node embedding methods that are able to encode these information.

3. *How and to which extent can collective learning/inference improve the expressive power of Graph Neural Networks?* Given collective inference (CI) has been extensively used in the past to boost weak relational models (e.g. relational Naive Bayes [9], [10], relational logistic regression [11], it is natural to consider and verify whether collective learning or collective inference could also boost the expressive power of GNNs. Much of the efforts on GNNs have been focusing on designing new architectures to capture more complicated graph structures, whereas few attention has been paid to incorporate label dependence by CI, and the usage of label predictions in GNN structures needs careful design.

1.2 Main Hypotheses and Proposed Research

Throughout this thesis, we examine these hypotheses: (1) Node embedding methods can be enhanced to consider complex signals in dense, heterogeneous graphs (2) Collective learning and inference can boost the expressivity of Graph Neural Networks for semi-supervised node classification; and (3) Node representations can be learned from a single graph or a set of graphs to detect topological roles by considering the extended notion of regular or automorphic equivalence.

In the course we break the investigation into the following three parts based on the stream of research questions highlighted in the previous section.

1. A node embedding model that work for heterogeneous spatio-temporal social networks for link prediction tasks;

2. Formalized notions of representation-based equivalence w.r.t regular/automphic equivalence on a single graph or multiple graph samples, and a flexible node embedding framework to capture them.

3. A collective learning framework for Graph Neural Networks to boost expressive power for semi-supervised classification tasks;

1.3 Thesis Organization

The rest of this document is organized as follows.

- In Chapter 2, We introduce the background of important concepts about graph representation learning methodologies.

- Chapters 3, 4 and 5 build the main contributions of this dissertation. In Chapter 3, we present a node embedding model which works for dealing with dense(r), heterogeneous graphs with application in POI recommendation; In Chapter 4, we introduce *Reqe*, a node embedding framework for capturing topological equivalence patterns in single graphs and multiple graphs; In Chapter 5, we introduce a collective learning framework for Graph Neural Networks to boost expressive power for semi-supervised node classification tasks.

- Finally, Chapter 6 concludes with a summary of our contributions and outlines the future directions.

2. BACKGROUND

This thesis is built upon ideas from several research areas. In this chapter, we survey several basic concepts and/or models in these areas, which will be applied, analyzed and/or extended in the rest of this thesis.

2.1 Graph representation learning approaches

2.1.1 Unsupervised Node embeddings

Unsupervised graph embedding methods seek to learn representations that encode the graph structure. These embeddings have demonstrated outstanding performance on a number of tasks including node classification, knowledge-base completion, semi-supervised learning, and link prediction. In general, these methods (incl. [3], [2], [12], [13], [14]) operate in two discrete steps: First, they sample pair-wise relationships from the graph through random walks and counting node co-occurances. Second, they train an embedding model e.g. using Skipgram of word2vec [14], to learn representations that encode pairwise node similarities. The Skip-gram architecture is a one hidden layer neural network originally introduced for the word embedding problem, which has been successfully extended to the network embedding problem by treating the sampled node sequence as word sequence. In the network context, skip-gram learns for each node u two d-dimensional vectors, an embedding vector Z_u and a context vector Z'_u, by maximizing the log probability of observed network neighborhoods of the nodes:

$$\max_{Z,Z'} \sum_{u \in V} \sum_{v \in \mathcal{N}_u} \log P(v|u; Z, Z')$$

where $\mathcal{N}_u$ is the neighborhood of u and V is the set of all nodes; The connectivity probability $P(v|u; Z, Z')$ is modeled by a softmax:

$$P(v|u; Z, Z') = \frac{\exp(Z_v'^T \cdot Z_u)}{\sum_{w \in V} \exp(Z_w'^T \cdot Z_u)}$$

Since this objective is computationally infeasible for large-scale networks because each of the softmax terms requires summation over all vertices, Negative sampling (NS) provides a computationally feasible approximation to the objective by replacing each $\log P(v|u; Z, Z')$ term with

$$\log(\sigma(Z_v'^T \cdot Z_u)) + \sum_{\mathrm{i}=1}^{k} \mathbb{E}_{w\sim q(v)} \log(\sigma(-Z_w'^T \cdot Z_u))$$

where $q(v)$ is the noise distribution, which by default is proportional to d_v^β where d_v is degree of node v, and degree power β is a hyper-parameter, conventionally set to 3/4 because of good empirical performance [14].

Heterogeneous graph embedding

A large number of real-world graphs or networks are inherently heterogeneous, involving a diversity of node types and relation types. Most existing heterogeneous graph embedding methods are based on the idea of metapaths (see e.g. [15]). A metapath is an ordered sequence of node types and edge types defined on the network schema, which describes a composite relation between the nodes types involved. We can view a metapath as high-order proximity between two nodes. Most of the current methods doesn't consider edge weights and treat all nodes equally.

2.1.2 Semi-supervised Graph Neural Networks

Recently, there is increasing interest in extending deep learning approaches for graph data. Motivated by CNNs, RNNs, and autoencoders from deep learning, new generalizations of important operations have been rapidly developed to handle the complexity of graph data. Like other graph embedding methods, Graph Neural Networks (GNNs) also generate node representation ($\boldsymbol{h}$ based on graph structure and node attributes. Modern GNNs follow a neighborhood aggregation strategy, where we iteratively update the representation of a node by aggregating representations of its neighbors. After k iterations, a node's representation $h_v^{(k)}$ captures the structural information within its k-hop network neighborhood, and self-

representation of the node is combined using a COMBINE function. Formally, the k-th layer of a GNN is defined as

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}(\{h_u^{(k-1)} : u \in \mathcal{N}(v)\}), h_v^{(k)} = \text{COMBINE}^{(k)}(h_v^{(k-1)}, a_v^{(k)})$$

where $h_v^{(0)}$ is set to node attribute of node v ($\boldsymbol{X}_v$), and $\mathcal{N}(v)$ is the neighbors of node v. The choice of AGGREGATE and COMBINE functions in GNNs is various in different methods. For example, GCN[16] uses element-wise mean-pooling for aggregating nearby information, and GraphSage [17] uses max-pooling and LSTM for updating the representation in its different variants.

2.2 Equivalence concepts for social networks

Equivalence has become a fundamental concept in social network representations of social structure.

Below we outline several definitions of node equivalence in a graph $G = (V, E)$ with nodes $v \in V$ and edges $\mathrm{e_{ij}} \in E$. Let $\mathcal{N}(\mathrm{i})$ be a function that returns the set of neighbors of node v_i (i.e., $\{v_\mathrm{j}\}_{\mathrm{e_{ij}} \in E}$).

The relationship between the below equivalences is: Strong structural equivalence $\implies$ Weak structural equivalence $\implies$ Automorphic equivalence $\implies$ Regular equivalence.

Definition 2.2.1. *Strong structural equivalence ([18] [19])*
Two vertices u and v are strongly structural equivalent if, and only if, they have the same neighborhoods, i.e.

$$s(u) = s(v) \iff \mathcal{N}(\mathrm{i}) = \mathcal{N}(\mathrm{j})$$

Definition 2.2.2. *Weak structural equivalence [20]*
Two vertices u and v are weakly structurally equivalent if, and only if, the permutation $\boldsymbol{\pi} = (u, v)$ is an automorphism. i.e.

$$u \equiv v \iff \exists \boldsymbol{\pi} \ s.t. \ \boldsymbol{\pi}(u) = v \wedge \boldsymbol{\pi}(v) = u \wedge \forall w \in [V - \{u, v\}] \ \boldsymbol{\pi}(w) = w$$

where $\equiv$ denotes weak structural equivalent

Definition 2.2.3. *Automorphic equivalence [21]*

Two vertices u and v are automorphically equivalent if and only if there exists an automorphism π such that $\pi(u) = v$. Automorphic equivalence is a natural generalization of weak structural equivalence.

Automorphic equivalence can be viewed as a relaxation of structural equivalence. More intuitively, structural equivalence essentially asks if a single node can be exchanged for another while preserving the connections/relationships of that node, whereas automorphic equivalence is based on sets of nodes whom are exchangeable as subgraphs [22].

Definition 2.2.4. *Regular equivalence [23]*

Equivalent actors have the same types of relations with equivalent others, but not necessarily the same others. Formally, it is defined recursively with respect to node roles $r(v)$

$$r(\mathrm{i}) = r(\mathrm{j}) \iff multiset\{r(\mathrm{i}')|\mathrm{i}' \in \mathcal{N}(\mathrm{i})\} = multiset\{r(\mathrm{j}')|\mathrm{j}' \in \mathcal{N}(\mathrm{j})\}$$

In recent years, there has been work on theoretical investigations of social networks and the accompanying equivalences, including axioms and logical characterizations [24] [25], spectral methods [26], and block models [27].

2.3 Collective learning and inference for relational data

Conventional relational machine learning (RML) developed methods to learn joint models from labeled graphs [10], [11]

$$P(\boldsymbol{Y}_U|\boldsymbol{X}, \boldsymbol{A})$$

To achieve this, many of the methods use *pseudolikelihood* estimation and consider a Markov assumption —every node $v_\mathrm{i} \in V$ is considered conditionally independent of the rest of the network given its Markov Blanket ($\mathcal{MB}(v_\mathrm{i})$). For undirected graphs, this is often simply the set of the immediate neighbors of v_i.

Given the Markov blanket assumption, RML methods typically use a local conditional model (e.g., relational Naive Bayes [28], relational logistic regression [29]) to learn and infer

labels within the network. The pseudolikelihood objective considers the nodes in a labeled subgraph G_L, where the labels of all neighbors are known:

$$O = \sum_{v \in V_L^{(\text{tr})}} \log P(y_v^{(\text{tr})} | Y_{\mathcal{MB}(v)}^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})}) \tag{2.1}$$

The *key* difference between eq. (2.1) and the GNNs objective in eq. (5.1): the RML model is conditioned on the labels $\boldsymbol{Y}$ even when there are no observed labels in the test data, i.e., even when $\boldsymbol{Y}_L^{(\text{te})} = \emptyset$. When the model is applied to make predictions in an unlabeled graph, *joint (i.e., collective) inference* methods such as variational inference or Gibbs sampling must be applied in order to use the conditionals from eq. (2.1). This combines the local conditional probabilities with global inference to estimate the joint distribution over the unlabeled vertices, e.g.,:

$$P(\boldsymbol{Y}_U | \boldsymbol{Y}_L, \boldsymbol{X}) \approx Q(\boldsymbol{Y}_U) = \Pi_{v_{\text{i}} \in V_U} Q_{\text{i}}(y)$$

where each component $Q_{\text{i}}(y)$ is iteratively updated.

Alternatively, a Gibbs sampler iteratively draws a label from the corresponding conditional distributions of the unlabeled vertices:

$$\hat{y}_v \sim P(y_v | \hat{Y}_{\mathcal{MB}(v)}, \boldsymbol{Y}_L, \boldsymbol{X}, A), \ \forall v \in V.$$

Note that for conventional RMLs, we assume a fully labeled (sub)network for learning, thus $Y_{\mathcal{MB}(v_{\text{i}})}$ only includes known labels, i.e., $v_{\text{j}} \in V_L$.

For transductive settings, where the goal is to learn and predict within a partially labeled graph, RML methods have considered semi-supervised formulations [4], [9], [30] to model the joint probability distribution:

$$P(\boldsymbol{Y}_U | \boldsymbol{Y}_L, \boldsymbol{X}, \boldsymbol{A})$$

In this case RML methods use both *collective learning* and collective inference procedures for semi-supervised learning. For example Expectation Maximization (EM) [9], [30], iterative

updates the parameter estimates by utilizing the expected values of the unlabeled examples to relearn the parameters.

For instance, the PL-EM algorithm [30] optimizes the pseudolikelihood

$$\textbf{E-Step: } \text{evaluate } P(\boldsymbol{Y}_U^{(\text{tr})} | \boldsymbol{Y}_L^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})}, \Theta_{t-1})$$

$$\textbf{M-Step: } \text{learn } \Theta_t :$$

$$\Theta_t = \arg\max_{\Theta} \sum_{\boldsymbol{Y}_U^{(\text{tr})} \in \Gamma_U} P(\boldsymbol{Y}_U^{(\text{tr})} | \boldsymbol{Y}_L^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})}, \Theta_{t-1}) \times \sum_{v \in V^{(\text{tr})}} \log P(y_v^{(\text{tr})} | \boldsymbol{Y}_{\mathcal{MB}(v)}^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})}, \Theta)$$

3. NODE EMBEDDINGS ON HETEROGENEOUS GRAPHS FOR POI RECOMMENDATION

In this chapter we present a node embedding model adapted to characteristics of a real-world dense(r), heterogeneous graph with applications in POI recommendation and friend suggestion.

3.1 Introduction

Millions of check-in records in *location-based social networks* (LBSNs) provide an opportunity to study users' mobility pattern and social behavior from a spatial-temporal perspective. In recent years, the *point-of-interest* (POI) recommendation/prediction problem has attracted significant attention [31], [32], [33], [34], [35], particularly for advertising and personalization. In POI tasks, the goal is to use user behavioral data to model users' activities at different locations and times, and then make predictions (or recommendations) for relevant venues based on their current context (including spatial, temporal, and other contextual information).

While POI predictions have broad applicability to myriad organizations, to date research has focused on developing POI methods based solely on voluntary check-in datasets collected from online social network apps such as Foursquare or Yelp [36], [37]. While these data contain rich information about recreational activities (e.g., restaurants, nightlife, and entertainment), the reliance on voluntary reporting results in sparse information about more prosaic aspects of daily life (e.g., offices, errands, houses). Moreover, recreation-based check-in data may bias conclusions drawn about mobility patterns or personal preferences. For example, Foursquare users often visit a POI only once, so the users' check-ins may not be sufficient to derive preferences for venues themselves, but only for venue *categories*. Also since check-ins to location-based social networks are often sporadic [36], it can be difficult to identify consistent user patterns.

In this work, we present the first analysis of a spatio-temporal educational "check-in" dataset, with the aim of using POI predictions to personalize student recommendations (e.g., clubs, friends, study locations) and to understand behavior patterns that increase student

retention and satisfaction. The results also provide a better idea of how campus facilities are utilized and how students connect with each other. The Purdue University "check-in" data records (anonymized) users' access to WiFi access points on campus, with venue information about locations (e.g., dining hall, library, dorm, gym). Specifically, we analyze WiFi access history across on-campus buildings, for all freshmen over one semester.

Compared to well-known check-in datasets like Foursquare, these data contain (1) more active users, (2) a richer set of daily activities (e.g., study, dine, exercise, rest), and (3) well-annotated spatial range (i.e., on campus). These characteristics make it easier to analyze the unique properties of user check-in data and extract interesting social and mobility patterns. Notably the WiFi access logs provide better temporal resolution than previous LBSN datasets, since a user "checks-in" whenever her device sends or receives a packet through a wireless connection. Similar data are collected by GPS trackers, where location observations are passively recorded [38]. But while GPS tracking provides more extensive information about users' movements, it does not provide the rich venue and activity information associated with check-in data.

POI prediction and recommendation tasks are different from more traditional recommendation tasks because they involve a more structured, context-rich environment [39]. In addition to user-POI check-in frequencies, the users and POIs are usually associated with a rich set of attributes, such as POI category, spatio-temporal information, personal activity. A heterogeneous graph structure is thus a natural choice to for spatio-temporal POI prediction tasks, since it is more amenable to representing and reasoning with rich context compared to tensor factorization methods (e.g., [33]).

Recently, methods which learn graph representations by *embedding* nodes in a vector space have gained traction from the research community, and graph embedding methods have been widely adopted for a variety of tasks, including text mining [40], online event detection [41] and author identification [42]. In this work, we extend these efforts and propose a network-based embedding method called *Embedding for Dense Heterogeneous Graphs* (*EDHG*). Our approach (i) incorporates personal preferences, temporal patterns, and activity types into a sparse(r) view of the heterogeneous graph, (ii) uses global knowledge of the graph to generate negative samples, (iii) jointly learns vector representations for the nodes in

the graph, i.e., users, POIs, time-slots, and then (iv) uses the learned representations for user and time specific POI recommendation.

We empirically evaluate the effectiveness of *EDHG* using POI prediction and friend suggestion tasks and show that it outperforms previous state-of-the-art POI recommendation methods. Our investigation shows that reason for the improvement stems from the process of (i) heterogeneous graph construction, and (ii) negative sampling. We show that the processes used in previous methods are more suitable to OSN check-in data based on sparse voluntary reporting, than dense(r) check-in data based on location tracking.

To summarize, our work makes the following contributions:

1. Presents the first educational "check-in" dataset and explores its unique mobility and social characteristics;

2. Identifies the challenges for time-aware POI prediction in educational check-in data based on increased density due to location-based tracking (compared to previous voluntary-report LSBN data);

3. Proposes a novel heterogeneous information network-based model to encode the relations between users, POIs, and time-slots, and evaluates its efficacy for POI and user recommendation tasks.

3.2 Data Characteristics

In this section, we discuss the characteristics of the Purdue educational "check-in" dataset and showcase its unique aspects compared to previous check-in data.

3.2.1 Data sample

In this work, we use two sample datasets from the Purdue Office of Instituional Research: (i) WiFi log data and (ii) building location profiles[1]. We consider a sample of the data restricted to freshmen students in the 2016-17 academic year. The 376Gb WiFi log file contains over 1 billion entries, each of which records a data communication between a campus

[1]↑Collected and analyzed anonymously, with IRB approval.

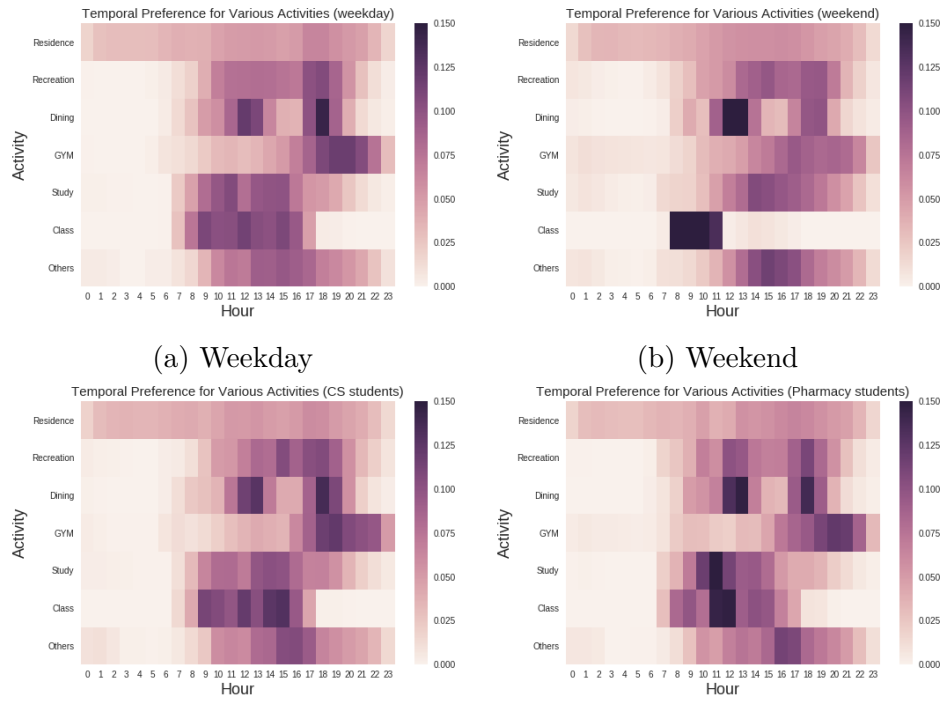
Table 3.1. Dataset description

Item	*Number*	*Description*
Users	6250	Freshmen
POIs	221	On-campus buildings
POI category	4	Academic, Residential, Administration, Auxiliary
POI functionality	7	Residence, Recreation, Dining, Exercise, Library/Lab, Classrooms, Others
Time span	1 sem.	Fall: 08/22/16 to 12/17/16

WiFi access point and a personal device in the time period 7/31/2016 to 6/30/2017. Each entry contains activity time (date, hour and minute), anonymized user id, MAC address, and building id. The building profile provides building information, including building id, name, category, and functionalities. Note that each building belongs to one category but might have multiple functionalities. We remove users with fewer than 100 check-ins. We also drop the check-in records generated by MAC addresses that only checkin at a single building, as these devices are likely to be stationary PCs in dorms or offices. Moreover, using class registration information, we attach a 'in-class' label for the record if the WiFi access point is in the building associated with their course schedule at that time/day. While, we retain these in-class checkins for the analysis in this section, we remove them for the modeling in Section 3.2.5, to focus the prediction task on less predictable user movements. The final processed sample has 540 million logs in total. More dataset statistics are shown in Table 3.1.

3.2.2 Temporal dynamics of user preferences

Figures 3.1a-3.1d show the students aggregated temporal preference for each type of activity in terms of the conditional probability $Pr(time = \tau | activity = a)$ for a given time slot τ and activity a (e.g., Dining). We can see that different activities show unique temporal patterns. For example, on weekdays (Fig. 3.1a) students usually visit the dining halls (i.e., dining activity) around 12pm and 6pm, and go to the gym around 8pm. Check-ins at the residence halls are visible throughout the day, reflecting the variability in students daily routines and the dorms' versatility.



(a) Weekday (b) Weekend

(c) CS (d) Pharmacy

Figure 3.1. Hourly activity preference for a weekday, b weekend, c computer science students, and d pharmacy students

Figures 3.1a and 3.1b show the differences in time preferences for weekday and weekend, respectively. Students only have classes on Saturday morning, and they are more likely to start studying (including staying in the lab or library) at later times on weekends. For non-academic activities on weekends, visiting hours to the gym are more distributed owing to their more flexible schedule, and more students choose to have lunch rather than dinner on campus.

We also investigate if students' temporal preferences vary by major. Figures 3.1c and 3.1d, show the preferences for 302 computer science students and 267 Pharmacy students, respectively. We can see that the overall preferences are similar for *Dining*, while there are some differences in taking classes (shown in activity *Class*), staying in research labs/library (shown in activity *Study*), and exercise (shown in activity *Gym*). Specifically, Pharmacy students attend class more often from 11am to 12pm, while CS students attend class from morning to afternoon. CS students spend more time in academic buildings (from 10am to

7pm) than Pharmacy students who prefer to study in the morning and around noon. For non-academic activities, students from both majors show similar temporal preference, while pharmacy students tend to go to the gym at later times.

3.2.3 Co-visitation behavior

While individual visitation histories can indicate temporal and spatial preference, in isolation they do not indicate relationships among the students. However, *co-visitation* events (i.e., when two students are in the same place at the same time), may be a noisy indicator of relations among students. Any one co-visitation event may be due to random chance, but a larger set of events, particularly when 'in-class' events are dropped, is likely to indicate student friendship. To the best of our knowledge, a study of user pairwise co-visitation events hasn't been investigated in other spatio-temporal analyses.

Since our dataset contains discrete WiFi login records, we merge each user's consecutive logins in the same building, and assume that the user stays in the building throughout this period. For example, if a user checked in at the library every four to eight minutes from 5pm to 6pm with no checkins at other buildings in between, we will merge these check-ins and record that the user stayed in library from 5pm to 6pm. In this way, we augment the visit history with duration time for each user, and use that to compute the pairwise co-visitation count matrix. As each co-visitation is per minute, the pairwise co-visitation count is the total time (in minutes) two users spend together in the same building. For example, the pair of users with the largest co-visitation count spent 38,129 minutes together, which is roughly 27 days, more than 25% of the semester. Note that the in-class check-ins are removed for computing co-visitation. In this way, we only consider the activity outside of class for analyzing co-visitation, which we believe is more informative for determining friend realtions. We will use the pairwise co-visitation count to examine the performance of our embeddings on a friend suggestion task in section 3.3.5.

Once the visit history is augmented with duration time for each user, we compute the number of users visiting the same building at the same time. For each building, we calculate the number of unique visitors for each minute over the semester and filter out the moments

when there are fewer than two visitors. We show the normalized histogram in fig. 3.2. It indicates that the number of users appearing at the same time in the building may reflect building categories, as co-visitation happens more frequently in dorms and the gym compared to academic building (e.g., CS).

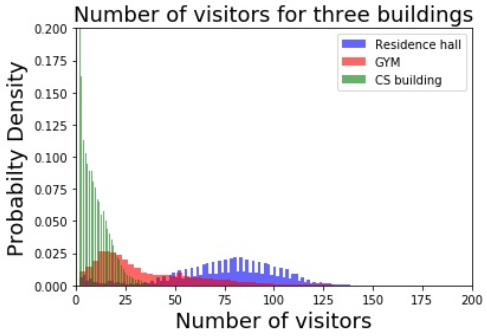
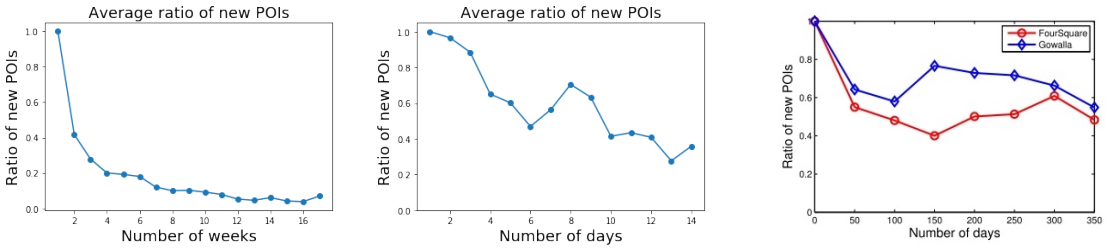


Figure 3.2. Histogram of co-visitation size for an academic building (CS), the Gym, and a residence hall, over all times.

3.2.4 Exploration behavior



(a) Student "check-in" dataset (b) Student "check-in" dataset (c) Foursquare and Gowalla

Figure 3.3. Average ratio of new POIs: (3.3a-3.3b) Purdue data (weeks/days), and 3.3c foursquare and Gowalla dataset (from [43]).

We compare the exploration behaviors in our educational "check-in" dataset to traditional POI recommendation datasets like Foursquare and Gowalla. Figure 3.3 shows the average ratio of new POIs over all users for every new week. For example, the ratio at week two is the proportion of POIs visited during the second week that have not been visited in previously.

Compared with Foursquare users (fig. 3.3c) who keep exploring new POIs all year round, freshmen (fig. 3.3a) appear to explore the campus very quickly (within 2-3 weeks), and then

stick to a fixed range of buildings over the remainder of the semester. But when we zoom into the first two weeks (fig. 3.3b), new students show similar exploration behaviors as in the Foursquare data, with 40 to 60 percent new POIs every day. This provides us with a unique opportunity to model two types of behaviors with different slices of the data: (1) the first few weeks of freshmen semester—exploring new places, and (2) the latter half of the semester—routinely visiting familiar places in a relatively limited activity range.

3.2.5 Proposed *EDHG* Method

In this section, we outline our proposed heterogeneous graph embedding method for POI prediction. Specifically, we consider a time-aware location prediction problem. Given a user and time slot (e.g., Monday 8 am), the model should predict a place that is most likely to be visited.

We refer to our method as *Embedding for Dense Heterogeneous Graphs* (*EDHG*). It is designed specifically to reflect the characteristics of our educational check-in data, which is more dense than traditional LBSN check-in data. To better leverage contextual information, we propose a joint embedding model, which maps user, location, time and activity category into a common latent space.

In this section, we introduce *EDHG* step by step. We first construct a heterogeneous graph using the check-in records, then we learn continuous feature representations for vertices by capturing features of connectivity and structural similarity for pairs of nodes. In Sections 3.2.8-3.2.9, we discuss how to use the learned representations for POI prediction and friend suggestions, respectively.

3.2.6 Heterogeneous Graph Construction

Time indexing scheme. According to our data exploration results, the temporal characteristics of students behavior contain two aspects: (1) periodicity, and (2) preference variance. For example, students' check-ins have clear weekly cyclic patterns. Moreover, students usually visit academic buildings more on weekdays and stay at resident halls more on weekends.

In order to capture these temporal cyclic patterns, we designed a time indexing scheme to encode a standard time stamp to a particular time id. We consider the preference variance in two scales: hours of a day and different days of a week. First, a time stamp is divided into two slices in terms of weekday and hour slot. Next, we split a week into 7 days and a day into the following four sessions:

1. Morning – hours between 6 am and 11:59 am
2. Afternoon – hours between 12 pm and 4:59 pm
3. Evening – hours between 5 pm and 11:59 pm
4. Night – hours between 12 am and 5:59 am

This totals 28 distinct time slot ids, which can represent both weekly and daily preference variance.

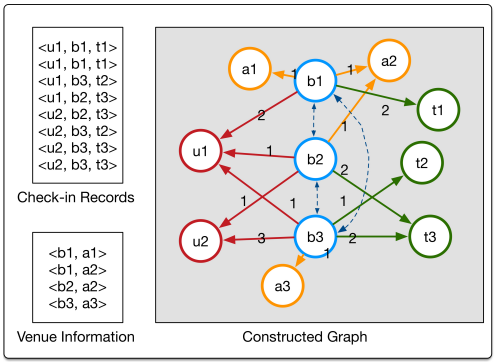


Figure 3.4. Heterogeneous graph constructed using eight example check-in records and venue information

Weighted graph construction. We construct a weighted heterogeneous information network by aggregating the check-in records and venue information. An example is shown in fig. 3.4 with eight check-in records. In this example, u_1, u_2 denote two users, b_1, b_2, b_3 denote three buildings/POIs, t_1, t_2, t_3 denote three time slots, and a_1, a_2, a_3 are three types of activities corresponding to POI functionalities, which we obtained from the venue information. Our model considers three types of edges, i.e., POI-user, POI-time and POI-activity. For POI-time and POI-user edges, edge weights are co-occurrence counts for pair of nodes in

the check-in records. For POI-activity edges, edge weights are set to 1. Our constructed graph contains 6250 user nodes, 221 POI nodes and 7 activities nodes. The POI-user graph density is 22.45%, the POI-time graph density is 82.35%, and the POI-activity graph density is 17.19%.

Note that past work [44] has included POI-POI edges in the graph by considering user transitions from one POI to another. However, these edges increase the average density of our graph substantially (POI-POI density: 60.13%). As we will show in the experiments, these edges degrade the performance of the model, particularly on unvisited nodes, so we do not include them in the graph.

3.2.7 Graph Embedding

We adapt the graph embedding approach from [44], which is an extension of [40] geared for POI recommendation. These approaches are all based on the skip-gram model [14] applied to graphs. Given an instance (word/node) and its context (neighbors), the objective of skip-gram is to minimize the log loss of predicting the context using the instance embedding as input features. We employ a similar objective (as described below), but adjust the negative sampling approach to better fit the characteristics of the heterogeneous graph in our setting.

Specifically, we partition our heterogeneous graph into three bipartite graphs (POI-user graph G_{bu}, POI-time graph G_{bt} and POI-activity graph G_{ba}). Below, we first introduce the graph embedding method for each bipartite graph, then we present our approach for negative sampling, and finally we show how to jointly learn the embeddings over the whole graph.

Bipartite graph embedding. Given a bipartite graph $G_{AB} = (V_A \cup V_B, E)$ where V_A and V_B are two disjoint sets of vertices of different types, and E is the set of edges between them, our task is to find the parameters θ of a model $p_\theta(v_\mathrm{i}|v_\mathrm{j})$ ($v_\mathrm{i} \in V_A$: context vertex; $v_\mathrm{j} \in V_B$: target vertex) that closely approximates the empirical distribution $\tilde{p}(v_\mathrm{i}|v_\mathrm{j})$ in terms of minimizing cross-entropy. Here the empirical distribution is given by the graph, i.e.,

$$\tilde{p}(v_\mathrm{i}|v_\mathrm{j}) = \frac{w_\mathrm{ij}}{deg(\mathrm{j})}$$

where w_ij is the edge weight between v_i and v_j, or zero if v_i and v_j are not connected.

We define the conditional probability of vertex v_{i} generated by vertex v_{j} as the outcome of a softmax function:

$$p_\theta(v_{\mathrm{i}}|v_{\mathrm{j}}) = \frac{\mathrm{e}^{\vec{z_{\mathrm{i}}}^T \vec{z_{\mathrm{j}}}}}{\sum_{\mathrm{i}' \in V_A} \mathrm{e}^{\vec{z_{\mathrm{i}'}}^T \vec{z_{\mathrm{j}}}}} \tag{3.1}$$

where $\vec{z_v}$ denotes the embedding for a vertex v_v. For each vertex v_{j} in V_B, Eq.3.1 defines a conditional distribution $p(\cdot|v_{\mathrm{j}})$ over all the vertices in the set V_A. For each pair of vertices v_{j}, $v_{\mathrm{j}'}$, their second-order proximity can actually be determined by their conditional distributions $p_\theta(\cdot|v_{\mathrm{j}})$, $p_\theta(\cdot|v_{\mathrm{j}'})$.

To learn embeddings that ensure the conditional distribution $p_\theta(\cdot|v_{\mathrm{j}})$ closely approximates the empirical distribution $\tilde{p}(\cdot|v_{\mathrm{j}})$, we minimize the following objective function over the graph G_{AB}:

$$O_{AB} = \sum_{\mathrm{j} \in V_B} \lambda_{\mathrm{j}} d(\tilde{p}(\cdot|v_{\mathrm{j}}), p_\theta(\cdot|v_{\mathrm{j}})) \tag{3.2}$$

where $d(\cdot, \cdot)$ is the KL-divergence between two distributions, and λ_{j} is the importance of vertex v_{j} in the graph. Replacing $d(\cdot, \cdot)$ with KL-divergence, setting $\lambda_{\mathrm{j}} = deg(\mathrm{j}) = \sum_{\mathrm{i} \in V_A} w_{\mathrm{ij}}$ and omitting some constants, the objective function can be written as:

$$O_{AB} = -\sum_{(\mathrm{i},\mathrm{j}) \in E} w_{\mathrm{ij}} \log p_\theta(v_{\mathrm{i}}|v_{\mathrm{j}}) \tag{3.3}$$

Negative sampling. Optimizing the objective in Eq. 3.3 is computationally expensive, as it requires the summation over the entire set of vertices when calculating the conditional probability $p_\theta(\cdot|v_{\mathrm{j}})$. To address this problem, we adopt the approach of negative sampling proposed in word2vec [14], which instead of considering all pairs of nodes, samples a smaller set of observed edges, and then samples multiple "negative" edges for each observed edge. Specifically, in each step, a binary edge $\mathrm{e} = (\mathrm{i}, \mathrm{j})$ is sampled with the probability proportional to its weight w_{ij}, and then multiple negative edges $(\mathrm{i}', \mathrm{j})$ are sampled from a specified noise distribution $q(\mathrm{i}')$.

The default noise distribution used in word2vec (and subsequently used by most, if not all, skip-gram based graph embedding models) is defined as a unigram distribution:

$q(\mathrm{i}) \propto deg(\mathrm{i})^{3/4}$, where $deg(\mathrm{i})$ denotes the degree of vertex v_i. This means that more "popular" vertices are more likely to be selected as negative samples. This makes sense in most NLP and graph embedding problems, where the word co-occurrence matrix or graph adjacency matrix is very sparse. The intuition behind this form of negative sampling is to distinguish between the true context word/vertex and another popular word/vertex which is unlikely to be a context.

However, the graph adjacency matrix is relatively dense in our WiFI check-in data, due to longer user trajectories (i.e., more frequent check-ins). For example, our POI-POI graph adjacency matrix density is 60.13%, whereas in the foursquare dataset the POI-POI graph is extremely sparse with 0.03% density. If we use the above popularity-based negative sampling method for our data, we find that 96% of POI vertices sampled as "negatives" are actually connected to the target vertices—which obviously hinders estimation.

To address this issue, we define a new process for efficient negative sampling utilizing the global statistics, i.e., the graph adjacency matrix. Moreover, we integrate the POI categorical information into the noise distribution. When a POI is from a popular category, it's less likely to be a *true* negative sample, i.e., it's more likely to be connected to the target vertex. By incorporating the global statistics and POI categorical information into the negative sampling procedure, our *EDHG* model incorporates global features into the local predictive method. In practice, we replace the default noise distribution $q(\mathrm{i})$ with alternative $q(\mathrm{i}|\mathrm{j})$. Here v_j is the given target vertex, and v_i is the generated negative sample vertex:

$$q(\mathrm{i}|\mathrm{j}) \propto 1 - \frac{w_{\mathrm{ij}}}{deg(\mathrm{i})} \times Pr(cat(\mathrm{i})) \tag{3.4}$$

where w_{ij} denotes the weight of edge e_{ij}, or equals zero if there is no edge between vertex v_i and v_j, and $deg(\mathrm{i})$ is v_i's degree. $Pr(cat(\mathrm{i}))$ is the ratio of checkins in POIs with same category as POI i, or equals 1 when vertex i is not a POI node. Note that $cat(\mathrm{i})$ corresponds one of the four POI categories in Table 3.1.

Using edge sampling as in [1] and negative sampling as described above, our final objective function for the bipartite graph G_{AB} is:

$$O_{AB} = -\Sigma_{(\mathrm{i},\mathrm{j})\in E} \left[\log \sigma(\vec{z_\mathrm{i}}^T \vec{z_\mathrm{j}}) + \Sigma_{n=1}^{m} \mathbb{E}_{v_{\mathrm{i}'}\sim q(\cdot|\mathrm{j})} \left(\log \sigma(-\vec{z_{\mathrm{i}'}}^T \vec{z_\mathrm{j}})\right)\right] \tag{3.5}$$

Here σ refers to the sigmoid function and we sample m negative examples for each positive example. In our implementation, we use the alias table method from [45] to draw a negative sample with a pre-computed alias table based on the noise distribution $q(\cdot|\mathrm{j})$. This ensures that it takes O(1) time to repeatedly draw samples from the same distribution. In this way we can achieve the same time complexity as the original LINE model, which is demonstrated to be scalable. Then we adopt the asynchronous stochastic gradient algorithm (ASGD) [46] to optimize Eq. 3.5. In each iteration, if the edge $\mathrm{e_{ij}}$ is sampled, the gradient w.r.t. the embedding vector $\vec{z}_{\mathrm{i}}$ of vertex v_{i} will be calculated as $\frac{\partial O_{AB}}{\partial \vec{z}_{\mathrm{i}}}$.

Joint training. The overall objective is the sum of the objectives for three bipartite graphs G_{bu}, G_{bt} and G_{ab}:

$$O = O_{bu} + O_{bt} + O_{ab} \tag{3.6}$$

where each component objective O_{bu}, O_{bt} and O_{ab} is specified by Eq. 3.5. We learn a joint node embedding by iterating through the three component bipartite graphs in a round-robin fashion and updating the vector representations in each bipartite graph embedding procedure. See Algorithm 1 for more details.

3.2.8 Predicting POIs using Embeddings

Once we have trained our model and learned representations for users, time slots, and locations, we can perform location prediction on new check-in data using simple operations on vectors. Given a query $(user, time)$ i.e., $q = (u, \tau)$, we first project the timestamp τ into time slot t using the time indexing scheme described in Section 3.2.6, and then rank the POIs based on their location in the embedding. More precisely, given a query $q = (u, \tau)$, for each POI b, we compute its ranking score as:

$$S(b \mid u, \tau = t) = \vec{z_b}^T \vec{z_u} + \vec{z_b}^T \vec{z_t} \tag{3.7}$$

where $\vec{z_b}, \vec{z_b}, \vec{z_t}$ are embeddings for user u, POI b, time slot t respectively. Then we select the k POIs with the highest ranking scores as predictions. Note the POI embedding $\vec{b}$ reflects

Algorithm 1 *EDHG* training algorithm

Input: Bipartite graphs (POI-user graph G_{bu}, POI-time graph G_{bt}, POI-activity graph G_{ab}), number of iterations N, negative sample size m, vector dimension d.

Output: latent node embeddings for—users: $Z_u \in R^{|U|\times d}$, POIs: $Z_b \in R^{|B|\times d}$, time slots: $Z_t \in R^{|T|\times d}$, and activities: $Z_a \in R^{|A|\times d}$.

procedure JOINT TRAIN($N, m, d, G_{bu}, G_{bt}, G_{ab}$)
    Initialize Z_u, Z_b, Z_t and Z_a
    while $iter \leq N$ **do**
        Bipartite graph embedding(G_{bu}, m)
                ▷ update Z_b, Z_u
        Bipartite graph embedding(G_{bt}, m)
                ▷ update Z_b, Z_t
        Bipartite graph embedding(G_{ab}, m)
                ▷ update Z_a, Z_b
    return Z_u, Z_b, Z_t and Z_a

procedure BIPARTITE GRAPH EMBEDDING(G_{AB}, m)
    sample an edge e_{ij} ($v_i \in V_A, v_j \in V_B$)
    sample m negative nodes from $q(\cdot|j)$ (denote as $v_{i'}$)
    update z_i, z_j, and $z_{i'}$ to minimize Eq. 3.5.

activity information via the POI-activity graph, since our model jointly learns the embedding of multiple relational networks in the same latent space. Therefore, for both visited POIs and unvisited POIs (also called cold-start POIs), we can perform user recommendations using the same scoring function.

3.2.9 Suggesting Friends using Embeddings

As the embeddings learned from the model fuse the interactions between user-POI, POI-time and POI-activity, we can make use of the embeddings to suggest potential friends for a given user based on their pairwise similarity. Specifically, for a query user u, $\forall v \in U \setminus u$, we compute ${z_u}^T z_v$ and rank the results over U, the set of users. From this, we return the top ranked users as people that are more likely to be friends of u.

3.3 Experimental Evaluation

3.3.1 Methodology

In the experiments, we concatenate each student's first 80% check-in records in chronological order to create the training set examples and then use the remaining 20% as the test set. We set the number of iterations (N) to 100M with a batch size of 1, the dimension of the embedding vector (d) is set to 100, and we sample 10 negative samples (m) for each vertex pair.

We use accuracy@k as the measure of prediction effectiveness, which is a commonly used metric for this task (see e.g., [43], [44]). However, in contrast with previous work, which only compare the score of the true POI to the score of *unvisited* POIs during evaluation, we evaluate by comparing the true POI's score to the score of all other POIs (both visited and unvisited). Specifically, for each check-in record (user, time, POI) in the testset, we recommend the top k POIs for the query (user, time) as described in 3.2.8, and determine if the true POI appears in the top-k list (which is defined as a 'hit'). The accuracy@k is defined as the ratio of hits to the testset size.

3.3.2 Comparison Models

We compare our proposed model *EDHG* to baselines, state-of-the-art alternative methods and *EDHG* variants.

NBC: Naive Bayes classifier using (user, time-slot) as joint features. For each query (u, t), the probability of predicting POI b is given by $p(b|u, t) \propto p(u, t|b) \cdot p(b)$ where b denotes a candidate POI, and (u, t) denote a (user, time-slot) pair. This is a strong baseline which takes into account POI popularity and a combination of personal and temporal preference based on counting.

GE [44]: The state-of-art graph embedding method for time-aware POI recommendation (developed using Foursquare and Gowalla data). GE uses POI-POI edges, POI-time edges, POI-region edges, and POI-activity edges, and jointly embeds POIs, times, regions and activities into a latent space. User embeddings are computed as sum of recent visited POIs' embeddings. See Section 3.4 for details.

GE++: An augmented version of GE that we create to assess the effect of learning user embeddings directly during joint training. This version of GE incorporates POI-user edges in the graph, in addition to its heterogeneous graph embedding.

EDHG: Our proposed model, where we include the POI-user graph, the POI-time graph, and the POI-activity graph with our improved negative sampling method.

***EDHG*-NS**: A simplified version of *EDHG* , in which we use the traditional method for generating negative samples based on vertex degree.

***EDHG*-POI**: An augmented version of *EDHG*, where we also include the POI-POI bipartite graph in the heterogeneous graph for learning the embeddings. Note that we only record a POI-POI edge is there is a transition between the two POIs within a four hour time window.

All the models are run on a single machine with 8G memory using 20 threads. Both *EDHG* and its variants are very efficient—it takes about 18 minutes (excluding pre-computation of negative sampling alias table) to process a network with 6486 nodes and 315,407 edges.

Table 3.2. Prediction accuracy

Type	Acc@k / Model	k = 1	k = 3	k = 5	k = 10
visited	GE	0.1079	0.3781	0.5104	0.6543
	GE++	0.3019	0.5190	0.6063	0.6909
	EDHG-NS	0.3321	0.5846	0.7024	0.8137
	EDHG-POI	0.6832	0.7912	0.8368	0.8954
	EDHG	**0.6846**	**0.7915**	0.8367	0.8961
	NBC	0.6765	0.7895	**0.8495**	**0.9016**
un-visited	GE	0.0027	0.0073	0.0241	0.0641
	GE++	**0.0084**	0.0227	0.0332	0.0671
	EDHG-NS	0.0057	0.0128	0.0301	0.0598
	EDHG-POI	0.0034	0.0145	0.0195	0.0334
	EDHG	0.0072	**0.0307**	**0.0360**	**0.0710**
	NBC	5.4e-05	6.3e-04	0.0025	0.0133
total	GE	0.1084	0.3720	0.5026	0.6443
	GE++	0.2981	0.5125	0.5988	0.6828
	EDHG-NS	0.3270	0.5772	0.6937	0.7996
	EDHG-POI	0.6744	0.7811	0.8261	0.8842
	EDHG	**0.6760**	**0.7816**	0.8263	0.8854
	NBC	0.6677	0.7793	**0.8385**	**0.8901**

3.3.3 Predictive Effectiveness

Here we present the experimental results for all prediction methods using well-tuned parameters. Prediction effectiveness in terms of accuracy@k is shown in Table 3.2. We report results for visited and unvisited POIs to highlight the difference between in-sample and out-of-sample performance. We also use the 20% test data to show learning curves for accuracy@1 and @3 in Figures 3.5, 3.6 for visited and unvisited POIs respectively. From the results we can make the following observations:

***EDHG* v.s. *EDHG*-NS**: the full *EDHG* consistently outperforms *EDHG*-NS for both visited and unvisited POIs, with a 100% performance gain in terms of accuracy@1, and 35.4% in terms of accuracy@3. The significant performance gain is due to the improved negative sampling procedures, which selects more informative negative samples for SGD updates. This indicates that it is promising to customize the empirical noise distribution used in negative sampling for various tasks or datasets.

***EDHG* v.s. *EDHG*-POI**: The *EDHG*-POI variant includes the POI-POI transition graph in the original graph which prior work on GE claimed as an important component, but it doesn't improve performance on our recommendation task, and it even downgrades performance for unvisited POIs. This indicates that transition behavior is not informative in our data, as there are too many transitions betweens buildings that cannot be explained by a single reason.

***EDHG* v.s. GE/GE++**: *EDHG* significantly outperforms GE for both visited and unvisited POIs. The reasons might be due to (1) GE using the POI-POI graph to model the "locality" of individual check-ins for Foursquare data. However, as revealed by the comparison between *EDHG* and *EDHG*-POI, including the POI-POI graph doesn't help in our setting. Or (2) GE doesn't include users as entities in their graph representation, but computes the user embeddings based on recent visit histories. Due to the limited number of POIs in our data, computing the user embedding computed in this way may fail to capture personal preferences. Considering the performance of GE++, which we adapt to our data by adding the POI-user graph to the original GE model, modeling users in the graph helps improve its predictions, but the performance of GE++ is still inferior to that of *EDHG*.

***EDHG* v.s. NBC**: *EDHG* achieves comparable performance for visited places and significantly outperforms NBC for unvisited places. In reality, when we look at the learning curves for prediction accuracy, fig. 3.5 shows that our model converges very fast while NBC needs more data to achieve a comparable result; and fig. 3.6 shows that our model actually "learns" how to recommend unvisited places with increasing accuracy, while NBC fails to deal with the cold-start recommendation problem, even when provided with a large amount of training data.

3.3.4 Parameter Sensitivity

Granularity of temporal pattern. In Table 3.2 we evaluated the predictive performance of our model with a combination of weekly of daily pattern using 28 time slots. Here, we design two additional variants to explore the effect of temporal patterns with different

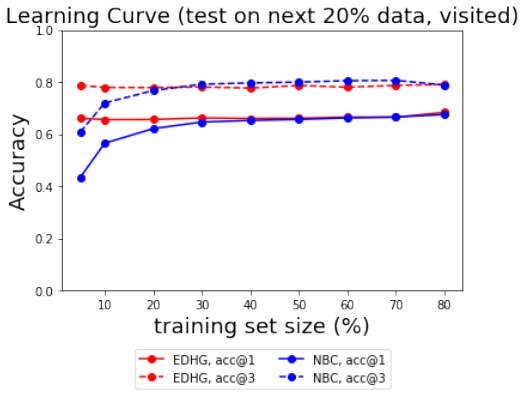


Figure 3.5. Learning curve for visited POIs

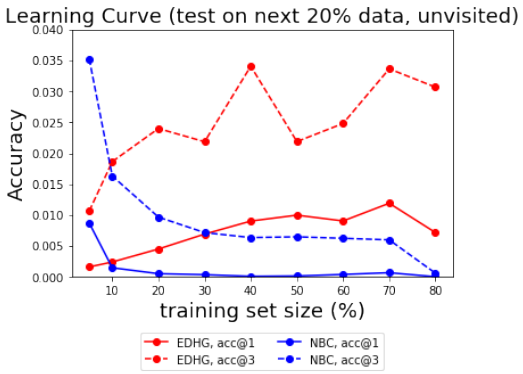


Figure 3.6. Learning curve for unvisited POIs

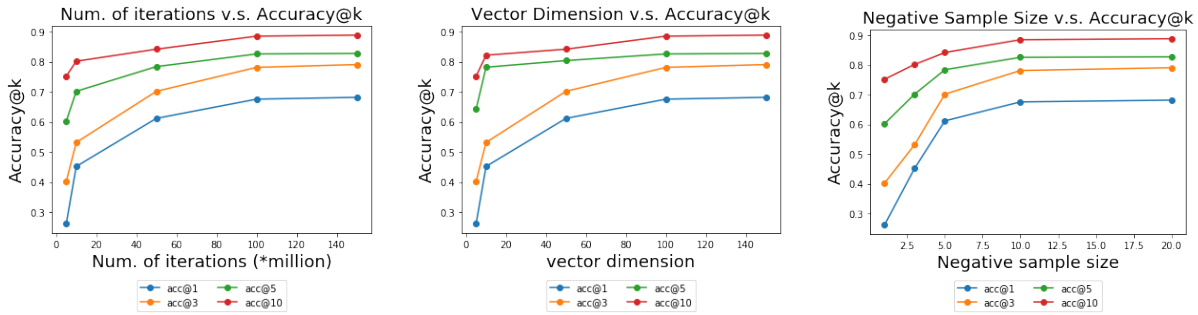
granularity. The *EDHG*-hour only considers time period of day (4 time slots) and *EDHG*-dow only considers day of week (7 time slots). The results are shown in table 3.3.

From table 3.3 we can see that both Day of Week and Hour of Day are important temporal factors. Specifically, when we only consider weekly patterns (day of week), prediction accuracy decreases by roughly 10%; when we only consider daily patterns (hour of day), prediction accuracy slightly decreases by 1.5%. This indicates that time-of-day effects are more significant than day-of-week effects in terms of POI prediction. It's likely that students, particularly freshman, have less flexibility in daily routines due to their course schedule, which makes time-of-day a more important factor for nearly all types of activities.

Number of iterations & vector dimension. Figures 3.7a and 3.7b show the performance of *EDHG* with different number of iterations N and embedding dimensions d. Note that, the units for N is set to 1 million. We can see from fig. 3.7a that the accuracy increases and converges quickly when the number of iterations is larger than 50M. We used $N = 100(M)$

Table 3.3. Prediction accuracy v.s. temporal granularity.

Type	Acc@k / Model	k = 1	k = 3	k = 5	k = 10
visited	*EDHG*-dow	0.6025	0.7010	0.7544	0.8403
	EDHG-hour	0.6727	0.7850	0.8284	0.8927
	EDHG	**0.6846**	**0.7915**	**0.8367**	**0.8961**
un-visited	*EDHG*-dow	0.0041	0.0083	0.0141	0.0402
	EDHG-hour	0.0022	0.0080	0.0260	0.0490
	EDHG	**0.0072**	**0.0307**	**0.0360**	**0.0710**
total	*EDHG*-dow	0.5947	0.6920	0.7448	0.8299
	EDHG-hour	0.6639	0.7727	0.8102	0.8821
	EDHG	**0.6760**	**0.7816**	**0.8263**	**0.8854**



(a) (b) (c)

Figure 3.7. Impact of number of iterations, embedding vector dimension, and negative sample size on prediction accuracy@k.

to ensure convergence. For embedding dimension, we chose $d = 100$ as the accuracy does not increase substantially after that point.

Negative sample size. Figure 3.7c presents the performance of *EDHG* with different numbers of negative samples per example. With more negative samples the accuracy increases, and it plateaus when negative sample size is 10. Therefore, we chose $m = 10$ negative samples for use during optimization.

3.3.5 Friend Suggestion Effectiveness

To examine the efficacy of using *EDHG*'s vector representation for suggesting friends, we first need to identify a proxy signal for evaluation (since we do not have ground truth information about friend relations among the students). Specifically, we consider the following two ways data to determine "true" friends for evaluation:

Covisit As in shown in Section 3.2.3, a co-visitation record is generated when two users check in at the same building at the same time (time unit: minute). In this approach, we identify "friends" of a query user as those with the largest co-visitation counts.

Location Based on the user-building check-in count matrix, we create a ranking list of buildings for each user, with the most frequently visited building ranked highest. In this approach, we identify "friends" of a query user as those that have the smallest distance between the users' ranked list of buildings (using Kendall τ distance). We apply the friend suggestion to the most active users in our dataset, sorted by activity level. For each user,

given a set of "true" friends from one of the baselines above, we evaluate the top 10 friend suggestions from *EDHG* using Mean Reciprocal Rank (MRR). MRR is computed as:

$$\frac{1}{|U|}\sum_{\mathrm{i}=1}^{|U|}\sum_{\mathrm{j}\in F_{\mathrm{i}}}\frac{1}{rank(\mathrm{j})}$$

where U is the set of active users, $F_{\mathrm{i}} = 10$ is the set of "true" friends of user i which are obtained from the data and $rank(\mathrm{j})$ is the rank of item j in the ranking list. We compare the performance of *EDHG* and GE in terms of MRR scores.

Since the **Covisit** baseline encodes both temporal and geographical preference, and the **Location** baseline only takes into account geographical preference, **Covisit** is likely a better proxy for "true" friends, and thus the ideal search results should have a higher MRR score w.r.t. **Covisit**.

The results are shown in fig. 3.8. We can make the following observations based on the results. Comparing **Covisit** and **Location**, *EDHG* suggests friends with more relevance to co-visitation counts than merely geographical preference, while GE does the opposite. Since neither method uses user co-visitation data directly in its model, this implies that *EDHG* captures social behaviors from the spatial-temporal data more accurately. Comparing the general MRR scores of the two models, *EDHG* suggests friends with higher accuracy in general. We also calculated the MMR between the **Covisit** and **Location** friend lists (the green line in the graph). The relatively low MRR score reveals that a large potion of co-visitation behavior cannot be explained by location preference. In reality, students with same major and same year usually stay in a same set of places, e.g. academic buildings and libraries, but their temporal preferences may vary significantly. The plot shows how performance changes as the increase of $|U|$. We can see that, on the co-visitation data, *EDHG*'s MMR score decreases as less active users are included in U. This indicates that *EDHG* discovers better suggestions for more active users, which suggest that with richer check-in information we can capture more precise social relationships.

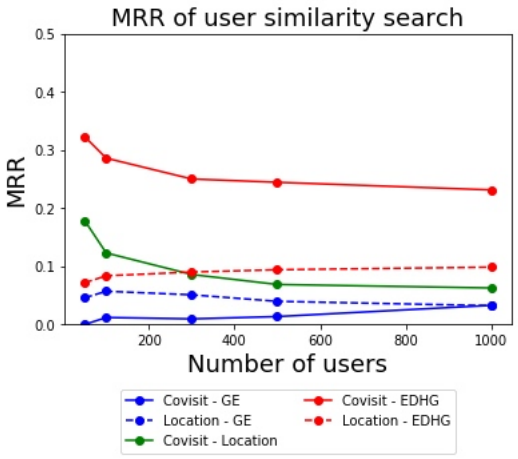


Figure 3.8. Number of frequent users v.s. MRR scores

3.3.6 Visualization of Embeddings

Figure 3.9 shows a visualization of the learned user embeddings, where we project the $d = 100$ dimensions into 2D using t-SNE [47]. From the visualization we can clearly find two clusters of computer science students and pharmacy students (colored green and red respectively). This can be understood through their differences in temporal preference (as shown in Figures 3.1c-3.1d) and also their geographical preferences (i.e., these two majors share very few academic buildings).

3.4 Related Work

POI recommendation methods have received extensive research attention in the last five years, and many approaches have been proposed. For example, Wang et al. ([35], [48]) applied sparse additive generative models to incorporate multiple factors for POI recommendation. [34] proposed a Spatial-Aware Hierarchical Collaborative Deep Learning model (SH-CDL), which jointly performs deep representation learning for POIs and hierarchically additive representation learning for spatial-aware personal preferences. [44] proposed a graph embedding model GE for context-aware POI recommendation, which uses the POI-POI transition graph, POI-region graph, and POI-category graph, and jointly learn the representations for POI, region, and time with the same method as PTE [40]. User embeddings are then computed

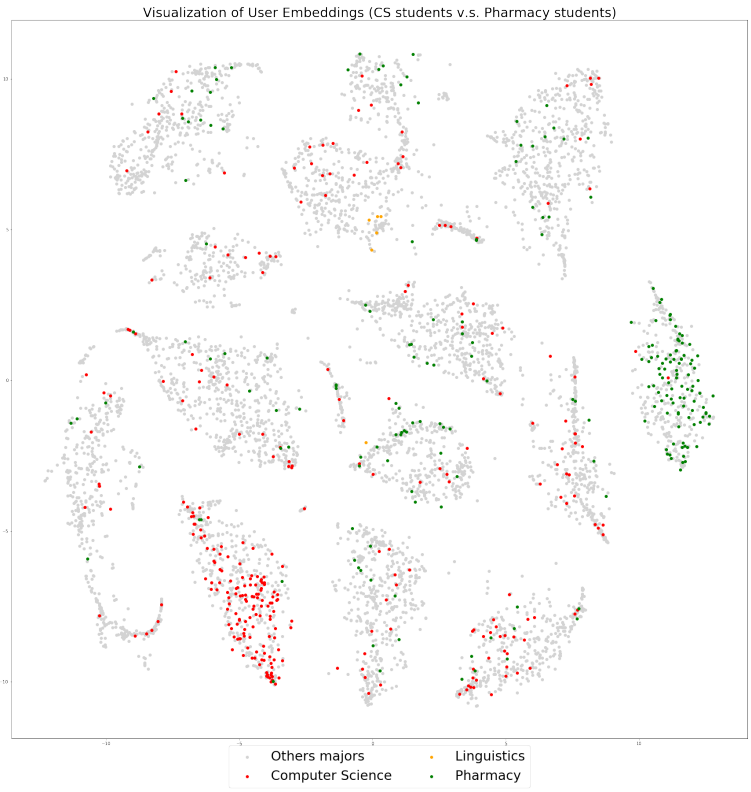


Figure 3.9. User embeddings

as weighted sum of recent POI embeddings, and with user, POI, time embeddings they can perform time-aware POI recommendation. GE achieved better performance than all previous work on this task, which is why we use it as a baseline for evaluation in this paper.

In addition, our work is related to the extensive literature on network embedding, which has attracted a great deal of attention in recent years. Many of these recent methods are technically inspired by Skipgram [14]. For example, Deepwalk [3] uses the embedding of a node to predict the context in the graph, where the context is generated by a random walk. Metapath2vec [15] extends DeepWalk for heterogeneous graph embedding. LINE [1] extends the skip-gram model to have multiple context spaces for modeling both first and second order proximity. PTE [40] adapts the LINE model [1] for embedding bipartite networks. Note that PTE model is directly adopted by GE [44] for POI recommendation, and we further adjust PTE to our setting by improving the negative sampling method and targeting the graph construction process.

Heterogeneous information network embedding has been broadly applied to multiple tasks. For example, [40] predicted text embeddings based on heterogeneous text networks which showed great potential in document classification. [41] proposed ReAct, a method that processes continuous geo-tagged social media (GTSM) streams into a heterogeneous graph and obtains recency-aware activity models on the fly, in order to reveal up-to-date spatiotemporal activities. [42] proposed a task-guided and path-augmented heterogeneous network embedding for author identification task.

3.5 Discussions

This paper presents our analysis of the first educational "check-in" dataset and proposes *EDHG*, a heterogeneous graph embedding based method to model more dense spatio-temporal checkin activity. To account for the unique characteristics of the data, we improve the negative sampling method to incorporate global statistics of the graph/data into the noise distribution. We also show that it's better to drop the POI-POI transition edges when the check-in data is more dense. We evaluated *EDHG* with two tasks: time-aware POI prediction and friend suggestion. On both tasks, our proposed model outperforms the previous state-of-art methods and baselines. These initial results indicate the promise of using student trajectory information for personalized recommendations in education apps, as well as in predictive models of student retention and satisfaction.

Several interesting research problems remain for further exploration. For example, we did not make direct use of the co-visitation data in the model but rather withheld it for evaluation of the friend suggestions. We plan to incorporate it in the training process and see whether social interactions impact student checkin behavior. Also, inspired by [49], we may be able to further improve the negative sampling by dynamically selecting informative negative samples during each SGD update.

4. REQE: TOWARDS REPRESENTATION-BASED EQUIVALENCE FOR NODE EMBEDDINGS, IN SINGLE GRAPHS AND SETS OF GRAPHS

4.1 Introduction

The goal of structural role discovery is to identify nodes with topologically similar network neighborhoods that reside in potentially distant parts of the network. For example, in a protein-protein interaction network, two proteins with the same functionality might not interact with each other or have common neighbors, but may be embedded in similar subgraphs. In social network analysis, regular equivalence [20] has been used to capture this idea through the role/identity of vertices in a network. The notion of social "roles" is a centerpiece of most sociological theorizing [50]. The concept of regular equivalence, and the methods used to identify and describe regular equivalence sets, correspond quite closely to the sociological concept of roles.

However, since structural roles of nodes are typically defined over a discrete space, it limits their usage for a variety of real-world tasks including visualization, link prediction, anomaly detection, and recommendation. There is existing work relying on hand-crafted distances or recursions to partition nodes into equivalent classes. For example, RolX [51] uses matrix factorization to generate mixed membership across the roles based on a collection of structural features.

On the other hand, a considerable amount of recent attention is devoted to the development of graph/node embedding methods, which are designed to produce continuous vector-valued node embeddings that capture patterns in network structure. These methods have been extended to different types of graphs, e.g. attributed graphs, hierarchical graphs, and multi-relational graphs (e.g. knowledge graphs), and have been widely adopted for a variety of tasks, including text mining, on-line event detection, and knowledge completion. Much of the work in graph embedding based on random walks has been designed to preserve node proximity (e.g. LINE [1], Node2vec [2], DeepWalk [52]) explicitly or implicitly. Therefore, they fail to capture the structural relatedness among vertices that reside in different parts of

the graphs. Recently, there have been some node embedding models considering structural similarity or social role. These methods are mostly based on particular structural properties including degree, centrality, or motifs. We review these methods in the next section.

In this work, we formally define *representation-based equivalence* for both automorphic equivalence and regular equivalence. We then develop a flexible framework to learn node embeddings in an unsupervised manner, which aims to preserve node equivalence with respect to a specified set of structural properties by combining aspects of automorphic and regular equivalence.

Capturing the notion of node equivalence enables us to move towards preserving node isomorphism in the embedding. We note that the recursive definition of node roles in regular equivalence (i.e. the role of the node depends on its neighbors' roles) resembles the iterative process of the Weisfeiler-lehman isomorphism test (WL test) [53]. Recently there has been increasing attention on analyzing the expressiveness of node representations learned with Graph Neural Networks (GNNs) in relation to the WL test [8]. However, the unsupervised variants of GNNs are all trained with a proximity-based loss function similar to Node2vec [2], which means their models aim to capture local proximity rather than topological roles, and this limits their expressivity.

Specifically, our approach—*Reqe*—leverages recurrent neural networks to reflect the recursive nature of node roles definition, combined with negative sampling based on structural properties and degree-guided regularization. Our framework is general enough to also include node attributes as additional properties, as real world structural roles might need to account for node labels/types, or latent structures that are correlated with attributes. We show theoretically that the expressive power of *Reqe* in terms of capturing node automorphic equivalence comes from encoding long-range structural properties, as well as performing recursive neighbor propagation. For evaluation, we design three different tasks to test the model's effectiveness at (i) preserving equivalences in the learned embeddings, (ii) predicting other structural properties not included in the optimization, and (iii) predicting labels in real-world node classification tasks. We compare our proposed method *Reqe* to several state-of-the-art graph embedding methods and show that *Reqe* out-performs the competing methods in all of the three tasks.

To summarize, in this work we make the following contributions:

1. Formalize the notion of *Representation-based automorphic/regular equivalence* for both single graph and multiple samples of graphs, which bridges the gap between equivalences and node embeddings;

2. Introduce a novel and flexible framework *Reqe* to learn *structural* node embeddings w.r.t specified structural properties and node attributes, and theoretically analyze its expressive power;

3. Extensively evaluate the proposed framework *Reqe* on multiple tasks and real-world datasets from a variety of applications areas.

4.2 Representation-based Equivalence

4.2.1 Background

Below we outline several definitions of node equivalence in a graph $G = (V, E)$ with nodes $v \in V$ and edges $\mathrm{e_{ij}} \in E$. Let $\mathcal{N}(\mathrm{i})$ be a function that returns the set of neighbors of node v_{i} (i.e., $\{v_{\mathrm{j}}\}_{\mathrm{e_{ij}} \in E}$). The relationship between the equivalences is: Structural equivalence $\implies$ Automorphic equivalence $\implies$ Exact regular equivalence $\implies$ Regular equivalence.

Definition 4.2.1. (Strong) structural equivalence *[18] [19]*

Two vertices u and v are strongly structural equivalent if, and only if, they have the same neighborhoods, i.e.

$$s(u) = s(v) \iff \mathcal{N}(u) = \mathcal{N}(v)$$

Definition 4.2.2. Automorphic equivalence *[21], [54] Two nodes u and v are automorphically equivalent if all the vertices can be re-labeled to form an isomorphic graph with the labels of u and v interchanged. Two automorphically equivalent vertices share exactly the same label-independent properties.*

We use $u \equiv v$ to denote automorphic equivalence.

Definition 4.2.3. Exact regular equivalence *[55]. Two nodes u and v are exact regular equivalent if they have the same node role, and role is defined recursively via neighbors' roles:*

$$r(u) = r(v) \iff multiset\{r(u')|u' \in \mathcal{N}(u)\} = multiset\{r(v')|v' \in \mathcal{N}(v)\}$$

Exact regular colorations in the context of social networks were first discussed by Borgatti and Everett [56] but were not formally defined until later in Everett and Borgatti ([57]) where they were called exact colorations.

Regular equivalence is a similar concept but relaxes the *multiset* to *set*, which means two nodes have same role if their neighbors have same set of roles. Note that this means two regularly equivalent nodes do not necessarily have the same degree. In this paper we focus on *exact* regular equivalence, with multiset comparisons.

Next we define structural properties of nodes in a graph.

Definition 4.2.4. Structural property *[56]. Given a graph $G = (V, E)$, the property $t : V \to \mathcal{R}$ is structural if for every automorphism ϕ of the graph G and every vertex $v \in V$, $t(v) = t(\phi(v))$ holds. If two nodes are automorphically equivalent, then they must share the same structural properties, i.e.*

$$u \equiv v \implies t(u) = t(v)$$

Definition 4.2.5. Complete structural property *[56]. A collection of structural properties $\mathcal{T}_C = \{t_1, t_2, ..., t_m\}$ is complete (for automorphic equivalence) iff for each pair of vertices u and v,*

$$(\forall \mathrm{i}, 1 \leq \mathrm{i} \leq m, t_{\mathrm{i}}(u) = t_{\mathrm{i}}(v)) \iff u \equiv v$$

It was originally defined w.r.t structural equivalence in [56], but it also holds for automorphic equivalence since $(\forall \mathrm{i}, 1 \leq \mathrm{i} \leq m, t_{\mathrm{i}}(u) = t_{\mathrm{i}}(v)) \implies s(u) = s(v) \implies u \equiv v$.

Examples of structural properties are degree, degree at distance k, centrality. To date there isn't a known collection of structural properties that is proved complete [58]. However, since any structural property will obey Definition 4.2.4, the more properties that are added to $\mathcal{T}$, the closer it will be to complete.

4.2.2 Representation equivalence

We propose to learn low-dimensional embedding that projects the graph to continuous space which approximates equivalence in the original graph(s).

Regular equivalence embedding with a single graph

We assume an embedding that aims to encode a set of structural properties $\mathcal{T}$, i.e. $\forall u, v \in \mathbf{V}$, we have

$$\forall t \in \mathcal{T}, t(u) = t(v)) \implies \boldsymbol{D}(\boldsymbol{z}_u, \boldsymbol{z}_v) < \epsilon \tag{4.1}$$

where $\boldsymbol{z}_u$ and $\boldsymbol{z}_v$ are embedding vectors of nodes u and v, $\boldsymbol{D}(\cdot, \cdot)$ is a standard distance measure, e.g. euclidean distance. Furthermore, for a node w s.t. $\exists t \in \mathcal{T}_C, t(u) \neq t(w)$, the embedding will aim to differentiate u and w by embedding them in different locations since they will be viewed as a negative example.

Then we define *representation-based automorphic equivalence* and *representation-based regular equivalence* w.r.t. embeddings that encode structural properties.

Definition 4.2.6. Representation-based automorphic equivalence. *Given a graph G, suppose embedding $\boldsymbol{Z}$ encodes a set of structural properties $\mathcal{T}$ as defined in Equation (4.1). If $\mathcal{T}$ is the complete structural properties $\mathcal{T}_C$ (Definition 4.2.5), then we have $u \equiv v \iff D(z_u, z_v) = 0$.*

Similarly, we propose representation-based (exact) regular equivalence in similar vein as Definition 4.2.6.

Definition 4.2.7. Representation-based regular equivalence. *Given a graph G, suppose embedding $\boldsymbol{Z}$ encodes a set of structural properties $\mathcal{T}$ as defined in Equation (4.1). Two nodes are representation-based regular equivalent w.r.t to $\mathcal{T}$ iff:*

1. $\boldsymbol{D}(\boldsymbol{z}_u, \boldsymbol{z}_v) = 0$
2. $\boldsymbol{D}(multiset(\boldsymbol{z}'_u | u' \in \mathcal{N}(u)), multiset(\boldsymbol{z}'_v | v' \in \mathcal{N}(u)) = 0$

Note here we primarily consider network characteristics (i.e., topological roles) as structural properties, but node attributes can also be included as properties in $\mathcal{T}$.

Regular equivalence embedding with multiple graphs

We first extend the notion of exact regular equivalence to the multiple graph case w.r.t a graph distribution defined by $\boldsymbol{A}^*$.

Definition 4.2.8. Exact regular equivalence with multiple graphs. *Assume a set of aligned graphs* [1] *are sampled from a probabilistic adjacency matrix* $\boldsymbol{A}^*$ *where* $\boldsymbol{A}^*_{\mathrm{ij}}$ *is the probability of connecting node* i *and* j. *Two nodes are regular equivalent if and only if they have the same node role* $r(v)$*, where node role is defined recursively as:*

$$r(\mathrm{i}) = r(\mathrm{j}) \iff$$
$$multiset\{(r(\mathrm{i}'), \boldsymbol{A}^*_{\mathrm{i},\mathrm{i}'}) | \mathrm{i}' \in \mathcal{N}(\mathrm{i})\} = multiset\{(r(\mathrm{j}'), \boldsymbol{A}^*_{\mathrm{j},\mathrm{j}'}) | \mathrm{j}' \in \mathcal{N}(\mathrm{j})\}$$

This is a very strict definition as it requires the neighboring roles should have exactly the same probability distribution. In practice, we're given a set of generated samples $\mathcal{G}$ instead of true distribution $\boldsymbol{A}^*$, so we can only estimate empirical distribution $\widehat{\boldsymbol{A}}$ from samples.

Definition 4.2.9. Probabilistic structural property. *Given a probabilistic adjacency matrix* $\boldsymbol{A}^*$ *over node set* $\mathcal{V}$*, for any graph* G *containing the same node set, the probability of generating graph* G *is* $P(G) = \Pi_{(\mathrm{i},\mathrm{j})\in E_G} \boldsymbol{A}^*_{\mathrm{ij}} \Pi_{(\mathrm{i},\mathrm{j})\notin E_G} (1 - \boldsymbol{A}^*_{\mathrm{ij}})$. *For any structural property* $t : V \to \mathcal{R}^{|V|}$*, we define the probabilistic structural property* t^* *as a random variable, and* $P(t^* = \tau) = \sum\{P(G) | t_G = \tau\}$ *where* $t_G \in \mathbb{R}^V$ *is the structural property value in graph* G*. Note that different structural properties are not independent with graphs samples, i.e. for two structural properties* t_1 *and* t_2*,* $P(t_1^*, t_2^*) \neq P(t_1^*)P(t_2^*)$.

Based on Definition 4.2.5, we define two nodes to be automorphically equivalent w.r.t $\boldsymbol{A}^*$ *if and only if then share the same joint distribution of complete probabilistic structural*

[1] ↑aligned graphs are a set of graphs on the same set of nodes, where the node ids are shared across the graphs so that we know the correspondence between the nodes across the graphs.

properties. Similar to the single graph case, assume the embedding encodes a set of structural properties $\mathcal{T}$, i.e.

$$P(t_1^*(u), t_2^*(u), \cdots, t_m^*(u)) = P(t_1^*(v), t_2^*(v), \cdots, t_m^*(v)) \tag{4.2}$$

Definition 4.2.10. Representation-based automorphic equivalence with multiple graphs.

Given a set of graphs $\mathcal{G} = \{G_1, G_2, \cdots, G_K\}$, suppose embeddings $\boldsymbol{Z}$ encodes a set of structural properties $\mathcal{T}$ as defined in Equation (4.2), then if the set is complete (i.e. $\mathcal{T} = \mathcal{T}_C$), we have $u \equiv v \iff D(z_u, z_v) = 0$.

We list three necessary conditions for automorphic equivalence with multiple graphs below, including the sufficient condition (A) as $u \equiv v \iff (A) \implies (B) \implies (C)$.

1. $P(t_1^*(u), t_2^*(u), \cdots, t_m^*(u)) = P(t_1^*(v), t_2^*(v), \cdots, t_m^*(v))$
2. $\forall t \in \mathcal{T}_C, t^*(u) \overset{d}{=} t^*(v)$
3. $\mathbb{E}[(t_1^*(u), \cdots, t_m^*(u))] = \mathbb{E}[(t_1^*(v), \cdots, t_m^*(v))]$

We show in Figure 4.1 that having strictly more information (all the properties values v.s. quantiles of properties v.s. average values of properties) when learning the embeddings makes the embedding more accurate in terms of capturing automorphic equivalence. However the tradeoff is that is also computationally more expensive.

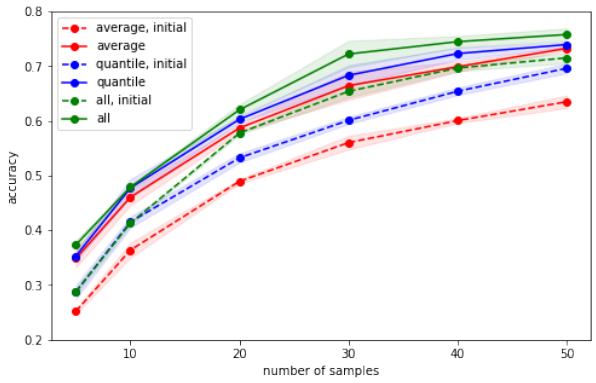


Figure 4.1. Average v.s. quantiles v.s. full data as model input

We then propose Representation-based regular equivalence w.r.t. multiple graphs.

Definition 4.2.11. Representation-based regular equivalence with multiple graphs. *Given a set of graphs* $\mathcal{G} = \{G_1, G_2, \cdots, G_K\}$ *and a set of structural properties* $\mathcal{T} = \{t_1, t_2, \cdots\}$, *suppose embeddings* $\boldsymbol{Z}$ *encodes a set of structural properties* $\mathcal{T}$ *as defined in Equation (4.2).* $\boldsymbol{z}_u$ *and* $\boldsymbol{z}_v$ *are embedding vectors of nodes* u *and* v, *then* u *and* v *are representation-based regular equivalent w.r.t* $\mathcal{G}$ *and* $\mathcal{T}$ *iff:*

1. $\boldsymbol{D}(\boldsymbol{z}_u, \boldsymbol{z}_v) = 0$
2. $\mathbb{E}_{u',v'}[\boldsymbol{D}(multiset(\boldsymbol{z}'_u | u' \sim \widehat{\boldsymbol{A}}_u), multiset(\boldsymbol{z}'_v | v' \sim \widehat{\boldsymbol{A}}_v)] = 0$

$\widehat{\boldsymbol{A}}$ is the (empirical) linkage probability matrix, i.e. $\widehat{\boldsymbol{A}}_{uv}$ is computed as the number of links between u and v divided by total number of graphs, and then normalized for sum of 1. The more samples we have, the closer empirical distribution $\widehat{\boldsymbol{A}}$ is to true $\boldsymbol{A}^*$.

4.3 Proposed framework: *Reqe*

We propose a general framework *Reqe* which aims to preserve automorphic equivalence when learning a node embedding based on a set of given node properties. The key idea of *Reqe* is to combine aspects of Definitions 4.2.6 and 4.2.7: (1) using more complicated structural properties to get closer to the complete structural properties $\mathcal{T}_C$, and (2) apply role-based neighbor propagation to capture longer-range structural properties through recursive steps, which offsets a fixed set of properties in $\mathcal{T}$.

We first describe the single input graph case (Algorithm 3) and then introduce the extension to multiple graphs case (Algorithm 4). Formally, given an input graph $G = (V, E)$, and a set of properties $\mathcal{T} = \{t_1, t_2, ...\}$, we aim to learn a d-dimensional embedding $\mathcal{E}$ that is representation-based regularly equivalent. Here we select degree centrality, stationary distribution and closeness centrality as examples of structural properties.

Initialization. The node embedding is initialized according to a set of structural property. Given a set of structural properties, we initialize the embedding as concatenation of vectors that are sampled from a Normal distribution centered at the property values. For example, if

$|\mathcal{T}| = 2$ and t_1 is degree centrality $\boldsymbol{d} \in \mathbb{R}^V$ and t_2 is closeness centrality $\boldsymbol{c} \in \mathbb{R}^V$, then the embedding of node i is initialized as

$$Z_{\mathrm{i},k} \sim \begin{cases} \mathcal{N}(\boldsymbol{d}_{\mathrm{i}}, \sigma^2) & 0 \leq k \leq d/|\mathcal{T}| \\ \mathcal{N}(\boldsymbol{c}_{\mathrm{i}}, \sigma^2) & d/|\mathcal{T}| < k < d \end{cases} \tag{4.3}$$

where d is the dimension of node embedding, and we use σ^2 of 0.01 in our implementation.

Stationary distribution as structural property. We compute the stationary distribution as a structural identity feature, which is then used for negative sampling. First we compute a $V \times V$ matrix $\boldsymbol{S} \in \mathbb{R}^{V \times V}$ via personalized PageRank [59], where each entry $\boldsymbol{S}[s,t]$ is the probability that a random walk starting from vertex s will stop at vertex t. Therefore, each column $\boldsymbol{S}[:,t]$ encodes the stationary distribution of vertex t. We use the K largest values of $\boldsymbol{S}[:,t]$ as an approximation of stationary distribution of vertex t. Since the value of $\boldsymbol{S}$ follows a power law distribution, the K largest values include most of the information. We then construct a full structural feature matrix by concatenating the transposed stationary distribution matrix $\boldsymbol{S}^T$ with all the other given properties. It is a $V \times (|\mathcal{T}| + K)$ matrix with each row encoding the set of structural properties for the associated node. The Algorithm 2 below computes stationary distribution features in a given graph, which is used in our proposed *Reqe* for both single graph and multiple graphs cases.

Algorithm 2 Compute Stationary Distribution

procedure STAT_DIST(G, K, α)
  $\boldsymbol{S} \leftarrow \boldsymbol{0}$
  for $v \in V$ **do**
    $P_v \leftarrow \text{RootedRandomWalk}(G, v, \alpha)$
    $\forall u \in P_v, S[v,u] \leftarrow S[v,u] + 1$
  Sort $\boldsymbol{S}[:,u]$ in descending order
  $\boldsymbol{S}[:,u] \leftarrow \boldsymbol{S}[:,u]/sum(\boldsymbol{S}[:,u])$
  $\boldsymbol{S} \leftarrow \boldsymbol{S}[:,:K]^T$
  Return $\boldsymbol{S}$

Negative sampling. To encode the structural properties in the node embedding (as discussed in Section 4.2.2), we adopt negative sampling procedure. We first construct a normalized dissimilarity matrix $\boldsymbol{D}$ from the structural feature matrix, where each entry $\boldsymbol{D}_{\mathrm{ij}}$ is computed as the dissimilarity value between the structural features of node v_{i} and v_{j}, and each row of $\boldsymbol{D}$ sums to 1. There are multiple choices of similarity measure, e.g. cosine similarity, Euclidean distance, and we use cosine similarity for simplicity.

We then use the dissimilarity matrix for negative sampling and incorporate the following loss.

$$\mathcal{L}_{ns} = -\sum_{v \in V} \sum_{\mathrm{i}=1}^{m} \|\boldsymbol{z}_v - \boldsymbol{z}_{u'_{\mathrm{i}}}\|^2$$

where node u'_{i} is a negative sample drawn from the v-th row of the normalized dissimilarity matrix, i.e. $u' \sim \boldsymbol{D}[v, :]$

Neighborhood role embedding. Note the recurrent nature of the condition (3) in the definition of representation-based regular equivalence, i.e. the role of the center node can be determined from the role of the neighboring nodes. To encode this information, we consider a sequence model to represent the aggregated neighbors' role. The intuition is that if the current embedding reflects the structural role information, then the aggregated neighborhood embedding should reflect the multi-set of the neighbors' roles, which is indicative of the center node's role.

In practice, we generate a list of neighbors of fixed length T by uniform sampling over all neighbors and sort by node degree, then feed the list of neighbors to a recurrent neural network (RNN) to produce neighborhood role embedding. The same neighbor sampling procedure is also used in GraphSAGE [17], an inductive graph neural network, to propagate node attribute information. Popular RNN architectures are LSTM unit and GRU. We examined both two variants and achieved comparable performance, thus we use GRU as default for efficiency reason, but we also provide a variant in our implementation code based on LSTM.

The following loss function is considered for neighborhood aggregation.

$$\mathcal{L}_{neighbor} = \sum_{v \in V} \|\boldsymbol{z}_v - GRU(\{\boldsymbol{z}_u | u \in \mathcal{N}(v)\})\|^2$$

Degree-guided regularization. As node degree is shown to be very important for preserving structural similarity and easy to compute, we add regularization guided by node degree. The intuitive idea is that the learnt embedding should be able to predict the degree.

$$\mathcal{L}_{reg} = \sum_{v \in V} \|\log(deg_v + 1) - MLP(\boldsymbol{z}_v)\|^2$$

The final loss function is the weighted sum of the three components.

$$\mathcal{L} = \mathcal{L}_{neighbor} + \alpha \mathcal{L}_{ns} + \lambda \mathcal{L}_{reg} \tag{4.4}$$

Here α and λ are two hyper-parameters which controls the relative importance of different factors, and they can be tuned by doing grid search over them. We assess the sensitivity of these hyper-parameters in Section 4.6.6. Algorithm 3 describes the overall training process for the single graph case.

Note that if the complete structural property $\mathcal{T}_C$ (Definition 4.2.5) is used as model input, then the *initialization* step (Equation (4.3)) will capture the automorphic equivalence in the original graph according to Definition 4.2.6[2]. However, since we do not have the complete structural property in practice, the initial embedding will not sufficiently differentiate non-isomorphic nodes due to the use of a limited set of properties in $\mathcal{T}$. The three components in the loss function – LSTM neighborhood encoder, negative sampling and degree-guided regularizer will update the initial embedding towards encoding automorphic equivalence by capturing longer-range structural properties, which makes up for $\mathcal{T}$ being not complete. For example, if two non-equivalent nodes have the same structural properties in $\mathcal{T}$, then they have the same initial embedding. However, their neighbors might have different properties and thus different embeddings, then the LSTM neighbor propagation is able to capture the difference in the neighborhood, which helps the model differentiate the two non-equivalent nodes through recursive steps.

Extension to multiple graphs. When the input data include multiple aligned graph samples, we estimate the necessary condition (C) of representation-based automorphic equivalence (Def. 4.2.10). Note, we choose to estimate (C) rather than (A) for computational efficiency.

[2]↑note that this requires $\sigma^2 = 0$ in Equation (4.3)

First, we compute the structural properties in each graph separately and use the average across the graphs for embedding initialization and for computing dissimilarity matrix as described above. Second, we use importance sampling when generating lists of samples, i.e. giving more importance to the neighbors appearing in more graph samples. The overall loss function and training procedures remain the same as the single graph case. Algorithm 4 describes the training process for the multiple graphs case. Specifically, the differences from the single graph case appear in lines 2-5 and line 12.

Complexity analysis. The total time complexity of *Reqe* including pre-processing is $\mathcal{O}(|V|^2 + |V|^k)$ where k depends on the complexity of computing the chosen structural properties. In our experiments, the most expensive property to compute is betweenness centrality (BC), which has a theoretical complexity of $\mathcal{O}(|V|^3)$. In practice, the pre-processing step wasn't a barrier due to the availability of approximate algorithms [60] to compute BC in $\mathcal{O}(|E|)$ time. For example, it took 4.69 minutes to compute all five structural properties (including BC) on the Cora network. For the multiple graphs case, suppose we have M graph samples, the pre-processing step involves computing the chosen structural properties on each graph, which takes $\mathcal{O}(M \cdot |V|^k) = \mathcal{O}(|V|^k)$. Thus the asymptotic complexity remains the same.

Specifically, *Reqe* takes two steps: the pre-processing step and model training. In the pre-processing step, we first compute the specified structural properties $\mathcal{T}$ and concatenate all structural properties into a f-dimensional feature vector for each node. Let $\mathcal{O}(|V|^k)$ refer to the complexity of the most time-consuming structural property, where $k \geq 0$. We then compute a pairwise dissimilarity table from the feature matrix of size $|V| \times f$, which takes $\mathcal{O}(f|V|^2)$. The overall complexity of the offline pre-processing is $\mathcal{O}(f|V|^2 + |V|^k))$ where k depends on the specified structural property. Note that any factorization/clustering methods will also take $\mathcal{O}(|V|^2)$ complexity due to the computation of pairwise distances. During model training, in each iteration, the time complexity of calculating gradients and updating parameters for each node $v \in V$ is $\mathcal{O}(\deg_v d^2)$, where $\deg_v$ is the degree of node v, and d is the embedding dimension. It also takes $\mathcal{O}(m)$ time for sampling m negative samples for each node. Since we sample a fixed number of neighbors (T) for each node, the time for each node is bounded by $\mathcal{O}(Td^2 + m)$. Therefore, the overall training complexity

is $\mathcal{O}(|V|(Td^2 + m))$. The embedding dimension d and the neighborhood size T are set to a constant number that is independent of graph size, thus the complexity of the model training process is linear in the number of nodes $|V|$. The total time complexity including the preprocessing is $\mathcal{O}(|V|^2 + |V|^k + |V|) = \mathcal{O}(|V|^2 + |V|^k)$ where k depends on the complexity of computing the chosen structural properties.

Compared to our main competitors struc2vec[61] and DRNE[52], *Reqe* training process has the same time complexity as DRNE for each iteration ($\mathcal{O}(|V|)$). [52] empirically validated that Struc2vec is more expensive with a complexity of $\mathcal{O}(|V|^{1.5})$ per iteration.

Algorithm 3 *Reqe* training algorithm (single graph)

Input: Graph G with V nodes, structural properties t, no. of iterations I, stationary dist. feature size K, embedding dimension d, jumping probability α, negative sample size m, neighbor sample size n.

Output: latent node embeddings $\boldsymbol{Z} \in \mathbb{R}^{V \times d}$

procedure MODEL TRAINING
  $\boldsymbol{S} \leftarrow$ Stat_dist(G, K, α) *compute stationary distribution features*
  $\boldsymbol{S} \leftarrow [\boldsymbol{S}, t]$
  $\boldsymbol{D} \leftarrow 1 -$ cos_sim$(\boldsymbol{S})$
  Initialize $\boldsymbol{Z} \sim \mathcal{N}(\boldsymbol{S}, \sigma^2)$
  while $iter \leq I$ **do**
    for $v \in V$ **do**
      Sample m negative nodes $u' \sim \boldsymbol{D}_v$
      Sample n neighbors u uniformly over all neighbors
    Update $\boldsymbol{Z}$ according to Eq. 4.4
  Return $\boldsymbol{Z}$

Algorithm 4 *Reqe* training algorithm (multiple aligned graphs)

Input: A set of aligned graphs $\mathcal{G} = \{G_1, G_2, \cdots, G_M\}$ on V nodes, structural properties in each graph $\{t_1, t_2, \cdots, t_M\}$, aggregated adjacency matrix $\boldsymbol{A}$, no. of iterations I, stationary dist. feature size K, vector dimension d, jumping probability α, negative sample size m, neighbor sample size n.

Output: latent node embeddings $\boldsymbol{Z} \in \mathbb{R}^{V \times d}$

procedure MODEL TRAINING
  for i $\in [1, \cdots, M]$ **do**
    $\boldsymbol{S}_{\mathrm{i}} \leftarrow \mathrm{Stat\_dist}(G_{\mathrm{i}}, K, \alpha)$ *compute stationary distribution features for graph* G_{i}
    $\boldsymbol{S}_{\mathrm{i}} \leftarrow [\boldsymbol{S}_{\mathrm{i}}, t_{\mathrm{i}}]$
  $\boldsymbol{S} = AVG\{S_1, \cdots, S_M\}$ *compute average node structural properties*
  $\boldsymbol{D} \leftarrow 1 - \mathrm{cos\_sim}(\boldsymbol{S})$
  Initialize $\boldsymbol{Z} \sim \mathcal{N}(\boldsymbol{S}, \sigma^2)$
  while $iter \leq I$ **do**
    for $v \in V$ **do**
      Sample m negative nodes $u' \sim \boldsymbol{D}_v$
      Sample n neighbors $u \sim \boldsymbol{A}_v$
    Update $\boldsymbol{Z}$ according to Eq. 4.4
  Return $\boldsymbol{Z}$

4.4 Related work

Most of the existing node embedding methods (e.g. LINE [1], DeepWalk[3], Node2vec[2]) consider (local) structural equivalence by preserving node proximity. However, these methods fail to capture the structurally similar nodes that are far apart in the graph, thus the embedding they produce doesn't reflect the structural role of the nodes in general.

There has been a few node embedding methods designed for capturing the general notion of structural equivalence (as opposed to positional proximity), or more precisely, automorphic equivalence, of which we choose some representative methods and summarized their contributions as follows.

Struc2vec [61] constructs a multi-layer graph to encode structural similarities at different scales. The method is powerful in encoding local neighborhood features for each node yet very expensive because of the construction of multi-layer graph. There has been other similar works (e.g. subgraph2vec [62]), but struc2vec is arguably better at preserving structural equivalence, so we choose Struc2vec as a representative work relying on node degree at distance 1 to k as structural properties. Another model subgraph2vec [62] adopted similar idea of measuring pairwise structural similarity, but the notion of structural equivalence is very rigid, as it is defined as a binary property dictated by the Weisfeiler-Lehman isomorphism test [53]. As struc2vec is arguably better than subgraph2vec for preserving structural equivalence, we choose Struc2vec as a representative work which relies on node degree at distance 1 to k and local neighborhood as structural properties.

GraphWave [63] learns structural node embedding based on the diffusion of a spectral graph wavelet centered at the node. This way the structural information is contained in how the diffusion spreads over the network. The wavelets are treated as probability distributions over the graph, and are then embedded using the empirical characteristic function.

DRNE [52] proposes to capture regular equivalence by introducing a recurrent neural network for neighborhood aggregation, and use node degree for regularization. DRNE achieved state-of-the-art performance on multiple *structural* node classification tasks.

SPINE [64] assigns a structural feature vector to each node based on Rooted PageRank, and structural identities are incorporated to jointly preserve local proximity and global proximity of the network simultaneously in the learned embeddings.

Role2vec [65] uses small sub-graphs (i.e. motifs) as structural features, and adopted attributed random walk to encode motifs to capture structural roles. HODE [66] also makes use of motifs as structural features.

As we can see from the descriptions, all of the above methods are based on a particular structural property (e.g. degree) or structural identity indicator (e.g. graph wavelet, sug-graph patterns), thus they are all able to capture automorphic equivalence that's encoded in the property used in the model.

In addition, **RolX** [51] is designed to *explicitly* identify the role of nodes using network structures. This unsupervised approach is based on enumerating various recursive structural features extracted with ReFeX [67] and factorizes the binary node-feature matrix to create low dimensional node embedding. In our experiments, we use RolX as an approximation of adopting discrete equivalences for node classification tasks.

For the multiple graph case, since there are no existing node embedding works that specifically aim to capture structural equivalence in multiple graphs, we choose two other baselines: weighted stochastic blockmodel (**SBM**) [68], and a representative *unsupervised* temporal graph embedding model **DynamicTriad** [69] which is optimized to learn node evolution.

In recent years, there has been work on theoretical investigations of social networks and the accompanying equivalences, including axioms and logical characterizations [24] [25], spectral methods [26], and block models [27].

Another related area of research is on Graph neural networks (GNNs) ([8], [16], [17], [70]), which focus on producing node representations for graph prediction tasks. Unlike node embedding models, most GNNs are designed for semi-supervised learning tasks with partially-labeled, attributed graphs. The unsupervised variant of GraphSAGE [17] uses a random walk loss (similar to Node2vec [2] and DeepWalk [3]) that optimizes node proximity rather than structural equivalence, and this limits their expressivity. We adopt a similar idea of neighborhood sampling and feature aggreagtor to preserve role similarity. However, in GraphSAGE the aggregated neighborhood features of node v_{i} is used for v_{i}'s embedding, whereas in our work the aggregated neighborhood of v_{i} is used in the loss, which indirectly updates the embedding of v_{i} based on the similarity of v_{i}'s neighbors' roles. Moreover, the loss function in *Reqe* mimics the recursive role equivalence and adopt structural property-guided negative sampling and regularization. The empirical study in Section 4.6.7 shows that GraphSAGE does not perform well in preserving automorphic equivalence compared to our proposed model *Reqe*, especially when no structural properties are used as input.

4.5 Expressive power of *Reqe*

In this section we analyze the expressive power of the proposed model *Reqe* , for both the single graph and multiple graph cases. Please find all the proofs in the appendix.

Theorem 4.5.1 (Representing structural equivalence in embeddings)**.** *Given a graph G and a set of structural properties $\mathcal{T}$, the embedding $\boldsymbol{Z}$ generated by* Reqe *is an approximation of automorphic equivalence in the original graph G. More specifically,*

1. *Any automorphically equivalent nodes should have same node embeddings, i.e. $u \equiv v \implies \boldsymbol{z}_u = \boldsymbol{z}_v$.*

2. *While there is no guarantee that $\boldsymbol{z}_u = \boldsymbol{z}_v \implies u \equiv v$, the more structural properties we have, the closer the embedding is to automorphic equivalence in the original graph G.*

The arguments of expressive power w.r.t structural properties in a single graph also hold for the *multiple graph* case when there are enough graph samples.

Theorem 4.5.2 (Representing structural equivalence in multiple graphs)**.** *For the multiple graph case, with enough graph samples and complete structural properties ($\mathcal{T}_c$), the embedding generated by* Reqe *is an approximation of automorphic equivalence in the original graphs.*

Theorem 4.5.3 (Boosted expressiveness through recursive steps)**.** *Given a fixed set of structural properties $\mathcal{T}$,* Reqe *is able to capture more longer-range properties faster by performing the recursive steps (i.e. updating node embedding by comparing neighbors' embedding). Specifically, given a fixed set of structural properties capturing node isomorphism within d-hop neighborhood, performing k recursive steps will enlarge the radius to $k \cdot d$, which essentially captures more structural properties and thus improve the accuracy of the embedding (according to Theorem 4.5.1(B)).*

According to Theorem 4.5.1 and Theorem 4.5.3, for any embedding method trying to capture automorphic equivalence in the graph, the expressive power depends on (1) the set of structural properties explicitly considered, which determines the value d (diameter of structural properties), (2) if and how the recursion is performed, which determines value

k (recursive steps). Table 4.1 shows the characterization of *Reqe* and existing methods in terms of range of properties and recursive steps. Our model can utilize any long-range structural properties with large d (as large as graph diameter, e.g. eigenvalue centrality) and the recursive embedding framework allows for large number of k (i.e. number of epochs) as opposed to the few recursive steps in Struc2vec or RolX, where k is typically a small number like three or five.

Table 4.1. Characterization of existing methods

Method	Local properties (small d)	Long-range properties (large d)	Recursive features (small k)	Recursive embedding (large k)	Handle aligned graphs
Struc2vec	✓		✓		
Subgraph2vec	✓		✓		
GraphWave	✓	✓			
SPINE	✓	✓			
role2vec	✓				
RolX	✓		✓		
DRNE	✓			✓	
Reqe	✓	✓		✓	✓

4.6 Experiments

In this section, we assess the model performance on three tasks, specifically whether the learned embedding can: (A) encode structural/regular equivalences, (B) be used to predict other structural properties, and (C) improve node classification or role discovery. The first two tasks aim to evaluate if our proposed method is effective in preserving structural properties, and the third task is used to evaluate if the learned embeddings are useful for real-world prediction tasks.

4.6.1 Datasets

We use graphs from various application areas, some with role related node labels. The dataset statistics are listed in Table 4.2 and the detailed descriptions with references are as follows.

- **Facebook-large** [71] is an extracted social network from Facebook where edges represent friendship.

- **Cora** and **Pubmed** [16] are citation networks with nodes representing publications and edges representing citation relation.

- **European** and **American airports** networks [61]. The two air-traffic networks are unweighted, undirected networks where nodes represent airports and edges indicate commercial flights. For each airport, the label reflects the activity level measured by the total number of landings plus takeoffs or the total number of people passing the airports. In particular, the quartiles are obtained from the empirical activity distribution to split the dataset into four groups, with label 0 given to the 25% least active airports, and so on. Classes are related to the role played by the airport in the transportation system.

- **Protein** graph. The protein graph is obtained from a collection of proteins [72] with ground-truth labels. Each node is labeled with a functional role of the protein.

- **Student interaction** graphs [73]. This dataset contains student interaction graphs from on-campus co-visitation behaviour at different locations. The nodes are freshmen at a public university, and edges represent the two students have co-visited at a certain venue. Four graphs are for co-visitation at four different venues, i.e. academic buildings, gym, dining halls, residence halls. We also aggregated all the edges from the four graphs to generate another aggregated graph. Each node is attributed with major and GPA. The predicted label is whether the student is going to dropout or not. Note that dropout behavior has been shown to correlate students social behavior [74].

- **DBLP** and **Facebook (temporal)** [75]. DBLP has 18 snapshots of co-authorship social networks, with label representing research area. Facebook has 55 snapshots with label indicating political view.

We generate a mirrored copy for the graphs with no node labels in tasks (A) and (B), and use the graphs with ground-truth role-correlated node labels in task (C).

Generating mirrored graphs. For each of the unlabeled graphs, e.g. Facebook, Cora and Pubmed graphs, we create a mirrored version. The mirrored graph is created by taking two copies of the original graph and adding random edges with probability 0.2 connecting

Table 4.2. Dataset statistics

Graph(s)	# Nodes	# Edges	Density	# Classes
Cora	5,416	11,140	3.79e-4	N/A
Facebook-large	15,248	56,388	2.42e-4	N/A
Pubmed	39,434	92,536	5.95e-5	N/A
Karate	68	162	3.50e-02	N/A
Jazz	396	5530	3.52e-02	N/A
Facebook-small	448	6421	3.20e-02	N/A
Proteins	504	1,788	7.05e-3	2
European airports	399	5,995	3.76e-2	4
American airports	1,190	13,599	9.60e-3	4
Student co-visitation	6,306	2,360,902	1.48e-2	2/5
Facebook (temporal)	2,716	22,712	3.18e-3	2
DBLP (temporal)	17,191	318,735	1.08e-3	2

mirrored nodes from different copies. In this way the two mirrored nodes are guaranteed to be isomorphic, i.e. automorphic equivalent.

Class labels. The six labeled graphs (including temporal graphs) have role-related node labels. Each node in the **Protein** graph is labeled with a functional role of the protein. For the two **airports** graphs, the node label reflect the airport activity level measured by the total number of landings plus takeoffs. For the **student** co-visitation graph, the predicted label is whether the student is going to dropout or not, which has been shown to correlate students social behavior [74]. We also have a temporal version of co-visitation graph including four monthly slices to test *Reqe* with multiple graphs. For the two other temporal graphs, **DBLP** contains 18 co-authorship graphs with class label of authors' research field, and **Facebook** contains 55 social network graphs with class label of political views.

4.6.2 Methodology

We compare our model *Reqe*with the five structural graph embedding models discussed in Section 4.4, plus a representative general node embedding method Node2vec [2] for task (A) and (B), and then selected the best performing models for task (C) on real-world datasets. In addition, to compare to methods that aim to identify structural roles, we add RolX for

Table 4.3. Mean and standard deviation for the distance between node pairs in the node embedding, with % reduction from all pairs to isomorphic pairs. OOM means "out of memory" error. Bold numbers represent the best performing model in each column. *Reqe*-2 uses two properties: degree and stationary distribution; *Reqe*-3 uses degree, stationary distribution and closeness centrality; *Reqe*-5 uses degree, stationary distribution, betweenness centrality, closeness centrality and kcore centrality.

		Cora			Facebook			Pubmed		
	Pairs	mean (std)	% reduc.	Time	mean (std)	% reduc.	Time	mean (std)	% reduc.	Time
node2vec	all	2.9897 (0.9173)	14.76%	0.66	3.6899 (1.0021)	-4.94%	2.36	3.7923 (0.3040)	-0.44%	5.46
	mirrored	2.5484 (0.9792)			3.8721 (0.5539)			3.8091 (0.3031)		
SPINE	all	0.0254 (0.0174)	-0.68%	50.12	5.2606 (1.2000)	-6.02%	264.44	OOM		
	mirrored	0.0256 (0.0174)			5.5771 (0.9042)					
GraphWave	all	0.0277 (0.0223)	-0.39%	2.70	0.0121 (0.0108)	0.35%	22.77	OOM		
	mirrored	0.0278 (0.0224)			0.0120 (0.0108)					
role2vec	all	3.1659 (0.5119)	20.57%	53.63	4.4366 (0.5789))	26.81%	553.89	4.9195 (0.5434)	48.05%	657.96
	mirrored	2.5146 (0.2741)			3.2470 (0.8149)			2.5573 (0.7599)		
DRNE	all	0.3475 (0.3318)	22.81%	3.36	0.5589 (0.4260)	49.47%	15.46	0.3956 (0.2840)	41.05%	42.40
	mirrored	0.2682 (0.1035)			0.2824 (0.1306)			0.2332 (0.1456)		
struc2vec	all	4.5630 (1.3408)	58.37%	21.48	5.0953 (1.5871)	63.84%	129.18	4.8015 (1.9335)	64.90%	386.14
	mirrored	1.8996 (0.4243)			1.8606 (0.4713)			1.6855 (0.4534)		
Reqe- 2	all	4.4020 (2.4936)	51.14%	4.29	9.3130 (6.2441)	73.84%	20.52	6.5036 (4.1128)	64.84%	43.66
	mirrored	2.1508 (1.3513)			2.4367 (1.7901)			2.4166 (1.6553)		
Reqe- 3	all	3.7756 (2.4412)	68.29%	6.47	10.2116 (6.5888)	**74.42%**	47.69	6.8964 (3.9293)	**68.41%**	226.88
	mirrored	1.1973 (0.4887)			2.6117 (2.2652)			2.3168 (1.5999)		
Reqe- 5	all	2.2862 (1.9818)	**75.34%**	9.35	6.4557 (4.2209)	70.01%	83.26	7.0423 (3.5943)	66.90%	298.15
	mirrored	0.5639 (0.1908)			1.9360 (1.2280)			2.3312 (1.4522)		

comparison in task (C). For all the tasks , the models output d-dimensional node embeddings $\boldsymbol{Z} = \boldsymbol{z}_1 \cdots \boldsymbol{z}_V$ for any input graph with V nodes, where $\boldsymbol{z}_{\mathrm{i}} \in \mathbb{R}^d$, d = 128.

For the node classification task (Task (C)), 60% of the nodes are used for training, 20% for validation, and the remaining 20% for testing. We do 100 trials randomizing the split, and report the mean and standard deviation of classification accuracy. As the node label is highly unbalanced in the student co-visitation graph, we report balanced accuracy instead. For the datasets containing multiple graphs, we either apply methods that are capable of modeling multiple graphs or a weight graph (i.e. *Reqe*, SBM and DynamicTriad), or merge the graphs into a single graph for other comparison methods (e.g. struc2vec).

4.6.3 Task (A): Equivalence encoding

In this task, we embed the whole mirrored graph, and compute the distance between the isomorphic mirrored nodes and all pair of nodes. An ideal node embedding model will

embed the mirrored nodes close to each other while making other nodes distributed in the embedding space.

The results are shown in Table 4.3. For each method, we also computed the percentage of reduction from all pairs to isomorphic pairs, which should be close to 100% for an ideal model. We can see that both node2vec and SPINE are not able to preserve node isomorphism, where the reduction generally goes the opposite way. This is because the two models tend to embed the connected nodes together, while the isomorphic nodes in these graphs usually reside in different parts. Graphwave achieves positive reduction rate on one graph but fails on the other. Both SPINE and Graphwave run out of memory on the largest dataset. The other models, role2vec, struc2vec, DRNE and *Reqe* all successfully embed the mirrored nodes close(r) in the embedding space with a positive reduction rate. Among these models, our model consistently excels in the task with high reduction rates across datasets and struc2vec is the strongest baseline.

To further evaluate the impact of the number of structural properties encoded, we examined three variants of our model using different numbers of structural properties. When more than two properties are provided, they are used to initialize the embedding and also used in computing the dissimilarity matrix. Comparing the three variants, we can see that including more properties in general helps the model project the mirrored nodes closer to each other while keeping a distributed graph embedding. For example, comparing *Reqe*-2 and *Reqe*-5 on Cora graph, the reduction rate increases from 51.14% to 75.34%.

We also conducted additional experiments on three small-scale graphs (Karate, Jazz and Facebook-small, each with < 500 nodes), and the percentages of reduction is shown in Figure 4.2. Comparing the results on the small-scale graphs and the large-scale graphs (in Table 4.3), *Reqe* out-performed all other methods by a larger margin on the large-scale graphs. Note that the three larger graphs are much sparser than the small graphs (see the densities in Table 4.2), therefore it is more difficult to capture the role information with limited signals. The node degree, for example, is not very useful in differentiating node roles, which causes DRNE [52] to perform much worse on these datasets.

Runtime As shown in Table 4.3, *Reqe* (best-performing variant) runs in 10 minutes on the Cora graph with 5416 nodes, and less than 4 hours on Pubmed graph with 39K nodes, while

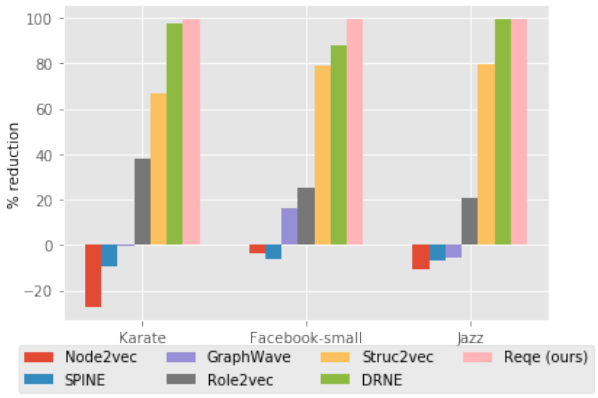


Figure 4.2. % reduction on three small graphs

Table 4.4. MSE on predicting structural properties: clustering coefficient (cc) and second-order degree centrality (sod)

	Cora		Facebook		Pubmed	
	cc	sod	cc	sod	cc	sod
node2vec	0.0555	0.0447	0.0844	0.0053	0.0362	0.0059
SPINE	1276	12527	0.0697	0.0027	OOM	
GraphWave	0.0981	0.0045	0.0837	0.0122	OOM	
role2vec	0.0895	0.0040	0.0759	0.0035	0.0242	0.0041
DRNE	0.0782	0.0037	**0.0597**	0.0008	0.0246	0.0038
struc2vec	**0.0351**	0.0029	0.0625	0.0006	**0.0234**	0.0035
Reqe-2	0.0630	0.0030	0.0618	0.0009	0.0292	0.0031
Reqe-3	0.0590	0.0018	0.0599	**0.0005**	0.0258	**0.0023**
Reqe-5	0.0452	**0.0005**	0.0606	0.0010	0.0278	0.0028
Reqe-3 (cc)	2e-08	0.0042	3e-08	0.0016	2.5e-08	0.0037

our main competitor stuc2vec[61] takes 21 minutes and 6.5 hours respectively. DRNE is slightly more efficient than *Reqe* by only considering the node degrees but the performance is significantly worse in all the tasks. Therefore, *Reqe* is more powerful than previous methods, without compromising efficiency.

4.6.4 Task (B): Structural property prediction

In this task, we test if the node embedding generated by the above methods preserves structural properties that weren't explicitly used in the model. Here we use clustering coefficient and second-order degree centrality as examples.

Table 4.5. Accuracy of node classification task on various datasets. Bold numbers represent the best performing model in each column.

	Europe airports	Protein	American airports
centrality	0.538 (0.054)	0.615 (0.046)	0.590 (0.032)
role2vec	0.294 (0.044)	0.682 (0.043)	0.448 (0.028)
DRNE	0.547 (0.056)	0.692 (0.051)	0.566 (0.032)
struc2vec	0.579 (0.050)	0.853 (0.036)	0.602 (0.027)
Reqe	**0.604 (0.047)**	**0.906 (0.027)**	**0.645 (0.026)**
RolX	0.425 (0.052)	0.568 (0.045)	0.335 (0.027)
RolX-disc	0.328 (0.044)	0.517 (0.044)	0.329 (0.026)
Reqe-disc	0.515 (0.048)	0.627 (0.048)	0.554 (0.028)

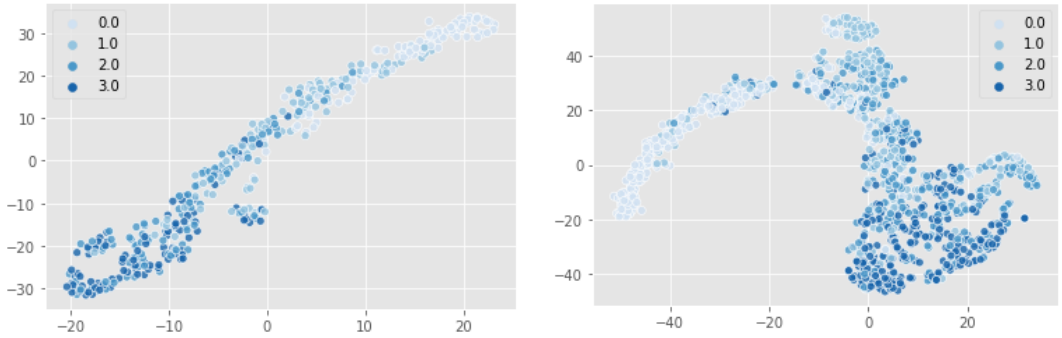
Table 4.4 shows the performance (mean squared error) of predicting structural properties. We can see that our method consistently achieves the best performance on both properties across three datasets, with the only three exceptions on clustering coefficient where struc2vec or DRNE perform better with a small margin. Again, including more properties in general helps with prediction as they provide more information about structural role. We also tried adding clustering coefficient in the model (denoted as *Reqe*-3(cc)) to show that it is able to preserve this same property with a e-08 mean squared error for proof of correctness.

4.6.5 Task (C): Node classification

On single graph Table 4.5 shows the node classification accuracy on American/European airports nd Protein graph. Centrality features are the initial embedding used in our method, which is a combination of degree centrality, closeness centrality and betweeness centrality and k-core. We can see that our model achieves the best results on all these datasets, and the struc2vec model is the second best. Note that centrality feature is correlated with the predicted labels with a better performance than role2vec, but our model is able to further improve it by including negative sampling loss and neighborhood aggregation.

In addition, we implemented two discrete models, i.e. RolX-disc and *Reqe*-disc to represent the discrete equivalences. RolX-disc is based on the original clustering results of RolX, and replace the distribution of cluster with assigning the class with highest probability. *Reqe*-disc uses K-means[76] to generate c (i.e. number of class) clusters on the node embedding produced

by *Reqe*. We use one-hot encoded feature for both two discrete models and perform the same supervised node classification task. Note that both two methods perform worse than all the node embedding models, which validates our claim that the discrete nature of equivalences limits its application. Moreover, *Reqe*-disc still out-performs RolX/RolX-disc, indicating that the embedding produced by *Reqe* contains more structural information than the features extracted by RolX.



(a) Europe airports embedding (b) American airports embedding

Figure 4.3. t-SNE visulization of the node embeddings

Figure 4.3 shows the visualization of the learned embeddings on airport graphs, where we project the $d = 128$ dimensions into 2D using t-SNE [47]. The clear cluster/spectrum pattern shows that the generated embeddings successfully capture the structural role information reflected by the label (quartiles).

For the student interaction graph, the task is to predict whether a student is going to dropout based on her interaction with other students. As the label is highly unbalanced (dropout rate is around 6%), we do a three-fold stratified sampling to split the training/testing data for cross validation, and report balanced accuracy and accuracy@k instead. Specifically, we rank the predicted probability of dropping out for all the students and generate a top-k list, then we determine how many true dropout students appear in the top-k list (which is defined as a 'hit'). The accuracy@k is defined as the ratio of hits to k.

Since GPA is highly correlated with dropout behavior, we tried including GPA as an additional node attribute in our model (denoted as *Reqe*-GPA). Specifically, in addition to the three structural properties (degree, stationary distribution and betweenness centrality) in

Table 4.6. Accuracy@k and balanced accuracy of detecting dropout students on student interaction graphs.

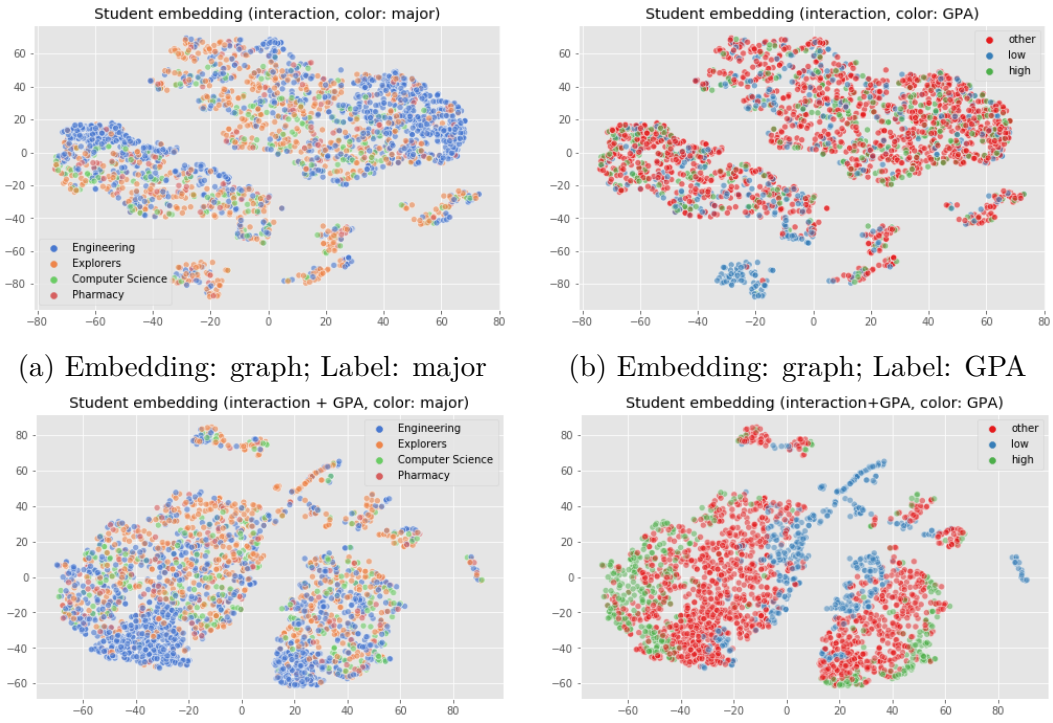
Model	Acc@10	Acc@15	Acc@30	Acc@50	Acc
struc2vec	0.0667	0.0667	0.0556	0.0667	0.5197
role2vec	0.1667	0.1111	0.1111	0.0800	0.5189
DRNE	0.1333	0.1333	0.2000	0.2000	0.6343
Reqe	**0.2667**	**0.2667**	**0.2667**	**0.2133**	**0.6387**
Reqe-GPA	**0.4333**	**0.4000**	**0.3556**	**0.3133**	**0.6596**

$\mathcal{T}$, now we use GPA, a node attribute with continuous value between 0 and 4, as t_4 in the set $\mathcal{T}$. Note that other methods cannot directly take node features as input.

Results of accuracy@k is shown in Table 4.6, from which we can see that our model performs much better than all other models on both accuracy@k and balanced accuracy, with DRNE performs the second best. Even without incorporating GPA, our model already out-performs all the other competing methods significantly. Moreover, including GPA in learning the embedding further improves our model performance, e.g. from 0.2667 to 0.4333 on accuracy@10, and from 0.6387 to 0.6596 on balanced accuracy.

In Figure 4.4 we show the 2D plots of student embedding produced by our model with and without GPA included in the model input. The nodes are a subset of students from the top four majors, i.e. engineering, explorers (major not decided), computer science and pharmacy. We can easily identity two clusters of engineering students Figure 4.4a. Notably in Figure 4.4b there is a small cluster of low-GPA students (around (-20, -80)) from various majors, which means even without including GPA explicitly, the node embedding is able to capture GPA information. When GPA is included in learning the embedding (Figure 4.4c-4.4d), we can see that GPA information has a large impact on the embedding in 4.4d, where we can see two continuous clusters containing low-other-high parts. While at the same time, the clusters of engineering students are still captured in 4.4c.

On multiple graphs Figure 4.5 shows the node classification accuracy on synthetic graphs for *Reqe* and weighted SBM. The graphs are generated with a SBM under various noise rate (i.e. ratio of random generated edges). The node labels are the block number used in the SBM. We can see both *Reqe* and weighted SBM can achieve nearly 100% accuracy



(a) Embedding: graph; Label: major (b) Embedding: graph; Label: GPA

(c) Embedding: graph+GPA; Label: major (d) Embedding: graph+GPA; Label: GPA

Figure 4.4. t-SNE visulization of the generated node embedding on student interaction graph

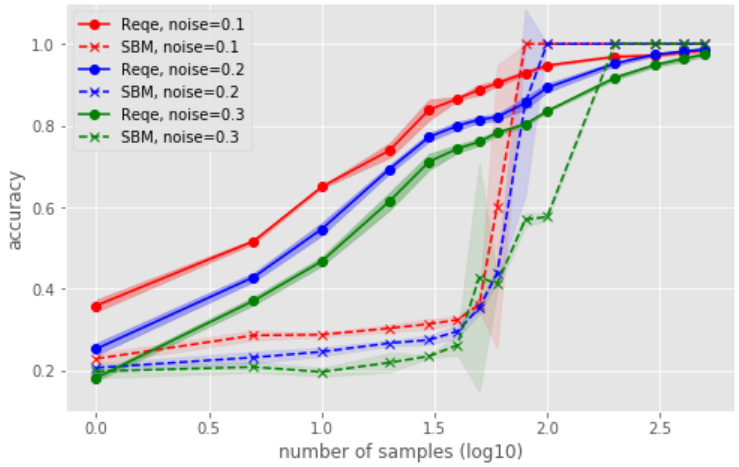


Figure 4.5. Node classification accuracy with multiple graphs (synthetic data) varying sample size

given enough samples, but weighted SBM requires more than 100 samples to achieve good performance, whereas *Reqe* perform relatively well (> 0.7 accuracy) even with 30 samples.

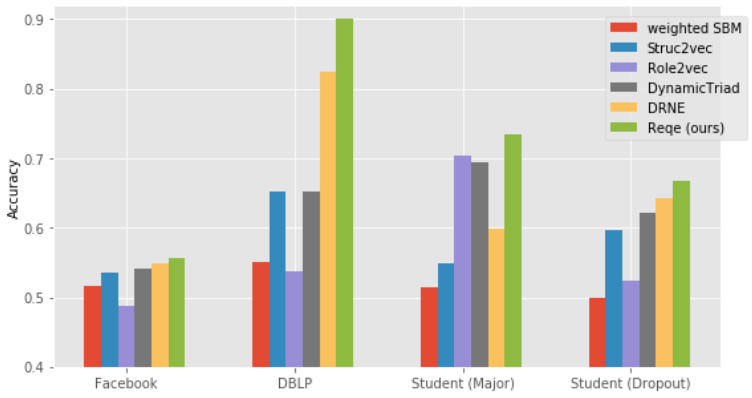


Figure 4.6. Node classification performance with multiple graphs (real-world data)

This shows the advantage of our model especially with small number of graphs – a more common real-world scenario.

Figure 4.6 shows the performance of various embedding methods and weighted SBM on three real-world datasets. Previous work [75] showed that the model performance on DBLP and Facebook data was stable when shuffling time steps, thus we can assume the graphs are sampled from a latent graph distribution function ($\boldsymbol{A}^*$) without temporal dependency. We can see that *Reqe* consistently out-performed all other baselines on all five tasks, with the largest margin of 9% on DBLP — the largest dataset.

4.6.6 Parameter sensitivity

The weighting factor α and λ are two hyper-parameters, which can be decided from a grid search with validation set. Figure 4.7 shows the node classification accuracy on American airport when varying the two parameters, from which we can see that the performance is more sensitive to α than λ, which means the degree-guided regularizer is less important than other structural properties used for negative sampling.

4.6.7 Comparison with GraphSAGE

We choose GraphSAGE[17] as a representative GNN method and empirically evaluate its ability in capturing automorphic equivalence in graphs. The original version (denoted as 'GS-nofeat') for unattributed graphs explicitly learns node embeddings with proximity-based

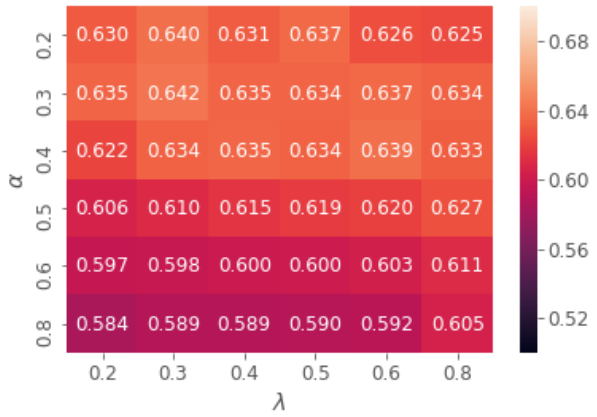


Figure 4.7. Parameter sensitivity w.r.t weighting factors

loss. For a fair comparison, we also implement an augmented version (denoted as 'GS-feat') to use the same node structural properties that *Reqe* uses as input features. We use the same model architecture as described in the original paper, e.g. two GNN layers, LSTM aggregator. In Table 4.7-4.8 we show the performance of the two GraphSAGE versions and *Reqe* for tasks (A) and (C).

We can see that the GS-nofeat version didn't perform well on the two tasks, which is not surprising given the recent theoretical results ([6], [77], [78]) showing that GNNs are not most expressive. This limitation is more obvious when no node features are used. The augmented GS-feat version improves performance by a large margin via the added node feature that record structural properties. This demonstrates the finding in Theorem 4.5.1, that adding more node properties captures automorphic equivalence in the graph more accurately. Moreover, our proposed model *Reqe* achieves better performance than both GraphSage versions, which illustrates the advantage of using a role-based loss function as discussed in Section 4.4.

4.7 Conclusion

In this work, we consider the problem of identifying structural roles in single graphs and multiple graphs using node embedding techniques. First, we formally define the notion of *Representation-based equivalence* w.r.t a set of structural properties for both cases, which

Table 4.7. Performance of GraphSAGE and *Reqe*-3 (the best performing *Reqe* variant) for task (A): mean and standard deviation for the distance between node pairs in the node embedding, with % reduction from all pairs to isomorphic pairs. The results for *Reqe* are copied from Table 4.3.

		Cora		Facebook		Pubmed	
	Pairs	mean (std)	% reduc.	mean (std)	% reduc.	mean (std)	% reduc.
GS-nofeat	all	1.4098 (0.1100)	3.51%	1.4009 (0.1292)	6.24%	1.3931 (0.1445)	10.67%
	mirrored	1.3604 (0.1315)		1.3136 (0.1406)		1.2444 (0.1582)	
GS-feat	all	1.3955 (0.2220)	54.17%	1.3901 (0.2084)	45.91%	1.3789 (0.2184)	51.26%
	mirrored	0.6395 (0.3393)		0.7520 (0.3401)		0.6671 (0.2958)	
Reqe-3	all	3.7756 (2.4412)	68.29%	10.2116 (6.5888)	74.42%	6.8964 (3.9293)	68.41%
	mirrored	1.1973 (0.4887)		2.6117 (2.2652)		2.3168 (1.5999)	

Table 4.8. Performance of GraphSAGE and *Reqe* for task (C): accuracy of node classification task on various datasets. The results for *Reqe* are copied from Table 4.5

	Europe airports	Protein	American airports
GS-nofeat	0.311 (0.047)	0.577 (0.044)	0.410 (0.030)
GS-feat	0.439 (0.045)	0.751 (0.044)	0.470 (0.028)
Reqe	0.604 (0.047)	0.906 (0.027)	0.645 (0.026)

attempts to preserve automorphic equivalence in the original graph(s) with learned node embedding. This leverages the advantage of distributed representation for real-world prediction tasks by preserving more structural (similarity) information than discrete roles. Second, we proposed a flexible framework *Reqe* which takes any number of structural properties, and produce node embeddings that reflect the structural role of each node. We further analyze the expressive power of our model *Reqe*, showing that our model is demonstrably better at preserving automorphic equivalence compared to other node embedding methods by incorporating any structural properties and recursive steps. Moreover, to the best of our knowledge our approach is the first method to learning structural embeddings over multiple graphs. Our empirical evaluation considers a variety of graphs and tasks. Results show that our model is effective for preserving regular equivalence, and also useful for solving real-world node classification/role discovery tasks.

4.8 Appendix: Proof of Theorems

4.8.1 Proof for Theorem 4.5.1

We restate the theorem for completeness.

Theorem 4.5.1 (Representing structural equivalence in embeddings)**.** *Given a graph G and a set of structural properties $\mathcal{T}$, the embedding $\boldsymbol{Z}$ generated by* Reqe *is an approximation of automorphic equivalence in the original graph G. More specifically,*

1. *Any automorphically equivalent nodes should have same node embeddings, i.e. $u \equiv v \implies \boldsymbol{z}_u = \boldsymbol{z}_v$.*
2. *While there is no guarantee that $\boldsymbol{z}_u = \boldsymbol{z}_v \implies u \equiv v$, the more structural properties we have, the closer the embedding is to automorphic equivalence in the original graph G.*

Proof. We show that for node embedding generated by *Reqe*:

1. Any automorphically equivalent nodes should have same node embeddings, i.e. $u \equiv v \implies \boldsymbol{z}_u = \boldsymbol{z}_v$.

2. While there is no guarantee that $\boldsymbol{z}_u = \boldsymbol{z}_v \implies u \equiv v$, the more structural properties we have, the closer the embedding is to automorphic equivalence in the original graph. More precisely, suppose the set S includes all pairs of non-automorphic equivalent nodes, i.e. $\mathcal{S} = \{(u, v) | u \not\equiv v, \forall u, v \in \mathcal{V}\}$, and an embedding $\boldsymbol{Z}$ can differentiate a subset of pairs, i.e. $\mathcal{S}_d(\boldsymbol{Z}) = \{(u, v) | u \not\equiv v \wedge \boldsymbol{z}_u \neq \boldsymbol{z}_v, \forall u, v \in \mathcal{V}\} (\mathcal{S}_d \subseteq \mathcal{S})$. Then for two embedding $\boldsymbol{Z}_a$ (encoding structural properties $\mathcal{T}_a$) and $\boldsymbol{Z}_b$ (encoding structural properties $\mathcal{T}_b$), if $\mathcal{T}_a \subset \mathcal{T}_b \subset \mathcal{T}_C$ (complete structural properties), then $\mathcal{S}_d(\boldsymbol{Z}_a) \subset \mathcal{S}_d(\boldsymbol{Z}_b) \subset \mathcal{S}$, which means there exists at least one pair of non-automorhpic equivalent nodes that $\boldsymbol{Z}_a$ can differentiate while $\boldsymbol{Z}_b$ cannot. Therefore, $\boldsymbol{Z}_a$ is more accurate in capturing automorphic equivalence in the original graph.

Given a graph G, suppose we have learned a representation matrix $\boldsymbol{Z} \in \mathcal{R}^{n \times d}$ that encodes a set of structural properties $\mathcal{T}_Z = \{t_1, t_2, \cdots, t_k\}$, which is a subset of *complete* structural properties $\mathcal{T}_C = \{t_1, t_2, \cdots, t_k, t_{k+1}, \cdots, t_m\}$.

(A): By definition of structural properties, for two nodes u and v that are automorphically equivalent, we have $\forall \mathrm{i}, 1 \leq \mathrm{i} \leq k, t_{\mathrm{i}}(u) = t_{\mathrm{i}}(v)$. Therefore, the initial embeddings (sampled using structural properties) of u and v are equal with ϵ error due to randomness in sampling. As their dissimilarity of structural properties is zero, they will never be selected as negative samples during training, and the (multi)set of their neighbors embeddings should also be the same (see Definition 4.2.3), the automorphically equivalent nodes u and v should get the same final embedding.

(B): If there exists a structural property $t_{\mathrm{j}} \in \mathcal{T}_C \setminus \mathcal{T}_Z$ that the current embedding $\boldsymbol{Z}$ fails to encode and $t_{\mathrm{j}}(u) \neq t_{\mathrm{j}}(v)$, which will lead to $\boldsymbol{D}(\boldsymbol{z}_u, \boldsymbol{z}_v) > \epsilon$, then we add the property t_{j} into the set $\mathcal{T}_Z$ and retrain the model to update the embedding. As u and v now have different structural features by including t_{j}, they have a certain chance to be sampled as negative samples, and also their neighbors embedding might be different so that they get different embeddings via recursion. In this way, we successfully differentiate the non-automorphic pair u and v. Repeating this procedure, we can make the embedding more accurate in capturing automorphic equivalence.

□

4.8.2 Proof for Theorem 4.5.2

Theorem 4.5.2 (Representing structural equivalence in multiple graphs)**.** *For the multiple graph case, with enough graph samples and complete structural properties ($\mathcal{T}_c$), the embedding generated by* Reqe *is an approximation of automorphic equivalence in the original graphs.*

Proof. We show that for node embedding generated by *Reqe*, in the limit (1). automorphically equivalent nodes are embedded close and (2). non-equivalent nodes get different embeddings with high probability.

(1). For automorphically equivalent nodes u and v, we have $\forall t \in \mathcal{T}_c, t^*(u) \stackrel{d}{=} t^*(v)$. As the average structural properties $\bar{t}$ computed from the graph samples is an unbiased estimator of true average of t^*, then the two nodes have same initial embedding. According to the definition of node equivalence, the neighbors of the two nodes are also regular equivalent, therefore, during the model training, the multiset of neighbors' embeddings are also the same.

(2). For non-equivalent nodes u and v, they get different embeddings with high probability.

- The more structural properties we have, the more non-equivalent pairs we can differentiate. Two non-equivalent nodes may have same joint distribution of c probabilistic structural properties ($c < m$), but there must exist another t^*_{c+1} that can break the equivalence since they're not automphically equivalent.

- The more samples we have, the more pairs we can differentiate. Two non-isomorphic nodes might have same estimated average of structural properties with k samples, but adding more samples the estimated value of $\hat{t}_{\mathrm{i}}$ is closer to true $E[t^*_{\mathrm{i}}]$, and at least one $E[t^*_{\mathrm{i}}]$ can differentiate them.

- Worst case: only one structural property has different expected values for one non-equivalent node pairs. Then during the model training when we consider condition (2) in def. 2.10, the multiset of neighbors' embeddings reflect the distribution of neighbors' structural properties. Since at least one non-equivalent pair of nodes have different initial embeddings, the difference can propagate to their neighbors and higher-order neighbors, which breaks the equivalence for all nodes with D steps where D is the graph diameter.

□

4.8.3 Proof for Theorem 4.5.3: Boosted expressiveness through recursive steps

Theorem 4.5.3 (Boosted expressiveness through recursive steps)**.** *Given a fixed set of structural properties $\mathcal{T}$,* Reqe *is able to capture more longer-range properties faster by performing the recursive steps (i.e. updating node embedding by comparing neighbors' embedding). Specifically, given a fixed set of structural properties capturing node isomorphism within d-hop neighborhood, performing k recursive steps will enlarge the radius to $k \cdot d$, which essentially captures more structural properties and thus improve the accuracy of the embedding (according to Theorem 4.5.1(B)).*

Proof. Suppose the largest diameter of the properties we computed is d, then we can at best consider the local network structure within d-hop neighborhood for determining if two nodes are automorphically equivalent, which means two nodes that are only isomorphic within d-hop neighborhood will be deemed automorphically equivalent. After one recursive step when we compare their neighbors' embedding which essentially captures their neighbors' $d-hop$ local structure, we're essentially increasing the neighborhood diameter from d to $2d$. Similarly, with k step we can consider neighborhood within radius $k \cdot d$. □

5. A COLLECTIVE LEARNING FRAMEWORK TO BOOST GNN EXPRESSIVENESS FOR NODE CLASSIFICATION

5.1 Introduction

A large body of work in relational learning focuses on *collective classification* frameworks for strengthening poorly-expressive (i.e., local) relational node classifiers (e.g., relational logistic regression, naive Bayes, decision trees [79]), by incorporating dependencies among node labels and propagating inferences during classification to improve performance, particularly in semi-supervised settings [4], [9], [30]. However, a long-standing open question is *when/if collective inference is needed*, particularly as more expressive relational graph models become available, e.g., Graph Neural Networks (GNNs).

Despite the recent success of GNNs at node and graph classification tasks [8], [16], [17], [80], these GNNs are no more powerful than the Weisfeiler-Lehman (WL) graph isomorphism test, and thus, inherit its shortcomings. In other words, existing GNNs are not universal (most-expressive) graph representations [5], [6], [8], [81]. This implies that these GNNs (which we refer to as WL-GNNs and also includes GCNs [16]) are not expressive enough for some node classification tasks, since their representation can provably fail to distinguish non-isomorphic nodes with different labels.

While recently there has been increasing interest in developing more expressive WL-GNNs for graph classification tasks that can differentiate non-isomorphic graphs by considering higher-order GNNs (e.g. [82]–[86], these methods primarily consider graph-level representations and, even when they can be adapted for node-level classification tasks, they would be computationally expensive to apply. Is there a easy-to-implement add-on procedure to existing WL-GNNs that can boost their node classification expressiveness?

To address this question, in this work, we theoretically and empirically investigate the potential for collective inference to improve the expressiveness of GNNs. We devise an add-on training and inference procedure, which we denote *collective learning*, that incorporates label dependencies among neighboring nodes via predicted label sampling—akin to how collective classification improves not-so-expressive classifiers—and show that it can improve the expressiveness of *any* WL-GNN.

Contributions:

- We propose *CL+GNN*, an add-on collective learning framework to GNNs that provably boosts their expressiveness for node classification tasks, beyond that of an *optimal* WL-GNN[1]. *CL+GNN* uses self-supervised learning and Monte Carlo sampled embeddings to incorporate node labels during inductive learning—and it can be implemented with *any* component GNN.

- We provide theoretical analysis of *CL+GNN*.

  - Theorem 5.4.1 shows that collective classification is provably unnecessary for GNNs that are most-expressive.

  - Since WL-GNNs are not most-expressive, Theorem 5.4.2 and Proposition 5.4.1 show that *CL+GNN* boosts the expressiveness of optimal WL-GNN and practical WL-GNNs.

  - Corollary 1 shows that previous attempts to incorporate collective inference into WL-GNNs (which in contrast to *CL+GNN* do not Monte Carlo sample embeddings) *cannot* increase expressivity beyond that of an optimal WL-GNN.

- We design and conduct extensive experiments to confirm the above theoretical claims. *CL+GNN* achieves a consistent improvement of node classification accuracy, across a variety of state-of-the-art WL-GNNs, for tasks involving unlabeled and partially-labeled test graphs. Our ablation study demonstrates the effectiveness of our approach incorporating collective learning in GNNs via self-supervised learning with Monte Carlo sampling of embeddings.

5.2 Problem formulation

We consider the problem of *inductive* node classification across partially-labeled graphs, which takes as input a graph $G^{(\text{tr})} = (V^{(\text{tr})}, E^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})})$ for training, where $V^{(\text{tr})}$ is a set of $n^{(\text{tr})}$ vertices, $E^{(\text{tr})} \subset V^{(\text{tr})} \times V^{(\text{tr})}$ is a set of edges with adjacency matrix $\boldsymbol{A}^{(\text{tr})}$, $\boldsymbol{X}^{(\text{tr})}$

[1]↑We use the term optimal WL-GNN to refer to the most expressive version of a GNN–one that has the same distinguishing power as a Weisfeiler-Lehman test. Note this is not a universal graph representation.

is a $n^{(\text{tr})} \times p$ matrix containing node attributes as p-dimensional vectors, and $\boldsymbol{Y}_L^{(\text{tr})}$ is a set of observed labels (with C classes) of a connected set of nodes $V_L^{(\text{tr})} \subset V^{(\text{tr})}$, where $V_L^{(\text{tr})}$ is assumed to be a proper subset of $V^{(\text{tr})}$, noting that $V_L^{(\text{tr})} \neq \emptyset$. Let $\boldsymbol{Y}_U^{(\text{tr})}$ be the unknown labels of nodes $V_U^{(\text{tr})} = V^{(\text{tr})} \setminus V_L^{(\text{tr})}$. The goal is to learn a *joint* model of $\boldsymbol{Y}_U^{(\text{tr})} \sim P(\boldsymbol{Y}_U | G^{(\text{tr})})$ and apply this same model to predict hidden labels $\boldsymbol{Y}_U^{(\text{te})}$ in another test graph $G^{(\text{te})}$, i.e., $\hat{\boldsymbol{Y}}_U^{(\text{te})} = \arg\max_{\boldsymbol{Y}_U} P(\boldsymbol{Y}_U | G^{(\text{te})})$. The test graph $G^{(\text{te})}$ can be partially labeled or unlabeled so $V_L^{(\text{te})} \supseteq \emptyset$.

Graph Neural Networks (GNNs), which aggregate node attribute information to produce node representations, have been successfully used for this task. At the same time, relational machine learning (RML) methods, which use collective inference to boost the performance of local node classifiers via (predicted) label dependencies, have also been successfully applied to this task.

Since state-of-the-art GNNs are not most-expressive for node classification [5], [8], collective classification ideas may help to improve the expressiveness of GNNs. In particular, collective inference methods often *sample* predicted labels (conditioned on observed labels) to improve the local representation around nodes and approximate the joint distribution $P(\boldsymbol{Y}_U | G^{(\text{te})})$. We also know from recent research that sampling randomized features can boost GNN expressiveness [6]. This leads to the key conjecture of this work Section 5.2, which we prove theoretically in Section 5.4 and validate empirically by extensive experimentation in Section 5.5.

Since current Graph Neural Networks (e.g. GCN, GraphSAGE, TK-GCN) cannot produce most expressive graph representations, collective learning (which takes label dependencies into account via Monte Carlo sampling) can improve the accuracy of node classification by producing a more expressive graph representation.

Why? Because WL-GNNs can extract more information about local neighborhood dependencies via sampling [6], and sampling predicted labels allows GNNs to pay attention to the relationship between node attributes, the graph topology, and label dependencies in local neighborhoods. With *collective learning*, GNNs will be able to incorporate more information into the estimated joint label distribution. Next, we describe our *collective learning* framework.

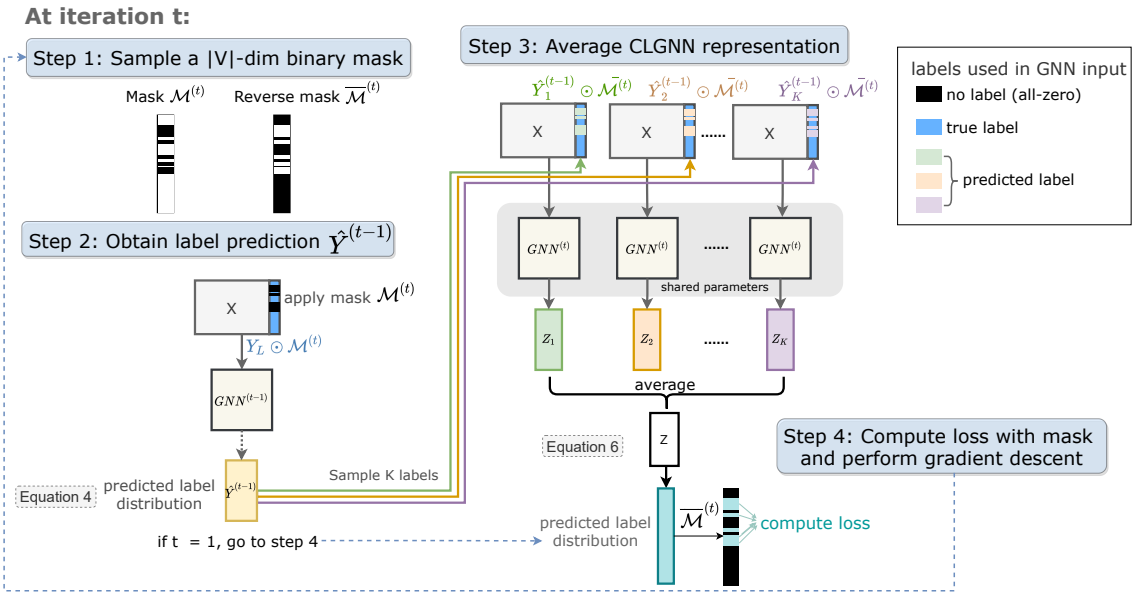


Figure 5.1. CLGNN model framework. Each iteration consists of four steps: (**Step 1**) Sample a random mask; (**Step 2**) Obtain predicted label distribution using the WL-GNN structure; (**Step 3**) Sample predicted labels for whatever nodes are masked, use again as input to the WL-GNN and average representations over the sampled predicted labels; (**Step 4**) Perform one optimization step by minimizing a negative log-likelihood upper bound.

5.3 Proposed Framework: *Collective Learning*

In this section, we outline *CL+GNN*. It is a general framework to incorporate any GNN, and combines *self-supervised learning approach* and *Monte Carlo embedding sampling* in an *iterative* process to improve inductive learning on partially labeled graphs.

Specifically, given a partially labeled training graph $G^{(\text{tr})} = (V^{(\text{tr})}, E^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})})$ with adjacency matrix $\boldsymbol{A}^{(\text{tr})}$ and a partially-labeled test graph $G^{(\text{te})} = (V^{(\text{te})}, E^{(\text{te})}, \boldsymbol{X}^{(\text{te})}, \boldsymbol{Y}_L^{(\text{te})})$ with adjacency matrix $\boldsymbol{A}^{(\text{te})}$. The goal of inductive node classification task is to train a joint model on $G^{(\text{tr})}$ to learn $P(\boldsymbol{Y}_U | G^{(\text{tr})})$ and apply it to $G^{(\text{te})}$ by replacing the input graph $G^{(\text{tr})}$ with $G^{(\text{te})}$. Suppose the graphs $G^{(\text{tr})}$ and $G^{(\text{te})}$, we can define $\boldsymbol{Y}_L^{(\text{tr})}$ as a binary (0-1) matrix of dimension $|V^{(\text{tr})}| \times C$, and $\boldsymbol{Y}_L^{(\text{te})}$ of dimension $|V^{(\text{te})}| \times C$, where the rows corresponding to the one-hot encoding of the (available) labels.

(Background) GNN and representation learning. Given a partially labeled graphs $G^{(\text{tr})}$, WL-GNNs generate node representation by propagating feature information throughout the graph. Specifically, $\forall v \in V^{(\text{tr})}$,

$$P(\boldsymbol{Y}_v | \boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})}) = \sigma(\mathbf{W}\boldsymbol{Z}_v + \mathbf{b}), \tag{5.1}$$

where $\boldsymbol{Z}_v = \text{GNN}(\boldsymbol{X}^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})}; \Theta)_v$ is the GNN representation of node v, $\sigma(\cdot)$ is the softmax activation, and Θ, $\mathbf{W}$ and $\mathbf{b}$ are model parameters, which are learned by minimizing the cross-entropy loss between true labels $\boldsymbol{Y}_L^{(\text{tr})}$ and the predicted labels.

The collective learning framework. Following Section 5.2, we propose Collective Learning GNNs (*CL+GNN*), which includes label information as input to GNNs to produce a more expressive representation. The overall framework follows four steps: (**Step 1**) Sample a random binary mask to include true labels (if available) in the input; (**Step 2**) Obtain predicted label distribution using the WL-GNN structure; (**Step 3**) Sample predicted labels for whatever nodes are masked, combine with available true labels (if any), and use again as input to the WL-GNN; finally average representations of the WL-GNN over the sampled predicted labels; (**Step 4**) Perform one optimization step by minimizing a negative log-likelihood upper bound. These steps are shown in Figure 5.1. Collective learning for WL-GNNs then consists of iterating over Steps 1-4 for $t = 1, \ldots, T$ iterations. Finally, once optimized, we perform inference via Monte Carlo estimates.

***CL+GNN* loss and its representation averaging.** The input to GNNs is typically the full graph $G^{(\text{tr})}$. If we included the observed labels $\boldsymbol{Y}_L^{(\text{tr})}$ directly in the input, then it would be trivial to learn a model that predicts part of the input. Instead, we either (*scenario test-unlabeled*) mask all label inputs if the test graph $G^{(\text{te})}$ is expected to be unlabeled; or (*scenario test-partial*) if $G^{(\text{te})}$ is expected to have partial labels, we apply a mask to the labels we wish to predict in training so they do not appear in the input $\boldsymbol{Y}_L^{(\text{tr})}$.

Specifically, at the t-th step of our optimization —these steps can be coarser than a gradient step — we either (*scenario test-partial*) sample a mask $\boldsymbol{M}^{(t)} \sim \text{Uniform}(\mathcal{M})$ or (*scenario test-unlabeled*) set $\boldsymbol{M}^{(t)} = \mathbf{0}$. For now, we assume we can sample $\hat{\mathbf{Y}}^{(t-1)} = (\hat{\boldsymbol{Y}}_v^{(t-1)})_{v \in V^{(\text{tr})}}$ from an estimate of the distribution $P(\boldsymbol{Y}_v^{(\text{tr})} | \boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})} \odot \boldsymbol{M}^{(t)}, \boldsymbol{A}^{(\text{tr})})$ —we

will come back to this assumption soon. Let $\boldsymbol{X}^{(\text{tr})}_{\boldsymbol{Y}_L^{(\text{tr})},\hat{\boldsymbol{Y}}^{(t-1)},\boldsymbol{M}^{(t)}}$ be the matrix concatenation between $\boldsymbol{Y}_L^{(\text{tr})} \odot \boldsymbol{M}^{(t)} + \hat{\boldsymbol{Y}}^{(t-1)} \odot \overline{\boldsymbol{M}}^{(t)}$ and $\boldsymbol{X}^{(\text{tr})}$, where again $\overline{\boldsymbol{M}} := \mathbf{1} - \boldsymbol{M}$ is the bitwise negated matrix of $\boldsymbol{M}$. Let

$$\boldsymbol{Z}_v^{(t)}(\boldsymbol{M}^{(t)};\Theta) = \mathbb{E}_{\hat{\boldsymbol{Y}}^{(t-1)}}\Big[\text{GNN}(\boldsymbol{X}^{(\text{tr})}_{\boldsymbol{Y}_L^{(\text{tr})},\hat{\boldsymbol{Y}}^{(t-1)},\boldsymbol{M}^{(t)}}, \boldsymbol{A}^{(\text{tr})};\Theta)_v\Big], \tag{5.2}$$

where GNN represents an arbitrary graph neural network model and Z_v^t is the *CL+GNN*'s representation obtained for node $v \in V^{(\text{tr})}$ at step $t \geq 1$.

Our optimization is defined over the expectation of $\boldsymbol{Z}_v^{(t)}(\boldsymbol{M}^{(t)})$ w.r.t. to the sampled predicted labels $\hat{\boldsymbol{Y}}^{(t-1)}$ (Equation (5.2)) and over a loss averaged over all sampled masks (noting that the case where $\boldsymbol{M}^{(t)} = \mathbf{0}$ is trivial):

$$\begin{aligned}\Theta_t, \mathbf{W}_t, \mathbf{b}_t = \arg\max_{\Theta,\mathbf{W},\mathbf{b}} \mathbb{E}_{\boldsymbol{M}^{(t)}}\Bigg[&\sum_{v \in V_L^{(\text{tr})}} \overline{\boldsymbol{M}}_v^{(t)} \\ &\times \log \sigma(\mathbf{W}\boldsymbol{Z}_v^{(t)}(\boldsymbol{M}^{(t)};\Theta) + \mathbf{b})_{y_v^{(\text{tr})}}\Bigg],\end{aligned} \tag{5.3}$$

where again, $\sigma(\cdot)$ is the softmax activation function, and $V_L^{(\text{tr})}$ are the labeled nodes in training graph.

Stochastic optimization of Equation (5.3). Equation (5.3) is based on a pseudo-likelihood, where the joint distribution of the labels $\{\boldsymbol{Y}_v^{(\text{tr})} : v \in V_L^{(\text{tr})} \text{ s.t. } \overline{\boldsymbol{M}}_v^{(t)} = 1\}$ is decomposed as marginal distributions resulting in the sum over $V_L^{(\text{tr})}$.

(Step 1) Sample a binary mask In (*scenario test-partial*), where $G^{(\text{te})}$ is expected to have some observed labels, we randomly sample a binary mask $\boldsymbol{M} \sim \text{Uniform}(\mathcal{M})$ from a set of masks, where $\boldsymbol{M}$ is a $|V^{(\text{tr})}| \times C$ binary (0-1) matrix with the same $|V^{(\text{tr})}|$-dimensional vector in each column. By applying the mask on the observed labels $\boldsymbol{Y}_L^{(\text{tr})}$, the set of true labels is effectively partitioned into two parts, where part of the true labels $\boldsymbol{Y}_L^{(\text{tr})} \odot \boldsymbol{M}$ are used as input to *CL+GNN*, and the other part $\boldsymbol{Y}_L^{(\text{tr})} \odot \overline{\boldsymbol{M}}$ are used as optimization target. Here $\overline{\boldsymbol{M}} := \mathbf{1} - \boldsymbol{M}$ is the bitwise negated matrix of $\boldsymbol{M}$.

(Step 2) Obtaining $\hat{\mathbf{Y}}^{(t-1)}$. Note that in Equation (5.2), we first need to obtain the predicted label distribution $\hat{\mathbf{Y}}^{(t-1)}$ with mask $M^{(t)}$ to sample labels from. At iteration t, we use the learned *CL+GNN* model parameter Θ_{t-1} to obtain $\boldsymbol{Z}_v^{(t-1)}$ according to Equation (5.2) and use the *CL+GNN* model parameters $\mathbf{W}_{t-1}, \mathbf{b}_{t-1}$ to obtain the label prediction recursively, i.e. $\forall v \in V^{(\text{tr})}$,

$$\hat{\boldsymbol{Y}}_v^{(t-1)} \sim \text{Categorical}(\sigma(\mathbf{W}_{t-1}\boldsymbol{Z}_v^{(t-1)}(\boldsymbol{M}^{(t)};\Theta_{t-1}) + \mathbf{b}_{t-1})), \tag{5.4}$$

where

$$\boldsymbol{Z}_v^{(t-1)}(\boldsymbol{M}^{(t)};\Theta_{t-1}) = \text{GNN}(\boldsymbol{X}^{(\text{tr})}_{\boldsymbol{Y}_L^{(\text{tr})},\mathbf{0},\boldsymbol{M}^{(t)}}, \boldsymbol{A}^{(\text{tr})};\Theta_{t-1})_v \tag{5.5}$$

Note that $\boldsymbol{Z}_v^{(t-1)}(\boldsymbol{M}^{(t)};\Theta)$ does not use any predicted labels in the GNN input, i.e. it uses the true labels for masked nodes or all-zero labels for unmasked nodes.

In order to optimize Equation (5.3), we compute gradient estimates w.r.t. Θ and $\mathbf{b}$ using the following sampling procedure.

(Step 3) We first need to compute an unbiased estimate of $\{\boldsymbol{Z}_v^{(t-1)}\}_{v \in V_L^{(\text{tr})}}$ in Equation (5.2) using K i.i.d. samples $\hat{\boldsymbol{Y}}^{(t-1)}$ from the model obtained at time step $t-1$ (as describe above), i.e.

$$\tilde{\boldsymbol{Z}}_v^{(t)}(\boldsymbol{M}^{(t)};\Theta_t) = \frac{1}{K}\sum_{k=1}^{K} \text{GNN}(\boldsymbol{X}^{(\text{tr})}_{\boldsymbol{Y}_L^{(\text{tr})},\hat{\boldsymbol{Y}}_k^{(t-1)},\boldsymbol{M}^{(t)}}, \boldsymbol{A}^{(\text{tr})};\Theta_t)_v, \tag{5.6}$$

where again $\boldsymbol{X}^{(\text{tr})}_{\boldsymbol{Y}_L^{(\text{tr})},\hat{\boldsymbol{Y}}^{(t-1)},\boldsymbol{M}^{(t)}}$ is the matrix concatenation between $\boldsymbol{X}^{(\text{tr})}$ and $\boldsymbol{Y}_L^{(\text{tr})} \odot \boldsymbol{M}^{(t)} + \hat{\boldsymbol{Y}}^{(t-1)} \odot \overline{\boldsymbol{M}}^{(t)}$.

Note that the time/space complexity of the *CL+GNN* is K times the time/space complexity of the corresponding GNN model as we have to compute K representations for each node at each stochastic gradient step.

(Step 4) Next, we need an unbiased estimate of the expectation over mask $\boldsymbol{M}^{(t)}$ in Equation (5.3). In *(scenario test-partial)* the unbiased estimates are obtained by sampling $\boldsymbol{M}^{(t)} \sim \text{Uniform}(\mathcal{M})$ at each gradient step, in the *(scenario test-unlabeled)* the value obtained is exact since $\boldsymbol{M}^{(t)} = \mathbf{0}$. The mask $\boldsymbol{M}^{(t)}$ is used, along with the estimate $\tilde{\boldsymbol{Z}}$ from Equa-

tion (5.6), to compute the loss function as in Equation (5.3) and perform a gradient descent step. Proposition 5.4.2 shows that the above procedure is a proper surrogate upperbound of the loss function.

Inference with learned model.

Once the *CL+GNN* parameters $\Theta_T, \boldsymbol{W}_T, \boldsymbol{b}_T$ are learned according to Equation (5.3) on the training graph $G^{(\text{tr})}$, given an any-size attributed graph $G^{(te)}$, we sample J masks $\boldsymbol{M}$ of size $|V^{(\text{te})}|$, either (*scenario test-partial*) sampling $\boldsymbol{M} \sim \text{Uniform}(\mathcal{M})$ or (*scenario test-unlabeled*) set $\boldsymbol{M} = \mathbf{0}$. For each mask, we apply the same procedure as in **(Step 2)** and **(Step 3)** to obtain predicted label distribution $\hat{\mathbf{Y}}^{(\text{tmp})}$, and then sample K labels $\{\hat{\mathbf{Y}}_1^{(\text{tmp})}, \cdots, \hat{\mathbf{Y}}_K^{(\text{tmp})}\}$ from it and pass to the learned model. The node representations for $v \in V^{(\text{te})}$ are obtained using $\boldsymbol{M}$ and $\hat{\mathbf{Y}}_{1,\cdots,K}^{(\text{tmp})}$:

$$\tilde{\boldsymbol{Z}}_v(M;\Theta_T) = \frac{1}{K}\sum_{k=1}^{K} \text{GNN}(\boldsymbol{X}^{(\text{te})}_{\boldsymbol{Y}_L^{(\text{te})},\hat{\boldsymbol{Y}}_k^{(\text{tmp})},\boldsymbol{M}}, \boldsymbol{A}^{(\text{te})};\Theta_T)_v,$$

where

$$(\hat{\boldsymbol{Y}}_k^{(\text{tmp})})_v \sim \text{Categorical}(\sigma(\mathbf{W}_T \boldsymbol{Z}_v^{(\text{tmp})}(\boldsymbol{M};\Theta_T) + \mathbf{b}_T)),$$

and

$$\boldsymbol{Z}_v^{(\text{tmp})}(\boldsymbol{M};\Theta_T) = \text{GNN}(\boldsymbol{X}_{\boldsymbol{Y}_L^{(\text{te})},\mathbf{0},\boldsymbol{M}^{(\text{te})}}, \boldsymbol{A}^{(\text{te})};\Theta_T)_v.$$

The final node representation is computed as the average over all sampled masks:

$$\tilde{\boldsymbol{Z}}_v = \frac{1}{J}\sum_{\text{j}=1}^{J} \tilde{\boldsymbol{Z}}_v(M_\text{j};\Theta_T),$$

where J and K are hyperparameters, J is the number of masks for our Monte Carlo average and K is the number of Monte Carlo samples of $\hat{\mathbf{Y}}^{(\text{tmp})}$. Then the label predictions are obtained using the learned *CL+GNN* parameters $\boldsymbol{W}_T, \boldsymbol{b}_T$:

$$\hat{\boldsymbol{Y}}_v^{(\text{te})} \sim \text{Categorical}(\sigma(\mathbf{W}_T \tilde{\boldsymbol{Z}}_v + \mathbf{b}_T)_v), \ \forall v \in V^{(\text{te})}. \tag{5.7}$$

5.4 *Collective Learning* Analysis

Is collective classification able to better represent target label distributions than node representation learning? The answer to this question is both *yes (for WL-GNNs) and no (for most-expressive representations)*. Theorem 5.4.1 shows that a most-expressive graph representation [6], [7], [87] would not benefit from a collective learning boost. All proofs can be found in the Appendix.

Theorem 5.4.1 (Collective classification can be unnecessary)**.** *Consider the task of predicting node labels when no labels are available in test data. Let* $\Gamma^\star(v, G^{(te)})$ *be a most-expressive representation of node* $v \in V^{(te)}$ *in graph* $G^{(te)}$ *. Then, for any collective learning procedure predicting the class label of* $v \in V^{(te)}$*, there exists a classifier that takes* $\Gamma^\star(v, G^{(te)})$ *as input and predicts the label of* v *with equal or higher accuracy.*

While Theorem 5.4.1 shows that the most-expressive graph representation does not need collective classification, WL-GNNs are not most-expressive [5], [6], [8]. Indeed, Theorem 5.4.2 and Proposition 5.4.1 show that *CL+GNN* boosts the expressiveness of optimal WL-GNN and practical WL-GNNs, respectivelly. Then, we show that the stochastic optimization in Step 3 optimizes a loss surrogate upper bound.

5.4.1 Expressive power of *CL+GNN*

[5] and [8] show that WL-GNNs are no more powerful in distinguishing non-isomorphic graphs and nodes as the standard Weisfeiler-Lehman graph isomorphism test (1-WL or just WL test). Two nodes are assumed isomorphic by the WL test if they have the same color assignment in the stable coloring.

The node-expressivity of a parameterized graph representation Γ (with parameter $\Gamma(\cdot; \boldsymbol{W})$) can then be determined by the set of graphs for which Γ can identify non-isomorphic nodes:

$$\begin{aligned}\mathcal{G}(\Gamma) = \{G : \exists \boldsymbol{W}_G^\star, \text{ s.t. } \forall u, v \in V_G, \Gamma(G; \boldsymbol{W}_G^\star)_v \\ = \Gamma(G; \boldsymbol{W}_G^\star)_u \text{ iff } u, v \text{ are isomorphic}, G \in \mathbb{G}\},\end{aligned}$$

where $\mathbb{G}$ is the set of all any-size attributed graphs, V_G is the set of nodes in graph G. We call $\mathcal{G}(\Gamma)$ the *identifiable set* of graph representation Γ.

The *most expressive* graph representation $\Gamma^\star$ has an *identifiable set* of all any-size attributed graphs, i.e. $\mathcal{G}(\Gamma^\star) = \mathbb{G}$. We refer to the WL-GNN that is equally expressive as WL test as the *optimal* WL-GNN (or WLGNN*), which is at least as expressive as all other WL-GNNs.

In this section we show that *collective learning* can boost the optimal WLGNN*, i.e., the identifiable set of WLGNN* is a proper subset of collective learning over WLGNN* (denoted *CL+GNN**)

$$\mathcal{G}(\text{WLGNN}^\star) \subsetneq \mathcal{G}(\textit{CL+GNN}^\star).$$

Theorem 5.4.2 (*CL+GNN** expressive power)**.** *Let WLGNN** *be an optimal WL-GNN. Then, the collective learning representation of Equation (5.2), using WLGNN** *as the GNN component, (denoted* CL+GNN**) is strictly more expressive than this WLGNN** *representation model applied to the same tasks.*

Theorem 5.4.2 answers Section 5.2, by showing that by incorporating collective learning and sampling procedures, *CL+GNN* can boost the expressiveness of WL-GNNs, including the optimal WLGNN*.

Corollary 1. *Consider a graph representation learning method that, at iteration t, replaces $\hat{\boldsymbol{Y}}^{(t-1)}$, in Equations (5.2) and (5.4) with a deterministic function over $\boldsymbol{Z}^{(t-1)}$, e.g., a softmax function that outputs $(P(\hat{\boldsymbol{Y}}_v^{(t-1)}|\boldsymbol{Z}^{(t-1)}))_{v\in V^{(tr)}}$. Then, such method will be no more expressive than the optimal WLGNN** *and, hence, less expressive than* CL+GNN**.*

Corollary 1 proves that existing collective approaches are no different than current GNN methods (hence, no boosting). More specifically, it shows that existing graph representation methods that —on the surface— may even look like *CL+GNN*, but do not perform the crucial step of sampling $(\hat{\boldsymbol{Y}}_v^{(t-1)})_{v\in V^{(\text{tr})}}$, unfortunately, are no more expressive than WL-GNNs. Examples of such methods include [88]–[91].

Next, we show the practical benefits of collective learning are even greater when the WL-GNN has limited expressive power due to limited message-passing layers.

5.4.2 How *CL+GNN* further expands the power of few-layer WL-GNNs

A d-layer ($d > 1$) WL-GNN will only aggregate neighborhood information within d hops of any given node (i.e., over a d-hop egonet, defined as the graph representing the connections among all nodes that are at most d hops away from the center node). In practice —mostly for computational reasons— WL-GNNs have many fewer layers than the graph's diameter D, i.e., $d < D$.

For instance, GCN [16] and GraphSAGE [17] both used $d = 2$ in their experiments. Hence, they cannot differentiate two non-isomorphic nodes that are isomorphic within their d-hop neighborhood. We now show that *CL+GNN* can gather $2d$-hop neighborhood information with a d-layer WL-GNN.

Proposition 5.4.1. *Let G_v^d be the d-hop egonet of a node v in graph G with diameter $D > d$. Let v_1 and v_2 be two non-isomorphic nodes whose d-hop egonets are isomorphic (i.e., $G_{v_1}^d$ is isomorphic to $G_{v_2}^d$) but $2d$-hop egonets are not isomorphic. Then, a WL-GNN representation with d layers will generate identical representations for v_1 and v_2 while* CL+GNN *is capable of giving distinct node representations.*

Proposition 5.4.1 shows that collective learning has yet another benefit: *CL+GNN* further boosts the power of WL-GNNs with limited message-passing layers by gathering neighborhood information within a larger radius. Specifically, *CL+GNN* built on a WL-GNN with d layers can enlarge the effective neighborhood radius from d to $2d$ in Equation (5.2) , while WL-GNN would have to stack $2d$ layers to achieve the same neighborhood radius, which in practice may cause optimization challenges (i.e., $d = 2$ is a common hyperparameter value in the literature).

5.4.3 Optimization of *CL+GNN*

Proposition 5.4.2. *If $\forall v \in V_L^{(tr)}$, $\nabla_\Theta(\mathbf{W}\boldsymbol{Z}_v^{(t)}(\boldsymbol{M}^{(t)};\Theta))_{y_v^{(tr)}}$ is bounded (e.g., via gradient clipping), then the optimization in Equation (5.3), with the unbiased sampling of $\{\boldsymbol{Z}_v^{(t-1)}\}_{v\in V^{(tr)}}$ and $\boldsymbol{M}^{(t)}$ described above, results in a Robbins-Monro [92] stochastic optimization algorithm that optimizes a surrogate upper bound of the loss in Equation (5.3).*

Since the optimization objective in Equation (5.3) is computationally impractical, as it requires computing all possible binary masks and label predictions, Proposition 5.4.2 shows that the sampling procedures used in *CL+GNN* that considers K samples of label predictions and a random mask at each gradient step is a feasible approach of estimating an unbiased upper bound of the objective.

5.5 Experiments

5.5.1 Experiment Setup

Datasets. We use datasets of Cora, Pubmed, Friendster, Facebook, and Protein. The largest dataset (Friendster [93]) has 43,880 nodes, which is a social network of users where the node attributes include numerical features (e.g number of photos posted) and categorical features (e.g. gender, college, etc.) encoded as binary one-hot features. The node labels represent one of the five age groups. The full dataset statistics is shown in Table 5.1.

Table 5.1. Dataset statistics

Dataset	# Nodes	# Attributes	# Classes	# Test
Cora	2708	1433	7	1000
Pubmed	19717	500	3	1000
Friendster	43880	644	5	6251
Facebook	4556	3	2	1000
Protein	12679	29	2	2376

- **Cora** and **Pubmed** are benchmark datasets for node classification tasks from [94]. They are citation networks with nodes representing publications and edges representing citation relation. Node attributes are bag-of-word features of each document, and the predicted label is the corresponding research field.
- **Facebook** [95] is social network of Facebook users from Purdue university, where nodes represent users and edges represent friendship. The features of the nodes are: religious views, gender and whether the user's hometown is in Indiana. The predicted labels is political view.

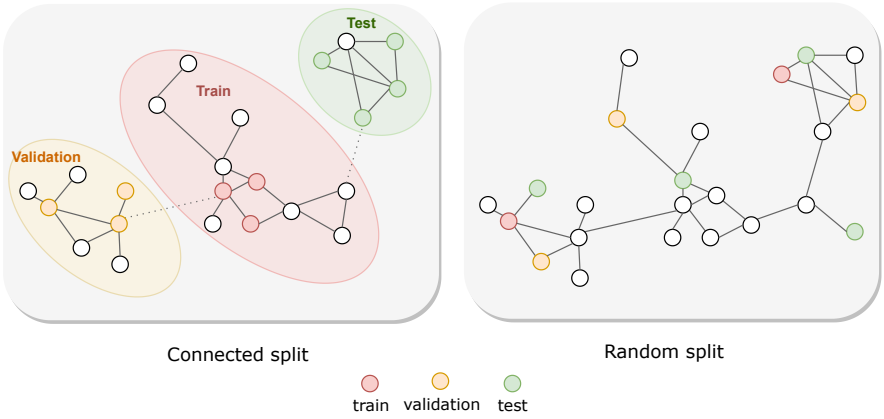


Figure 5.2. Different data splits between our inductive connected split (left) and conventional GNN random split (right)

- **Friendster** [93] is social network. Nodes represent users and edges represent friendship. The node attributes include numerical features (e.g number of photos posted, etc) and categorical features (e.g. gender, college, music interests, etc), encoded as binary one-hot features. The node labels represent one of the five age groups: 0-24, 25-30, 36-40, 46-50 and over 50. This version of the graph contain 40K nodes, 25K of which are labeled.

- **Protein** is a collection of protein graphs from [72]. Each node is labeled with a functional role of the protein, and has a 29 dimensional feature vector. We use 85 graphs with an average size of 150 nodes.

Train/Test split. Since most datasets used to test GNNs consist of a single graph, we apply Louvain community detection algorithm [97] to split each single graph into three clusters for training, validation, and testing respectively, and remove the edges across clusters —shown in Figure 5.2 (left). This mimics the inductive within-graph scenario that often occurs in real world settings, where a connected subgraph is used to learn a model to generalize the remainder of the graph —e.g., Facebook would train a model on Iceland or New Zealand and then apply it to the rest of the world, see methodology in [98].

Our train/test data split is different than previous GNN works, which have adopted random node split between train and test —shown in Figure 5.2 (right)— and can put test nodes close to the training nodes, making it much easier to leverage test node attributes

Table 5.2. Node classification accuracy with unlabeled and partially-labeled test data. Numbers in bold represent significant improvement in a paired t-test at the $p < 0.05$ level, and numbers with * represent the best performing method in each column. Cora$^{\text{connect}}$ and Pubmed$^{\text{connect}}$ are our processed graphs with the connected split illustrated in Figure 5.2 (left).

		Cora$^{\text{connect}}$		Pubmed$^{\text{connect}}$		Friendster		Facebook		Protein	
# train labels:		85 (3.21%)		300 (1.52%)		641 (1.47%)		80 (1.76%)		7607 (30%)	
% labels in $G^{(\text{te})}$:		0%	50%	0%	50%	0%	50%	0%	50%	0%	50%
Random		14.28 (0.00)	14.28 (0.00)	33.33 (0.00)	33.33 (0.00)	20.00 (0.00)	20.00 (0.00)	50.00 (0.00)	50.00 (0.00)	50.00 (0.00)	50.00 (0.00)
GCN [16]	-	64.74 (1.51)	66.34 (1.84)	54.56 (2.49)	58.41 (1.27)	25.97 (0.69)	24.26 (0.52)	50.58 (1.38)	51.04 (1.20)	75.86 (1.11)	77.54 (1.09)
	+ CL	**+3.72 (0.40)**	**+12.41 (1.96)**	**+1.95 (0.69)**	**+15.37 (2.01)**	**+0.70 (0.14)**	**+1.99 (0.74)**	**+2.24 (0.81)**	**+8.51 (1.09)**	**+1.22 (0.51)**	**+0.75 (0.33)**
GS [17]	-	65.35 (1.19)	67.71 (1.53)	55.56 (2.44)	59.12 (2.02)	26.45 (0.62)	24.75 (0.39)	51.14 (1.24)	52.06 (1.29)	73.85 (1.12)	73.01 (2.28)
	+ CL	**+2.81 (1.02)**	**+9.94(1.04)**	+1.05 (0.83)	**+14.71 (2.89)**	+0.13 (0.41)	**+1.40 (0.62)**	**+1.77 (0.55)**	**+7.80 (0.84)**	**+0.84 (0.12)**	+1.47 (0.63)
TK [80]	-	68.47 (1.31)	69.50 (0.55)	59.05 (2.13)	60.77 (1.53)	25.93 (0.91)	24.42 (1.44)	52.74 (1.62)	53.48 (1.48)	73.65 (1.69)	78.94 (1.50)
	+ CL	**+1.50 (0.61)**	**+7.92 (0.75)**	+0.23 (0.61)	**+13.62 (1.84)**	**+1.20 (0.14)**	**+2.34 (0.42)**	**+3.26 (0.98)**	**+4.60 (1.16)**	**+1.31 (0.27)**	**+1.36 (0.94)**
GRAND [96]	-	71.55 (1.07)	73.19 (0.41)	61.82 (6.40)	63.23 (7.22)	28.03 (1.02)	27.02 (0.84)	47.10 (0.27)	48.14 (0.52)	75.43 (1.12)	79.69 (0.29)
	+ CL	**+0.80 (0.31)**	**+2.30 (0.56)**	**+3.79 (1.50)**	**+5.17 (1.44)**	+0.37 (0.39)	**+4.21 (0.72)**	**+6.38 (2.29)**	**+5.72 (2.34)**	**+0.51 (0.36)**	**+0.75 (0.20)**
Best of CL		72.36 (1.20)*	78.31 (0.58)*	65.61 (6.60)*	74.39 (1.72)*	28.40 (0.85)*	31.23 (1.05)*	56.01 (1.48)	59.86 (0.83)	77.08 (1.03)	80.52 (0.37)
PL-EM [30]	-	20.66 (0.04)	54.22 (0.94)	38.85 (0.03)	65.65 (4.33)	18.13 (0.23)	22.25 (0.87)	50.58 (0.03)	61.17 (1.14)	78.46 (1.45)	77.95 (1.56)
ICA [11]	-	62.29 (2.18)	65.51 (1.30)	43.93 (6.84)	44.61 (6.24)	26.48 (1.37)	27.80 (1.56)	61.56 (1.10)*	62.04 (1.92)*	84.88 (3.35)*	84.39 (4.08)*
GMNN [90]	-	66.35 (3.12)	72.04 (2.45)	57.13 (3.01)	67.94 (4.40)	24.92 (1.20)	26.88 (1.53)	49.56 (0.88)	57.09 (0.78)	76.75 (0.74)	75.96 (0.76)

during training. Our use of a hard split between train and test (connected split) is the reason why the model performance reported in our paper is not directly comparable with the reported results in previous GNN papers, even though we used the same implementations and hyper-parameter search procedures. In our experiments, we tested two different label rates in test graph: 0 (unlabeled) and 50% (reveal 50% testing labels and evaluate on the rest). We run five trials for all the experiments, and in each trial we randomly pick a connected subgraph within the training cluster and reveal their labels for training.

As our method can be applied to any GNN models, we use four representative GNNs as examples:

- GCN [16] which includes two graph convolutional layers. Here we implemented an inductive variant of the original GCN model for our tasks.
- Supervised GraphSage [17] (denoted by GS) with Mean pooling aggregator. We use sample size of 5 for neighbor sampling.
- Truncated Krylov GCN [80] (denoted by TK), a recent GNN model that leverages multi-scale information in different ways and are scalable in depth. The TK has stronger expressive power and achieved state-of-the-art performance on node classification tasks.

We implemented Snowball architecture which achieved comparable performance with the other truncated Krylov architecture according to the original paper.

- GRAND [96], a recent GNN model using random propagation strategy to perform graph data augmentation, in order to mitigate the issues of over-smoothing and non-robustness. GRAND achieved state-of-the-art performance on several semi-supervised node classification tasks.

For each of the GNNs, we compare its baseline performance (on its own) to the performance achieved using collective learning in *CL+GNN* (using that GNN).

Hyperparameter setting. For a fair comparison, we adopt the same hyper-parameter tuning strategy for the baseline GNNs and *CL+GNN*, e.g. hidden dimensions, learning rate, early-stopping procedures. Specifically, we searched the initial learning rate within {0.005, 0.01, 0.05} with weight decay of 0.0005. Dropout is applied to all the layers with $p = 0.5$. Hidden units are searched within {16, 32} if the dataset wasn't used by the original paper, or set as the same number as originally chosen in the paper. The number of layers is set to 2 for GCN [16] and GraphSage [17] as used in their paper, and set to 10 for TK [8]. For GraphSage [17], the neighborhood sample size is set to 5. For GRAND [96], the two additional hyper-parameters are propagation step (K) and data augmentation times (S). We search K in $\{2, 5, 8\}$ and S in $\{2, 4\}$ by grid search.

For *CL+GNN*, the additional hyperparameters are (1) the sample size of predicted labels $\hat{Y}$ (K), and (2) the number of model iterations (T). We set sample size $K = 5$ for Friendster dataset (due to memory issue) and $K = 10$ for all other datasets. For label rate of 50%, the model is trained for $T = 10$ iterations, and each iteration contains 100 epochs. Note that we sample a new binary mask for each epoch as described in Section 5.3. For label rate of 0%, the model is trained for $T = 3$ iterations, and each iteration contains up to 500 epochs which can be early stopped if the validation accuracy decreases for a specified consecutive epochs. The numbers of iterations are empirically determined as only marginal improvements are observed after 3 iterations for unlabeled test data and 10 iterations for partially-labeled test data. The validation accuracy is used to choose the best epoch. For each search of

hyperparameter configuration, we run the experiments with 10 random seeds and select the best configuration of hyperparameters based on average accuracy on validation set.

Note that the hyper-parameter tuning could be done more aggressively to further boost the performance of *CL+GNN*, e.g. using more layers for TK [8] and searching K and S in a larger space for GRAND, but our main goal is to evaluate the relative improvements of *CL+GNN* on the corresponding non-collective GNNs.

In addition, we also compare to three relational classifiers, ICA [11], PL-EM [30] and GMNN [90]. The first two models apply collective learning and inference with simple local classifiers —— Naive Bayes for PL-EM and Logistic regression for ICA. GMNN is the state-of-the-art collective model with GNNs, which uses two GCN models to model label dependency and node attribute dependency respectively. All the three models take true labels in their input, thus we use $\boldsymbol{Y}_L^{(\text{tr})}$ for training and $\boldsymbol{Y}_L^{(\text{te})}$ for testing.

We report the average accuracy score and standard error of five trials for the baseline models, and compute the absolute improvement of accuracy of our method over the corresponding base GNN. The best performance among all *CL+GNN* is also reported. We compute the *balanced* accuracy scores on Friendster dataset as the label is highly imbalanced. To evaluate the significance of *CL+GNN* improvements, we performed a paired t-test with five trials.

5.5.2 Results

The node classification accuracy of all the models is shown in Table 5.2. Our proposed collective learning boost is denoted as +CL (for **C**ollective **L**earning) and our model performance (absolute % of improvement over the corresponding baseline GNN) is shown in shaded area. Numbers in bold represent significant improvement over the baseline GNN based on a paired t-test ($p < 0.05$), and numbers with * is the best performing method in each column.

Comparison with baseline GNN models. Table 5.2 shows that our method improves the corresponding non-collective GNN models for all the four model architectures (i.e. GCN, GraphSage, TK and GRAND). Although all the models have large variances over multiple trials —because different parts of the graphs are being trained in different trials— adding CL consistently improves the baseline GNN. The results from a paired t-test comparing

the performance of our method and the corresponding non-collective GNN shows that the improvement is almost always significant at $p = 0.05$ (marked as bold), with only five exceptions.

Comparing the gains on different datasets in Table 5.2, adding CL to GNNs achieved smaller gains on Friendster especially when no test labels were available. This is because Friendster is more sparse than the other graphs (e.g. edge density of Friendster is 1.5e-4 while Cora is 1.44e-3 [93]), which makes it hard for any model to propagate label information and capture label dependencies.

As expected, comparing the improvement over various GNNs with different expressive power, we observe that in general adding CL boosts the gains of simpler GNN models (i.e. GCN and GS). For example, Table 5.2 shows that adding CL to a GCN can boost its accuracy by +12.41% (Cora) while the boost over TK is smaller at +7.92% in the same task. This is in line with our assumption in Section 5.2 that collective learning can help weaker GNNs produce a more expressive representation. As GCN is less expressive than TK, there is a larger room to increase its expressiveness.

Note that the gains in Table 5.2 are generally much larger when we go from 0% to 50% of the labels available in test. For example, when combining with GCN, the improvements of our method are 3.72% and 2.24% for unlabeled Cora and Facebook test sets, but with partially-labeled test data, the improvements are 12.41% and 8.51% respectively. This shows the importance of modeling label dependency especially when the some test data labels are observed.

Comparison with other relational classifiers The two baseline non-GNN relational models —i.e. PL-EM and ICA— generally perform worse than the three GNNs, with exceptions on Protein and Facebook datasets. This could be because the two dataset has only a few node attributes (3 for Facebook and 29 for Protein), while the other graphs have hundreds or thousands of node attributes, which makes it easier for the more powerful classifier (i.e. GNNs) to overfit on Facebook and Protein. Moreover, this could also be because the two non-GNN models generally need a larger portion of labeled set to train the weak local classifier, whereas GNNs utilize a neural network architecture as "local classifier", which is better at representation learning by transforming and aggregating node attribute

information. However, when the model is trained with a large training set (e.g. with 30% nodes on Protein dataset), modeling the label dependency becomes crucial. At the same time, our method is still able to boost performance on the two datasets.

For GMNN [90], a collective GNN model, it achieves better performance than its non-collective base model, i.e. GCN on most of the datasets, and we can see that adding CL to GCN achieved comparable or better performance than GMNN. However, combing CL with other more powerful GNNs can easily out-perform GMNN (e.g., on Cora and Friendster, GRAND+CL significantly outperforms GMNN). When the test labels are available, GMNN is able to out-perform several GNNs by leveraging test label information, but the best of *CL+GNN* still out-performs GMNN consistently.

5.5.3 Ablation study

Table 5.3. Model performance on Cora (unlabeled test data) where we vary the sampling procedure and try ensemble. The alternative methods achieved sub-optimal performance compared to *CL+GNN*.

Base GNN	description	sampling from predicted label dist.	sampling from uniform label dist.	ensemble	% improvement
GCN	+ CL (ours)	✓			+3.72 (0.40)
	+ CL w/o predicted label		✓		+0.36 (1.27)
	+ Ensemble			✓	+0.32 (2.28)
TK	+ CL (ours)	✓			+1.54 (0.60)
	+ CL w/o predicted label		✓		-1.02 (0.55)
	+ Ensemble			✓	+0.63 (0.90)
GRAND	+ CL (ours)	✓			+0.80 (0.31)
	+ CL w/o predicted label		✓		+0.35 (0.66)
	+ Ensemble			✓	+0.71 (0.13)

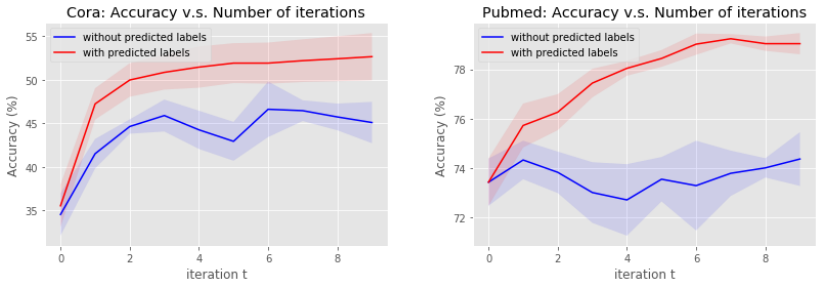
With or without random masking For the partially-labeled test data, we want to validate the necessity of applying the random node masking procedure, i.e. sampling a random mask for each epoch. To investigate this, we tested the performance of a model variant which only used a single fixed mask during training. We tried five binary masks and computed the average performance. Table 5.4 shows that when no random masking is applied, it achieved marginal gains or even hurt the performance. For example, *CL+GNN* with random masking achieved 2.30% improvement over GRAND, but the single mask variant

Table 5.4. Model performance on Cora (partially-labeled test data) where we vary the usage of two components. Both components add to the improvements of *CL+GNN*, and the ensemble method fails to improve GRAND.

Base GNN	description	random mask	prediction procedure			% improvement
			sampling from predicted label dist.	sampling from uniform label dist.	ensemble	
GCN	+ CL (ours)	✓	✓			+12.41 (1.96)
	+ CL w/o random mask		✓			+0.99 (1.03)
	+ CL w/o collective	✓				+8.44 (1.84)
	+ CL w/o predicted label	✓		✓		+10.82 (1.49)
	+ Ensemble	✓			✓	+11.18 (1.80)
TK	+ CL (ours)	✓	✓			+7.92 (0.75)
	+ CL w/o random mask		✓			+1.51 (1.63)
	+ CL w/o collective	✓				+5.06 (0.50)
	+ CL w/o predicted label	✓		✓		+6.05 (1.93)
	+ Ensemble	✓			✓	+6.13 (0.59)
GRAND	+ CL (ours)	✓	✓			+2.30 (0.56)
	+ CL w/o random mask		✓			-0.79 (1.32)
	+ CL w/o collective	✓				+1.21 (0.81)
	+ CL w/o predicted label	✓		✓		+0.40 (0.78)
	+ Ensemble	✓			✓	-0.60 (0.84)

decreased the performance by 0.79%. This could be because the model is biased to the single mask.

With or without predicted labels as input



(a) Cora, GCN (b) Pubmed, TK

Figure 5.3. *CL+GNN* performance with and without predicted labels on Cora and Pubmed. X-axis refers to iteration number t in Section 5.3.

To investigate if adding predicted labels in model input adds extra information with partially-labeled test data, we tested the performance of a model variant which only use true labels as input with the same node masking procedure. table 5.4 shows that the non-collective variant achieves some gains over the baseline GNN as the random masking procedure enables it to leverage test label information, but the improvement is much smaller

than the collective version. For example, when combined with GCN, *CL+GNN* with collective labels achieved 12.41% improvement while the non-collective version only achieved 8.44%. Moreover, Figure 5.3 shows two examples on Cora with GCN (Figure 5.3a) and Pubmed with TK (Figure 5.3b), where including predicted labels achieves better performance. We run the model 10 times and calculate the average and standard deviation (shown as shaded area) of classification accuracy at each iteration t as described in Section 5.3. We can see that adding predicted labels starts to improve the performance after the first iteration and achieves consistent gains.

Sampling from predicted labels or random ids

Creating more expressive GNN representations by averaging out random features was first proposed by [6]. [6] shows a whole-graph classification application, Circulant Skip Links (CSL) graphs, where such randomized feature averaging is provably (and empirically) more expressive than GNNs. Our Monte Carlo collective learning method can be seen as a type of feature averaging GNN representation though, unlike [6], the feature sampling is not at random, but rather driven by our own model recursively. Hence, it is fair to ask if our performance gains are simply because random feature averaging is beneficial to GNN representations? Or does collective learning sampling actually improve performance? We need an ablation study.

Therefore, we investigate whether the gains of our method for both *unlabeled test data* and *partially-labeled test data* are from incorporating feature randomness, or from sampling w.r.t predicted labels (collective learning). To do so, we replace the samples drawn from previous prediction $\hat{\boldsymbol{Y}}$ as uniformly drawn from the set of class labels *at each gradient step* in *CL+GNN*. Clearly, Table 5.3 and Table 5.4 show that the random features are not able to consistently improve the model performance as our method does. For example, when combined with TK, our method achieved a 1.54% improvement, while adding the random ids decreased the TK performance by 1.02%. In summary, collective learning goes beyond the purely randomized approach of [6], providing much larger, statistically significant, gains.

Comparison with ensemble approach

We tested an ensemble of 10 GNNs with random initialization, and tried GCN, TK and GRAND structures. Table 5.3 show that the ensemble approach was able to slightly improve

Table 5.5. Node classification accuracy varying number of **training labels** on Cora dataset. Numbers in bold represent significant improvement in a paired t-test at the $p < 0.05$ level.

# train labels		85 (3.21%)	140 (5.17%)	85 (3.21%)	140 (5.17%)
% test labels		0%	0%	50%	50%
GCN	-	64.74 (1.51)	66.19 (4.59)	66.34 (1.84)	69.10 (2.85)
	+ CL	**+3.72 (0.40)**	**+2.30 (1.10)**	**+12.41 (1.96)**	**+9.52 (2.36)**
TK	-	68.47 (1.31)	70.80 (1.51)	69.50 (0.55)	73.33 (1.01)
	+ CL	**+1.50 (0.61)**	**+0.89 (0.17)**	**+7.92 (0.75)**	**+5.17 (0.76)**
GRAND	-	71.55 (1.07)	74.75 (1.03)	73.19 (0.41)	75.81(1.09)
	+ CL	**+0.80 (0.31)**	+1.03 (0.50)	**+2.30 (0.56)**	**+1.10 (0.31)**

the GNN performance but the gains were consistently smaller than our proposed approach *CL+GNN*. For example, our approach achieved +3.72% gain on GCN while the ensemble approach only achieved +0.32%. Table 5.4 show that the ensemble approach has some gains with random masking when combined with GCN or TK. However, When combined with GRAND, ensemble method even slightly hurt the performance.

CL+GNN performance with varying training label rates and sample size K

To investigate the impact of the training label rates on the node classification accuracy, we repeated the experiments on Cora dataset with various numbers of training labels, on unlabeled test data and partially-labeled test data. Table 5.5 show the results for both test labels rates of 0% and 50%. We can see that in general *CL+GNN* framework achieved a larger improvement when fewer labels are available in the training graph. For example, with label rates of 1.52% and 3.04%, the improvements of our framework combining with GCN on unlabeled test data are 3.72%, 2.30% respectively. This shows that the *CL+GNN* framework is especially useful when only a small number of labels are available in training, which is the common use case of GNNs.

We also run some experiments varying K in $\{1, 5, 10, 20\}$ on Cora dataset with GCN and TK, and we found that with $K > 1$ there was consistent gain (see Fig. 5.4). We run each experiment in five trials (varying training nodes) as described in Section 5.5.1, and calculate the average and standard error (shown as shaded area).

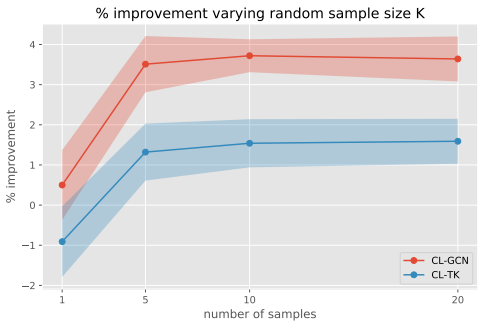


Figure 5.4. Impact of sample size K.

Complexity analysis. *CL+GNN* computes K embeddings at each stochastic gradient step, therefore, per-gradient step, *CL+GNN* is K slower than its component WL-GNN. Overall, after T iterations of Steps 1-3, *CL+GNN* total runtime increases by $T \times K$ over the original runtime of its component WL-GNN. The time and space complexity of *CL+GNN* is the same as WL-GNNs, i.e. $\mathcal{O}(m)$ where $m = |E|$.

Note that existing methods trying to boost the GNN expressiveness —e.g. PPGN [82], SMP [84])— are much more computationally expensive in time (at least $\Theta(mn)$) and space ($\Theta(n^2)$), where n and m are number of nodes and edges in the graph.

We note that we spent nearly no time engineering *CL+GNN* for speed or for improving our results. Our interest in this paper lies entirely on the gains of a direct application of collective learning to GNNs (*CL+GNN*). We fully expect that further engineering advances can reduce the computational burden due to Monte Carlo sampling and increase accuracy gains. For instance, parallelism can significantly reduce the time to collect K samples in *CL+GNN*.

5.6 Related Work

On collective learning and neural networks. There has been work on applying deep learning to collective classification. For example, [89] proposed to use LSTM-based RNNs for classification tasks on graphs. They transform each node and its set of neighbors into an unordered sequence and use an RNN to predict the class label as the output of that sequence.

[99] designed a deep learning model for collective classification in multi-relational domains, which learns local and relational features simultaneously to encode multi-relations.

The closest work to ours is [88], which proposed a recurrent collective classification (RCC) framework, a variant of ICA [11] including dynamic relational features encoding label information. Unlike our framework, this method does not sample labels $\hat{\boldsymbol{Y}}$, opting for an end-to-end training procedure. [91] opts for a similar no-sample RCC end-to-end training method as [88], now combining a differentiable graph kernel with an iterative stage. Graph Markov Neural Network (GMNN) [90] is another promising approach that applies statistical relational learning to GNNs. GMNNs model the joint label distribution with a conditional random field trained with the variational EM algorithm. GMNNs are trained by alternating between an E-step and an M-step, and two WL-GCNs are trained for the two steps respectively. These studies represent different ideas for bringing the power of collective classification to neural networks. Unfortunately, Corollary 1 shows that, without sampling $\hat{\boldsymbol{Y}}$, the above methods are still WL-GNNs, and hence, their use of collective classification fails to deliver any increase in expressiveness beyond an optimal WL-GNN (e.g., [8]). In our experiments, we compared to GMNN as a representative relational GNN method, and showed that while GMNN outperformed its component GCN, the best of *CL+GNN* still consistently out-performs GMNN.

In parallel to our work, [100] considers regression tasks by modeling the joint GNN residual of a target set $(y - \hat{y})$ as a multivariate Gaussian, defining the loss function as the marginal likelihood only over labeled nodes $\hat{y}_L$. In contrast, by using the more general foundation of collective classification, our framework can seamlessly model both classification and regression tasks, and include model predictions over the entire graph $\hat{\boldsymbol{Y}}$ as *CL+GNN*'s input, thus affecting both the model prediction and the GNN training in inductive node classification tasks.

Higher-order GNNs for more expressive graph representation Recently, there has been a few works proposed to boost the representation power of WLGNN [5], [81], [82], [84], [87], [101]. Most of these works consider representation for the entire graph or node sets by mimicking higher-order WL tests. However, most of them provide more theoretical implications for GNNs than practical usage due to their dependency on order-k tensors $\mathbb{R}^{n^k}$

(n : number of nodes, $k > 2$) and inability to leverage the sparsity of the graph structures. Among them PPGN [82] is relatively scalable with $\Theta(n^3)$ time complexity and $\Theta(n^2)$ space complexity to achieve the expressive power of the 2-WL test. A more recent method SMP [84] proposed a powerful and more scalable message-passing framework with time complexity of $\Theta(mn)$ (m : number of edges) and space complexity of $\Theta(n^2)$. Our work, on the other hand, focus on node-level representations rather than (sub)graph-level representations, and our overall time/space complexity is $\Theta(m)$. As these methods cannot be directly evaluated on node classification tasks and due to their computational inefficiency, we leave the assessment of *CL+GNN* gains if used with these more powerful GNN variants as future work.

On self-supervised learning and semi-supervised learning. Self-supervised learning is closely related to semi-supervised learning. In fact, self-supervised learning can be seen as a self-imposed semi-supervised learning task, where part of the input is masked (or transformed) and must be predicted back by the model [102]–[105]. Recently, self-supervised learning has been broadly applied to achieve state-of-the-art accuracy in computer vision [106], [107] and natural language processing [108] supervised learning tasks. The use of self-supervised learning in graph representation learning is intimately related to the use of pseudolikelihood to approximate true likelihood functions.

For further related work on collective classification, see **??**.

5.7 Conclusion

A long-standing question is when/if collective inference (CI) is needed when very expressive graph models are available (e.g., GNNs) for inductive node classification tasks. This work solves a few theoretical and empirical questions towards an answer. We show that, with the most expressive equivariant (node-embedding) GNNs, it is true that there is no need for collective learning.

While the development of more expressive GNNs generally focuses on changing the architecture, in this work we ask the question of whether CI could be a practical way to boost the real-world performance of a GNN, without changing its underlying architecture.

In this work we propose *collective learning* (CL), a modified CI approach for GNN-type classifiers that boosts their expressiveness, relying on both Monte Carlo sampling of node embeddings and (self-supervised) random masking in training. We show that collective learning can be combined with existing GNNs to improve their expressiveness (and we prove increased expressiveness with WL-GNNs).

We experimentally confirm our theoretical analysis across five real-world graphs and four component GNNs, and show by extensive empirical study that *CL+GNN* consistently, and significantly, boosts GNNs performance (up to 26%). One limitation of our proposed collective learning framework is the computational cost of using sampled embeddings during each stochastic gradient step. We leave exploration of mechanisms to reduce the additional computational burden (eg. via parallelization and/or more targeted sampling) to future work.

5.8 Appendix: Proofs of theorems

5.8.1 Proof of Theorem 5.4.1

We restate the theorem for completeness.

Theorem 5.4.1 (Collective classification can be unnecessary)**.** *Consider the task of predicting node labels when no labels are available in test data. Let* $\Gamma^\star(v, G^{(te)})$ *be a most-expressive representation of node* $v \in V^{(te)}$ *in graph* $G^{(te)}$ *. Then, for any collective learning procedure predicting the class label of* $v \in V^{(te)}$*, there exists a classifier that takes* $\Gamma^\star(v, G^{(te)})$ *as input and predicts the label of* v *with equal or higher accuracy.*

Proof. Let $\hat{Y}(v) = \varphi(\Gamma^\star(v, G))$ be a classifier function that takes the most expressive representation $\Gamma^\star(v, G)$ of node v as input and outputs a predicted class label for v.

Let $\hat{\boldsymbol{Y}}^{(t)}$ be the set of predicted labels at iteration t of collective classification and let Y_v be the true label of node $v \in V$. Then either (1) $Y_v \perp\!\!\!\perp_{\Gamma^\star(v,G),\varphi} \hat{\boldsymbol{Y}}^{(t)}$, or (2) $Y_v \not\perp\!\!\!\perp_{\Gamma^\star(v,G),\varphi} \hat{\boldsymbol{Y}}^{(t)}$.

Case (1): Given the classifier φ and the most expressive representation $\Gamma^\star(v, G)$, the true label of v is independent of the labels predicted with collective classification. In this case, the predicted labels of v's neighbors offer no additional information and, thus, collective classification is unnecessary.

Case (2): In this case, the true label of v is not independent of the predicted labels. By Theorem 1 of [7], we know that for any random variable H_v attached to node $v \in V$, it must be that $\exists \varphi'$ a measurable function independent of G s.t.

$$H_v \overset{\text{a.s.}}{=} \varphi'(\Gamma^\star(v, G), \epsilon_v),$$

where ϵ_v is an noise source exogenous to G (pure noise), and a.s. implies almost sure equality. Defining $H_v := Y_v$,

$$\varphi'(\Gamma^\star(v, G), \epsilon_v) \not\perp\!\!\!\perp_{\Gamma^\star(v,G),\varphi} \hat{\boldsymbol{Y}}^{(t)},$$

which means $\hat{\boldsymbol{Y}}^{(t)}$ must either be dependent on ϵ_v or contain domain knowledge information about the function φ' that is not in φ. Since $\hat{\boldsymbol{Y}}^{(t)}$ is a vector of random variables fully determined by G and φ, it cannot depend on an exogenous variable ϵ_v, Thus, the predictions must contain domain knowledge of φ'. Hence, we can directly incorporate this domain knowledge into another classifier $\varphi^\dagger$ s.t. $Y_v \perp\!\!\!\perp_{\Gamma^\star(v,G),\varphi^\dagger} \hat{\boldsymbol{Y}}^{(t)}$, for instance $\varphi^\dagger$ is a function of φ'. In this case, $\varphi^\dagger$ will predict the label of v with equal or higher accuracy than collective classification based on predicted labels $\hat{\boldsymbol{Y}}$, which finishes our proof.

□

5.8.2 Proof of Theorem 5.4.2

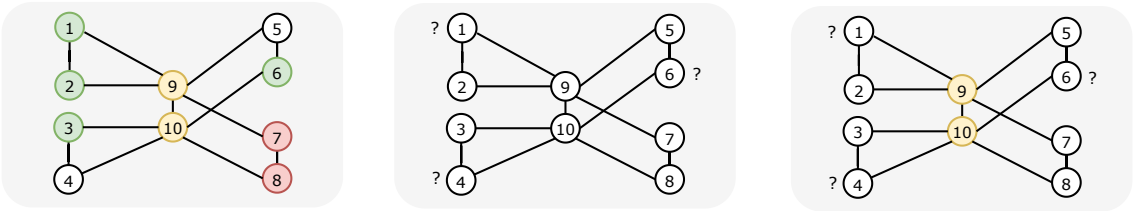


Figure 5.5. Training/testing graphs. Colors represent available node labels, and testing nodes are marked with question marks. WL-GNN cannot differentiate between the red and green nodes.

Theorem 5.4.2 (*CL+GNN** expressive power)**.** *Let WLGNN** *be an optimal WL-GNN. Then, the collective learning representation of Equation (5.2), using WLGNN** *as the GNN component, (denoted* CL+GNN**) is strictly more expressive than this WLGNN** *representation model applied to the same tasks.*

Proof. As defined, WLGNN is a most-expressive WL-GNN. We need to prove $\mathcal{G}(\text{WL-GNN}) \subsetneq \mathcal{G}(CL{+}GNN)$. We will do that by first showing $\mathcal{G}(CL{+}GNN) = \mathcal{G}(\text{WL-GNN}) \cup S'$ and then showing that $\exists G \in \mathcal{G}(CL{+}GNN)$ s.t. $G \notin \mathcal{G}(\text{WLGNN})$.

$\mathcal{G}(\text{CL+GNN}) = \mathcal{G}(\,WL\text{-}GNN) \cup S'$: First, we need to show that for any mask $M \in \mathcal{M}$, $\exists S \subseteq \mathcal{G}(CL{+}GNN)$ such that $S = \mathcal{G}(\text{WL-GNN})$. This is clearly true since, for labeled tests, in Equation (5.2) we can always construct a WLGNN^0 for a *CL+GNN*

$$\boldsymbol{Z}_v^{(t)}(\text{WLGNN}^0) = \mathbb{E}_{\hat{\boldsymbol{Y}}^{(t-1)}}\Big[\text{WLGNN}^0(\boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})} \odot M, \hat{\boldsymbol{Y}}^{(t-1)} \odot \overline{M}, \boldsymbol{A}^{(\text{tr})}; \Theta)_v\Big], \tag{5.8}$$

that ignores the $\hat{\boldsymbol{Y}}$ inputs. Similarly, for unlabeled tests, in Equation (5.2) we can always construct a WLGNN^1 for a *CL+GNN*

$$\boldsymbol{Z}_v^{(t)}(\text{WLGNN}^1) = \mathbb{E}_{\hat{\boldsymbol{Y}}^{(t-1)}}\left[\text{WLGNN}^1(\boldsymbol{X}^{(\text{tr})}, \hat{\boldsymbol{Y}}^{(t-1)}, \boldsymbol{A}^{(\text{tr})}; \Theta)_v\right],$$

that ignores the $\hat{\boldsymbol{Y}}^{(t-1)}$ inputs.

$\exists G \in \mathcal{G}(\text{CL+GNN})$ *s.t.* $G \notin \mathcal{G}(\,WL\text{-}GNN)$: Let G be the graph in Figure 5.5. We will first consider the case where the test data has partial labels. The case without labels follows directly from it. Using the graph G in Figure 5.5(a) (training) and Figure 5.5(c) (partially-labeled testing) we show that a WL-GNN is unable, in test, to correctly give representations to the left-most nodes $\{1, 2, 3, 4\}$ that are distinct from the right-most nodes $\{5, 6, 7, 8\}$ (the same happens for the unlabeled test graph in Figure 5.5(b)). We then show that the representation $\boldsymbol{Z}_v^{(t)}$ of Equation (5.8) is able to distinguish these two sets of nodes.

WL-CNN is not powerful enough to give distinct representations to nodes $\{1, \ldots, 8\}$ *in Figure 5.5(c)*: Consider giving an arbitrary feature value (say, the "color white") to all uncolored nodes $\{1, \ldots, 8\}$ in Figure 5.5(c). We will start showing that the 1-WL test is unable to give different colors to the nodes $\{1, \ldots, 8\}$ in this graph. Since WL-GNNs are no

more expressive than the 1-WL test [5], [8], showing that the above is a stable coloring for nodes $\{1,\ldots,8\}$ in the 1-WL test, proves the first part of our result. A stable 1-WL coloring is defined as a coloring scheme on the graph that has a 1-to-1 correspondence with the colors in the previous step of the 1-WL algorithm. Since the input to the hash function of the 1-WL test is the same for all of nodes $v \in \{1,\ldots,8\}$: The node itself has color white while the color set of the neighbors is the set $\{\text{white}, \text{yellow}\}$. In the next 1-WL round, all the white nodes will be mapped to the same color by the hash function. The colors of node $\{9, 10\}$ will be not the same as $\{1,\ldots,8\}$. Hence, the initial coloring of all nodes $\{1,\ldots,8\}$ white and $\{9, 10\}$ yellow is a stable coloring for 1-WL. Consequently, WL-GNN will give the same representation to all nodes in $\{1,\ldots,8\}$.

CL+GNN gives the same representations within the sets $\{1,\ldots,4\}$ *and* $\{5,\ldots,8\}$: At iteration $t \geq 0$ of *CL+GNN*, we start with the base of the recursion $\boldsymbol{Y}^{(t-2)} = \mathbf{0}$.

Now consider a given mask $\boldsymbol{M}^{(t)} \in \mathcal{M}$. Note that to sample $\hat{\boldsymbol{Y}}_v^{(t-1)}$ for $v \in \{1,\ldots,8\}$ we apply $\hat{\boldsymbol{Y}}^{(t-2)} = \mathbf{0}$ into Equation (5.2) to obtain $\boldsymbol{Z}_v^{(t-1)}$, and then apply $\boldsymbol{Z}_v^{(t-1)}$ into Equation (5.4), defining $\boldsymbol{X}^{(\text{tr})}_{\boldsymbol{Y}_L^{(\text{tr})},\hat{\boldsymbol{Y}}^{(t-1)},\boldsymbol{M}^{(t)}} = [\boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})} \odot \boldsymbol{M}^{(t)}, \mathbf{0}]$, which will give us $\hat{\boldsymbol{Y}}_v^{(t-1)} \sim P(\boldsymbol{Y}_v^{(t-1)} \,|\, \text{WLGNN}([\boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})} \odot \boldsymbol{M}^{(t)}, \mathbf{0}], \boldsymbol{A}^{(\text{tr})}; \Theta)_v)$, and any classes has a non-zero probability of being sampled since our output is a softmax.

Since nodes $\{1,\ldots,8\}$ all get the same representation in the above WLGNN, their respective sampled $\hat{\boldsymbol{Y}}_v^{(t-1)}$, $v \in \{1,\ldots,8\}$, will have the same distribution but possibly not the same values (due to sampling). Note that the nodes $\{1,\ldots,4\}$ will get the same average in Equation (5.8) since $\hat{\boldsymbol{Y}}_v^{(t-1)}$, $v \in \{1,\ldots,4\}$, has the same distribution and the nodes are isomorphic (even given the colors on nodes 9 and 10). Similarly, the nodes $\{5,\ldots,8\}$ will also get the same average in Equation (5.8).

CL+GNN gives distinct representations accross the sets $\{1,\ldots,4\}$ *and* $\{5,\ldots,8\}$: Finally, we now prove that exists a WL-GNN, which we will denote WLGNN^2, such that $\boldsymbol{Z}_v^{(t)}(\text{WLGNN}^2) \neq \boldsymbol{Z}_u^{(t)}(\text{WLGNN}^2)$ for $v \in \{1,\ldots,4\}$ and $u \in \{5,\ldots,8\}$. We will show that there is a joint sample of $\hat{\boldsymbol{Y}}_1^{(t-1)},\ldots,\hat{\boldsymbol{Y}}_8^{(t-1)}$ where there is no symmetry between the representations of nodes in $\{1,\ldots,4\}$ and $\{5,\ldots,8\}$. Since each layer of WLGNN^2 can have different parameters, we can easily encode differences in the number of hops it takes to reach a certain color. Moreover, at any WLGNN^2 layer, the representation of a node can perfectly

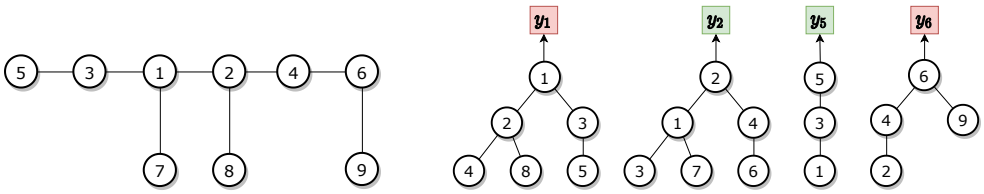
encode its own last-layer representation and the last-layer representation of its neighbors through a most-expressive multiset representation function [8].

It is enough for us to show that for a sampled $\hat{\boldsymbol{Y}}^{(t-1)}$ the sets of nodes $\{1,\ldots,4\}$ and $\{5,\ldots,8\}$ can get distinct unique representations under WLGNN^2. By unique, we mean, $\{1,\ldots,4\}$ can get representations in WLGNN^2 that cannot be obtained by the nodes in $\{5,\ldots,8\}$. This representation uniqueness makes sure the averages in Equation (5.2) are different. Without loss of generality we will consider giving a special sampled label to only one node i $\in \{1,5\}$ in one of the sets. The sampled labels $\hat{\boldsymbol{Y}}_{\text{i}}^{(t-1)}$ = green, while all other nodes $\{1,\ldots,8\}\backslash\{\text{i}\}$ get red, will happen with non-zero probability, hence, they must be part of the expectation in Equation (5.2). Note that node 2 (for i = 1) and 6 (for i = 5) will feel the effects of the green color in their neighbors differently. That is, for i = 5 there is a parameter choice for the layers of WLGNN^2 where the representation of node 6 uniquely encodes that the color green is within hops 1 (node 5) and 3 (from node 5 through nodes 9 and 10) of node 6 (if 6 treats its own representation differently from its neighbors). For i = 1, node 2's representation will encode that green is observed hops 1 (node 1) and 2 (from node 1 through node 9) (similarly, 2 treats its own representation differently from its neighbors). Hence, these representations can be made unique by WLGNN^2, i.e., no other $\hat{\boldsymbol{Y}}^{(t-1)}$ assignments will create the same patterns for nodes 2 and 6, and thus, since WLGNN^2 has most-expressive multiset representations, it can give a unique representation to nodes 2 and 6 for these two unique $\hat{\boldsymbol{Y}}^{(t-1)}$ configurations. These unique representations are enough to ensure $\boldsymbol{Z}_2^{(t)}(\text{WLGNN}^2) \neq \boldsymbol{Z}_6^{(t)}(\text{WLGNN}^2)$ for any $t \geq 1$, which concludes our proof.

□

5.8.3 Proof of Proposition 5.4.1

Proposition 5.4.1. *Let G_v^d be the d-hop egonet of a node v in graph G with diameter $D > d$. Let v_1 and v_2 be two non-isomorphic nodes whose d-hop egonets are isomorphic (i.e., $G_{v_1}^d$ is isomorphic to $G_{v_2}^d$) but $2d$-hop egonets are not isomorphic. Then, a WL-GNN representation with d layers will generate identical representations for v_1 and v_2 while* CL+GNN *is capable of giving distinct node representations.*



(a) Training graph (b) 2^{nc}-order neighborhood for label prdiction

Figure 5.6. WL-GNN using 2nd-order neighborhood cannot differentiate node 1 and 2, but *CL+GNN* built on this WLGNN can break the local isomorphism.

Proof. Let G be the graph in Figure 5.6(a) with no node features, and let WLGNN be of order 2, meaning it will generate node embeddings based on 2nd-order neighborhoods (shown in (b)). Since node 1 and 2 have the same 2nd-order neighborhood structure, WLGNN will generate identical node representation for them, which gives random label predictions. Meanwhile, as nodes 5 and 6 have distinct 2nd-order neighborhood structures, WLGNN generates different node representations for them, which enables the model to learn from the labels y_5 and y_6. We can assume the predicted label probability $P(\hat{\boldsymbol{Y}}_5^{(0)} = \text{green}) = 0.99$ and $P(\hat{Y}_6^{(0)} = \text{red}) = 0.98$. For *CL+GNN*, at iteration $t = 1$, we sample $\hat{\boldsymbol{Y}}^{(0)}$ from the WLGNN output $P(\hat{\boldsymbol{Y}}^{(0)})$ and use the samples as input. In the worst case, nodes $3, 4$ and $7, 8$ get the same distribution and sampled labels (i.e. $\hat{\boldsymbol{Y}}_3^{(0)} = \hat{\boldsymbol{Y}}_4^{(0)}$, $\hat{\boldsymbol{Y}}_7^{(0)} = \hat{\boldsymbol{Y}}_8^{(0)}$). Since the distribution of $\hat{\boldsymbol{Y}}_5^{(0)}$ and $\hat{\boldsymbol{Y}}_6^{(0)}$ are different, the samples of $\hat{\boldsymbol{Y}}_5^{(0)}$ and $\hat{\boldsymbol{Y}}_6^{(0)}$ are different, which breaks the tie between the 2nd-order neighborhood of nodes 1 and 2. Therefore, *CL+GNN* will produce different node representation starting from iteration $t = 1$ for nodes 1 and 2, which enables the model to learn from the training label y_1 and y_2, and thus gives more accurate predictions. □

The advantage of collective inference is more clear when it is used to strengthen less-expressive local classifiers, e.g. logistic regression Although GNN are much powerful than these local classifiers by aggregating high(er)-order graph information, collective learning can still help if GNN fail to make use of "global" information in graphs (or equivalently, if the order of GNN is small than graph diameter). Previous work [109] investigating the power of collective inference also showed that methods for collective inference benefit from *a clever*

factoring of the space of dependencies, by arguing that *these (collective inference) methods benefit from information propagated from outside of their local neighborhood. Predictions about the class label on other objects essentially "bundle information" about the graph beyond the immediate neighborhood.*

5.8.4 Proof of Proposition 5.4.2

Proposition 5.4.2. *If* $\forall v \in V_L^{(tr)}$, $\nabla_\Theta(\mathbf{W}\boldsymbol{Z}_v^{(t)}(\boldsymbol{M}^{(t)};\Theta))_{y_v^{(tr)}}$ *is bounded (e.g., via gradient clipping), then the optimization in Equation (5.3), with the unbiased sampling of* $\{\boldsymbol{Z}_v^{(t-1)}\}_{v\in V^{(tr)}}$ *and* $\boldsymbol{M}^{(t)}$ *described above, results in a Robbins-Monro [92] stochastic optimization algorithm that optimizes a surrogate upper bound of the loss in Equation (5.3).*

Proof. In our optimization, we only need to sample two variables $\hat{\boldsymbol{Y}}^{(t-1)}$ and $\boldsymbol{M}^{(t)}$. We obtain unbiased bounded-variance estimates of the derivative of the loss function if we sample $\boldsymbol{M}^{(t)} \sim \text{Uniform}(\mathcal{M})$ (and exact values when $\boldsymbol{M}^{(t)} = \mathbf{0}$). We can now compound that with unbiased bounded-variance estimates of the derivative if we estimate the expectation in Equation (5.2) $\{\boldsymbol{Z}_v^{(t-1)}\}_{v\in V}$ by i.i.d. sampling $\hat{\boldsymbol{Y}}^{(t-1)}$. The loss in Equation (5.3) is convex on $\boldsymbol{Z}_v^{(t)}$ since the negative log-likelihood of the multi-class logistic regression is convex on $\mathbf{W}$, which means it is also convex on $\boldsymbol{Z}_v^{(t)}$ as the loss is defined on the affine transformation $\mathbf{W}\boldsymbol{Z}_v^{(t)}$. The expectation of the loss always exist, since we assume $\nabla_\Theta(\mathbf{W}\boldsymbol{Z}_v^{(t)}(\boldsymbol{M}^{(t)};\Theta))_{y_v^{(\text{tr})}}$ is bounded for all $\forall v \in V_L^{(\text{tr})}$. Hence, as the loss is convex w.r.t. $\boldsymbol{Z}_v^{(t)}$, the expection w.r.t. $\boldsymbol{Z}_v^{(t)}$ exists, and we obtain an unbiased estimate of $\boldsymbol{Z}_v^{(t)}$, we can apply Jensen's inequality to show that the resulting Robbins-Monro stochastic optimization optimizes an upper bound of the loss in Equation (5.3). □

6. SUMMARY AND FUTURE DIRECTIONS

6.1 Summary

In this dissertation, we developed graph embedding and graph neural network frameworks to capture various aspects of graph structures for unsupervised and semi-supervised tasks. In this chapter, we summarize the contributions of this dissertation.

The contributions of this dissertation fall into the following aspects:

Theoretical

- In Chapter 5, we
  - formalized the notion of representation-based automorphic/regular equivalence, for single graphs and multiple aligned graphs;
  - formalized the extended notion of (exact) regular equivalence and probabilistic structural property for multiple aligned graphs;
  - analyzed the expressive power of the proposed method *Reqe* w.r.t structural properties and number of samples (for multiple graphs cases) $\longrightarrow$ Theorem 4.5.1, Theorem 4.5.2
- In Chapter 4, we showed:
  - Collective classification is provably unnecessary for GNNs that are most-expressive (Theorem 5.4.1);
  - Collective learning boost the expressiveness of the optimal WL-GNN (Theorem 5.4.2) and practical WL-GNNs (Proposition 5.4.1).
  - Previous attempts to incorporate collective inference into WL-GNNs (which in contrast to CL+GNN do not Monte Carlo sample embeddings) cannot increase expressivity beyond that of an optimal WL-GNN (Corollary 1).

Algorithmic/Methodological

- In Chapter 3, we developed a node embedding method (detailed in Algorithm 1) for heterogeneous information network-based with modified negative sampling approach.

- In Chapter 4, we developed a flexible framework *Reqe* to learn structural node embeddings w.r.t specified structural properties and node attributes, which handles both single graphs (see Algorithm 3) and multiple aligned graphs (see Algorithm 4).
- In Chapter 5, we developed an add-on collective learning framework which uses self-supervised learning and Monte Carlo sampled embeddings to incorporate node labels during inductive learning—and it can be implemented with *any* component GNN (see Section 5.3).

Empirical

- In Chapter 3, we
  - Presented the first educational "check-in" dataset and explores its unique mobility and social characteristics;
  - Identified the challenges for time-aware POI prediction in educational check-in data based on increased density due to location-based tracking (compared to previous voluntary-report LSBN data);
  - Applied the proposed node embedding model on the educational "check-in" dataset showed its effectiveness on two real-world tasks: time-aware POI prediction and friend suggestion.
- In Chapter 4, we empirically evaluated our proposed model *Reqe* on three tasks (equivalence encoding, structural property prediction, and node classification) on seven graph datasets from a variety of application areas.
- In Chapter 5, we
  - designed a graph sampling approach to split each single graph into three clusters for training, validation, and testing respectively — shown in Figure 5.2 (left), which mimiced the inductive within-graph scenario that often occurs in real world settings;

– evaluated the proposed collective learning framework *CL+GNN* across a variety of state-of-the-art WL-GNNs, for inductive tasks involving unlabeled and partially-labeled test graphs.

6.2 Implications and Future Directions

6.2.1 On node expressivity

One central problem in learning node embeddings is to capture node isomorphism in graphs with embeddings. In this dissertation we introduced two effective methods towards more expressive node embedding for semi-supervised and unsupervised tasks respectively. In Chapter 4, we described *Reqe*, a node embedding framework to capture automorphic equivalence by including structural properties. In Chapter 5, we introduced a collective learning framework to boost GNN expressiveness for semi-supervised node classification problem.

As shown in our experiments, the GNN-type models (e.g. GraphSAGE) may not perform well in predicting role-related labels when no node attributes are provided, this indicates that there is extra information in structural properties that the neighbor propagation process in GNNs is unable to capture, especially with limited number of layers in practice.

Recently, there has been increasing attention to boosting the representation power of WLGNN [5], [81], [82], [84], [87], [101]. Most of these works consider representation for the entire graph or node sets by mimicking higher-order WL tests. However, most of them provide more theoretical implications for GNNs than practical usage due to their dependency on order-k tensors $\mathbb{R}^{n^k}$ (n : number of nodes, $k > 2$). Before we can get to the most expressive GNNs, there is still large room for improvement, and we believe there could be multiple ways to boost the expressivity, e.g. ensemble learning.

6.2.2 On ensemble learning involving multiple embeddings

In Chapter 4, we proposed a method to learn a single embedding for each node in multiple graph samples, aiming to capture the underlying graph distribution. In Chapter 5, the collective learning procedures involve generating multiple samples of embeddings on

graphs with random masking. Both approaches reflected the concept of ensemble learning (specifically, bagging) – multiple models are fitted on slightly different data samples. In *CL+GNN* work, we compared with an ensemble of 10 GNN models with random initialization and random masking, and we found that the ensemble approach was able to improve over the baseline GNN performance, especially when the GNN is not very expressive, e.g. GCN. This implies that using ensemble of GNNs, or combine multiple embeddings on perturbed graphs are effective ways towards a more robust model, especially with limited computation resource and performing collective procedures is too expensive.

There has been some ongoing work on ensemble learning methods for neural networks including GNNs [110]–[112]. For example, [111] proposed to using a diversified ensemble layer as a building block for combining multiple neural networks, and jointly train them by maximizing their diversity and minimizing their own prediction loss. There are very few methods for end-to-end ensemble learning especially in the relational data domain, which is worth further attention towards more robust graph embeddings or GNN models. One promising direction would be learning an ensemble of GNNs that differ in model structure, e.g. different neighbor aggregators, and the random masking procedure can be applied to generate graph samples.

6.2.3 GNNs for link prediction in heterogeneous graphs

The heterogeneous graph embedding work for POI recommendation (Chapter 3) was done before GNNs became popular. There has been increasing interest in applying GNN-type models on unattributed and unlabeled graphs for link prediction tasks. e.g. VGAE [113] uses a GCN as encoder to learn node embeddings that best reconstruct the network. R-GCN [114] extends VGAE to the multi-relational graph cases, with application in knowledge graphs. On the other hand, graph-level GNNs are also developed for the same link prediction tasks, where local enclosing subgraphs are used as input for classification (e.g. SEAL [115]). Much of these works either considered homogeneous graphs or evaluated with knowledge graphs (e.g. AIFB [116]) and event-based database (e.g. ICEWS [117]). Special care needs to be taken when

such methods are applied to temporal interaction graphs (e.g. email/chat communication, researcher collaboration) with different dynamic patterns.

Recently, there has been some work along this line of research (e.g. [118]), capturing the node and edge heterogeneity and interaction patterns in dynamic graphs with GNNs remains an under-exploited area.

6.2.4 On efficient collective learning GNNs

One limitation of our proposed collective learning framework is the computational cost of using sampled embeddings during each stochastic gradient step, as our primary goal was to give a theoretical and empirical answer to a long-standing question about CI and GNNs without any effort to reduce the computational cost.

The time/space complexity of the *CL+GNN* is K (K: sample size) times the time/space complexity of the corresponding GNN model as we have to compute K representations for each node at each stochastic gradient step. Future work could leverage the exploration of mechanisms to reduce the additional computational burden (eg. via parallelization and/or more targeted sampling).

6.2.5 Node equivalence on dynamic graphs

In our work on capturing node equivalence patterns for multiple graphs, we considered K i.i.d. samples of graphs, which ignored the order of graphs/edges. In many real-world scenarios, dynamic or temporal graphs contains rich(er) and complicated node interaction patterns. While clustering [119] and community detection tasks [120], [121] on dynamic graphs has been investigated in the literature, much of these methods did not leverage graph embedding methods or GNNs. How to formally define role-based equivalence on temporal graphs or ordered graph samples, and how to capture it with an embedding model remains an open question.

REFERENCES

[1] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[2] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[3] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[4] D. Koller, N. Friedman, S. Džeroski, C. Sutton, A. McCallum, A. Pfeffer, P. Abbeel, M.-F. Wong, D. Heckerman, C. Meek, *et al.*, *Introduction to statistical relational learning*. MIT press, 2007.

[5] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4602–4609.

[6] R. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, "Relational pooling for graph representations," in *International Conference on Machine Learning*, PMLR, 2019, pp. 4663–4673.

[7] B. Srinivasan and B. Ribeiro, "On the equivalence between node embeddings and structural graph representations," *arXiv preprint arXiv:1910.00452*, 2019.

[8] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[9] R. Xiang and J. Neville, "Pseudolikelihood em for within-network relational learning," in *2008 Eighth IEEE International Conference on Data Mining*, IEEE, 2008, pp. 1103–1108.

[10] J. Neville and D. Jensen, "Iterative classification in relational data," in *Proc. AAAI-2000 workshop on learning statistical models from relational data*, 2000, pp. 13–20.

[11] Q. Lu and L. Getoor, "Link-based classification," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 496–503.

[12] B. Perozzi, V. Kulkarni, H. Chen, S. Skiena, and S. Don't Walk, "Online learning of multi-scale network embeddings," in *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2017, pp. 258–265.

[13] S. Ganguly, M. Gupta, V. Varma, V. Pudi, *et al.*, "Author2vec: Learning author representations by combining content and link information," in *Proceedings of the 25th International Conference Companion on World Wide Web*, International World Wide Web Conferences Steering Committee, 2016, pp. 49–50.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[15] Y. Dong, N. V. Chawla, and A. Swami, "Metapath2vec: Scalable representation learning for heterogeneous networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2017, pp. 135–144.

[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[17] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[18] F. Lorrain and H. C. White, "Structural equivalence of individuals in social networks," *The Journal of mathematical sociology*, vol. 1, no. 1, pp. 49–80, 1971.

[19] J. Scott, *Social Networks: Critical Concepts in Sociology*, ser. Critical concepts in sociology v. 2. Routledge, 2002, ISBN: 9780415251099. [Online]. Available: https://books.google.com/books?id=cYnSbKsmsrcC.

[20] M. G. Everett and S. P. Borgatti, "Regular equivalence: General theory," *Journal of mathematical sociology*, vol. 19, no. 1, pp. 29–52, 1994.

[21] M. G. Everett and S. Borgatti, "Calculating role similarities: An algorithm that helps determine the orbits of a graph," *Social networks*, vol. 10, no. 1, pp. 77–91, 1988.

[22] R. A. Rossi and N. K. Ahmed, "Role discovery in networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 4, pp. 1112–1131, 2014.

[23] M. G. Everett, "Role similarity and complexity in social networks," *Social Networks*, vol. 7, no. 4, pp. 353–359, 1985.

[24] R. Jin, V. E. Lee, and L. Li, "Scalable and axiomatic ranking of network role similarity," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 8, no. 1, pp. 1–37, 2014.

[25] T.-F. Fan and C.-J. Liau, "Logical characterizations of regular equivalence in weighted social networks," *Artificial Intelligence*, vol. 214, pp. 66–88, 2014.

[26] U. Brandes and J. Lerner, "Structural similarity: Spectral methods for relaxed blockmodeling," *Journal of classification*, vol. 27, no. 3, pp. 279–306, 2010.

[27] J. I. Casse, C. R. Shelton, and R. A. Hanneman, "A new criterion function for exploratory blockmodeling for structural and regular equivalence," *Social networks*, vol. 35, no. 1, pp. 31–50, 2013.

[28] J. Neville, D. Jensen, and B. Gallagher, "Simple estimators for relational bayesian classifiers," in *Third IEEE International Conference on Data Mining*, IEEE, 2003, pp. 609–612.

[29] A. Popescul, L. H. Ungar, S. Lawrence, and D. M. Pennock, "Towards structural logistic regression: Combining relational and statistical learning," *Departmental Papers (CIS)*, p. 134, 2002.

[30] J. J. Pfeiffer III, J. Neville, and P. N. Bennett, "Overcoming relational learning biases to accurately predict preferences in large scale networks," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 853–863.

[31] C. Yang, L. Bai, C. Zhang, Q. Yuan, and J. Han, "Bridging collaborative filtering and semi-supervised learning: A neural approach for poi recommendation," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2017, pp. 1245–1254.

[32] Z. Yao, "Exploiting human mobility patterns for point-of-interest recommendation," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, ACM, 2018, pp. 757–758.

[33] J. He, X. Li, L. Liao, D. Song, and W. K. Cheung, "Inferring a personalized next point-of-interest recommendation model with latent behavior patterns.," in *AAAI*, 2016, pp. 137–143.

[34] H. Yin, W. Wang, H. Wang, L. Chen, and X. Zhou, "Spatial-aware hierarchical collaborative deep learning for poi recommendation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 11, 2017.

[35] W. Wang, H. Yin, S. Sadiq, L. Chen, M. Xie, and X. Zhou, "Spore: A sequential personalized spatial item recommender system," in *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, IEEE, 2016, pp. 954–965.

[36] A. Noulas, S. Scellato, C. Mascolo, and M. Pontil, "An empirical study of geographic user activity patterns in foursquare.," *ICwSM*, vol. 11, no. 70-573, p. 2, 2011.

[37] S. Kylasa, G. Kollias, and A. Grama, "Social ties and checkin sites: Connections and latent structures in location-based social networks," *Social Network Analysis and Mining*, vol. 6, no. 1, 2016.

[38] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *Proceedings of the 18th international conference on World wide web*, ACM, 2009, pp. 791–800.

[39] Y. Sun, H. Yin, and X. Ren, "Recommendation in context-rich environment: An information network analysis approach," in *Proceedings of the 26th International Conference on World Wide Web Companion*, International World Wide Web Conferences Steering Committee, 2017, pp. 941–945.

[40] J. Tang, M. Qu, and Q. Mei, "Pte: Predictive text embedding through large-scale heterogeneous text networks," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2015, pp. 1165–1174.

[41] C. Zhang, K. Zhang, Q. Yuan, F. Tao, L. Zhang, T. Hanratty, and J. Han, "React: Online multimodal embedding for recency-aware spatiotemporal activity modeling," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, 2017, pp. 245–254.

[42] T. Chen and Y. Sun, "Task-guided and path-augmented heterogeneous network embedding for author identification," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, ACM, 2017, pp. 295–304.

[43] S. Feng, X. Li, Y. Zeng, G. Cong, Y. M. Chee, and Q. Yuan, "Personalized ranking metric embedding for next new poi recommendation.," in *IJCAI*, 2015, pp. 2069–2075.

[44] M. Xie, H. Yin, H. Wang, F. Xu, W. Chen, and S. Wang, "Learning graph-based poi embedding for location-based recommendation," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, ACM, 2016, pp. 15–24.

[45] A. Q. Li, A. Ahmed, S. Ravi, and A. J. Smola, "Reducing the sampling complexity of topic models," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2014, pp. 891–900.

[46] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, 2011, pp. 693–701.

[47] L. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, 2008.

[48] W. Wang, H. Yin, L. Chen, Y. Sun, S. Sadiq, and X. Zhou, "Geo-sage: A geographical sparse additive generative model for spatial item recommendation," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2015, pp. 1255–1264.

[49] L. Chen, F. Yuan, J. M. Jose, and W. Zhang, "Improving negative sampling for word representation using self-embedded features," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, ACM, 2018, pp. 99–107.

[50] R. A. Hanneman and M. Riddle, *Introduction to social network methods*, 2005.

[51] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li, "Rolx: Structural role extraction & mining in large graphs," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 1231–1239.

[52] K. Tu, P. Cui, X. Wang, P. S. Yu, and W. Zhu, "Deep recursive network embedding with regular equivalence," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, 2018, pp. 2357–2366.

[53] 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.

[54] S. Borgatti, M. G. Everett, and L. C. Freeman, "Ucinet iv version 1.64," *Natick, MA: Analytic Technologies*, 1996.

[55] M. G. Everett and S. P. Borgatti, "Exact colorations of graphs and digraphs," *Social networks*, vol. 18, no. 4, pp. 319–331, 1996.

[56] V. Batagelj, A. Ferligoj, and P. Doreian, "Direct and indirect methods for structural equivalence," *Social networks*, vol. 14, no. 1-2, pp. 63–90, 1992.

[57] M. Everett and S. Borgatti, "The regular coloration of graphs," in *INSTITUTE OF MATHEMATICS AND ITS APPLICATIONS CONFERENCE SERIES*, vol. 60, 1997, p. 49.

[58] S. Huang, T. Lv, X. Zhang, Y. Yang, W. Zheng, and C. Wen, "Identifying node role in social network based on multiple indicators," *PLOS ONE*, vol. 9, no. 8, pp. 1–16, Aug. 2014. DOI: 10.1371/journal.pone.0103733. [Online]. Available: https://doi.org/10.1371/journal.pone.0103733.

[59] T. Haveliwala, "Efficient computation of pagerank," Stanford, Tech. Rep., 1999.

[60] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[61] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "Struc2vec: Learning node representations from structural identity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2017, pp. 385–394.

[62] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan, "Subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs," *arXiv preprint arXiv:1606.08928*, 2016.

[63] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec, "Learning structural node embeddings via diffusion wavelets," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, 2018, pp. 1320–1329.

[64] J. Guo, L. Xu, and J. Liu, "Spine: Structural identity preserved inductive network embedding," in *Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019.

[65] N. K. Ahmed, R. Rossi, J. B. Lee, T. L. Willke, R. Zhou, X. Kong, and H. Eldardiry, "Learning role-based graph embeddings," *StarAI workshop - IJCAI*, 2018.

[66] R. A. Rossi, N. K. Ahmed, E. Koh, S. Kim, A. Rao, and Y. Abbasi-Yadkori, "A structural graph representation learning framework," in *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM)*, 2020, pp. 1–9.

[67] K. Henderson, B. Gallagher, L. Li, L. Akoglu, T. Eliassi-Rad, H. Tong, and C. Faloutsos, "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.

[68] C. Aicher and A. Z. e. a. Jacobs, "Adapting the stochastic block model to edge-weighted networks," *arXiv preprint arXiv:1305.5782*, 2013.

[69] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic Network Embedding by Modelling Triadic Closure Process," in *AAAI*, 2018.

[70] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[71] J. Leskovec and J. J. Mcauley, "Learning to discover social circles in ego networks," in *Advances in neural information processing systems*, 2012, pp. 539–547.

[72] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[73] M. Hang, I. Pytlarz, and J. Neville, "Exploring student check-in behavior for improved point-of-interest prediction," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 321–330.

[74] J. Bayer, H. Bydzovská, J. Géryk, T. Obsivac, and L. Popelinsky, "Predicting drop-out from social behaviour of students.," *International Educational Data Mining Society*, 2012.

[75] H. Park and J. Neville, "Exploiting interaction links for node classification with deep graph neural networks.," in *IJCAI*, 2019, pp. 3223–3230.

[76] K. Krishna and M. N. Murty, "Genetic k-means algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 3, pp. 433–439, 1999.

[77] H. Chen, H. Yin, T. Chen, Q. V. H. Nguyen, W.-C. Peng, and X. Li, "Exploiting centrality information with graph convolutions for network representation learning," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, IEEE, 2019, pp. 590–601.

[78] M. Hang, J. Neville, and B. Ribeiro, "A collective learning framework to boost gnn expressiveness for node classification," in *International Conference on Machine Learning*, PMLR, 2021, pp. 4040–4050.

[79] J. Neville, D. Jensen, L. Friedland, and M. Hay, "Learning relational probability trees," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 625–630.

[80] S. Luan, M. Zhao, X.-W. Chang, and D. Precup, "Break the ceiling: Stronger multi-scale deep graph convolutional networks," *arXiv preprint arXiv:1906.02174*, 2019.

[81] Z. Chen, S. Villar, L. Chen, and J. Bruna, "On the equivalence between graph isomorphism testing and function approximation with gnns," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 868–15 876.

[82] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," in *Advances in neural information processing systems*, 2019, pp. 2156–2167.

[83] G. Bouritsas, F. Frasca, S. Zafeiriou, and M. M. Bronstein, "Improving graph neural network expressivity via subgraph isomorphism counting," *arXiv preprint arXiv:2006.09252*, 2020.

[84] C. Vignac, A. Loukas, and P. Frossard, "Building powerful and equivariant graph neural networks with structural message-passing," *arXiv e-prints*, arXiv–2006, 2020.

[85] W. Azizian and M. Lelarge, "Characterizing the expressive power of invariant and equivariant graph neural networks," 2021.

[86] D. Beaini, S. Passaro, V. L'etourneau, W. L. Hamilton, G. Corso, and P. Li'o, "Directional graph networks," *arXiv preprint arXiv:2010.02863*, 2020.

[87] H. Maron, E. Fetaya, N. Segol, and Y. Lipman, "On the universality of invariant networks," in *International conference on machine learning*, PMLR, 2019, pp. 4363–4371.

[88] S. Fan and B. Huang, "Recurrent collective classification," *Knowledge and Information Systems*, vol. 60, no. 2, pp. 741–755, 2019.

[89] J. Moore and J. Neville, "Deep collective inference," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[90] M. Qu, Y. Bengio, and J. Tang, "Gmnn: Graph markov neural networks," *arXiv preprint arXiv:1905.06214*, 2019.

[91] P. Vijayan, Y. Chandak, M. M. Khapra, S. Parthasarathy, and B. Ravindran, "Hopf: Higher order propagation framework for deep collective classification," *arXiv preprint arXiv:1805.12421*, 2018.

[92] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, Sep. 1951. DOI: 10.1214/aoms/1177729586. [Online]. Available: https://doi.org/10.1214/aoms/1177729586.

[93] L. Teixeira, B. Jalaian, and B. Ribeiro, "Are graph neural networks miscalibrated?" *arXiv preprint arXiv:1905.02296*, 2019.

[94] 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, pp. 93–93, 2008.

[95] J. Yang, B. Ribeiro, and J. Neville, "Stochastic gradient descent for relational logistic regression via partial network crawls," *arXiv preprint arXiv:1707.07716*, 2017.

[96] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, "Graph random neural networks for semi-supervised learning on graphs," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[97] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, P10008, 2008.

[98] E. Bakshy, D. Eckles, and M. S. Bernstein, "Designing and deploying online field experiments," in *Proceedings of the 23rd international conference on World wide web*, 2014, pp. 283–292.

[99] T. Pham, T. Tran, D. Phung, and S. Venkatesh, "Column networks for collective classification," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[100] J. Jia and A. Benson, *Outcome correlation in graph neural network regression*, 2020. arXiv: 2002.08274 [cs.LG].

[101] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, "Invariant and equivariant graph networks," *arXiv preprint arXiv:1812.09902*, 2018.

[102] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1422–1430.

[103] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *European Conference on Computer Vision*, Springer, 2016, pp. 69–84.

[104] H.-Y. Lee, J.-B. Huang, M. Singh, and M.-H. Yang, "Unsupervised representation learning by sorting sequences," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 667–676.

[105] I. Misra, C. L. Zitnick, and M. Hebert, "Shuffle and learn: Unsupervised learning using temporal order verification," in *European Conference on Computer Vision*, Springer, 2016, pp. 527–544.

[106] O. J. Hénaff, A. Razavi, C. Doersch, S. Eslami, and A. v. d. Oord, "Data-efficient image recognition with contrastive predictive coding," *arXiv preprint arXiv:1905.09272*, 2019.

[107] S. Gidaris, A. Bursuc, N. Komodakis, P. Pérez, and M. Cord, "Boosting few-shot visual learning with self-supervision," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 8059–8068.

[108] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[109] D. Jensen, J. Neville, and B. Gallagher, "Why collective inference improves relational classification," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 593–598.

[110] E. E. Kosasih, J. Cabezas, X. Sumba, P. Bielak, K. Tagowski, K. Idanwekhai, B. A. Tjandra, and A. R. Jamasb, "On graph neural network ensembles for large-scale molecular property prediction," *arXiv preprint arXiv:2106.15529*, 2021.

[111] S. Zhang, M. Liu, and J. Yan, "The diversified ensemble neural network," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[112] Z. Zhang, V. R. Gao, and M. R. Sabuncu, "Ex uno plures: Splitting one model into an ensemble of subnetworks," *arXiv preprint arXiv:2106.04767*, 2021.

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

[114] 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*, Springer, 2018, pp. 593–607.

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

[116] S. Bloehdorn and Y. Sure, "Kernel methods for mining instance data in ontologies," in *The Semantic Web*, Springer, 2007, pp. 58–71.

[117] E. Boschee, J. Lautenschlager, S. O'Brien, S. Shellman, J. Starz, and M. Ward, *ICEWS Coded Event Data*, version V30, 2015. DOI: 10.7910/DVN/28075. [Online]. Available: https://doi.org/10.7910/DVN/28075.

[118] Y. Chang, C. Chen, W. Hu, Z. Zheng, X. Zhou, and L. Sun, "Megnn: Meta-path extracted graph neural network for heterogeneous graph representation learning," *Knowledge-Based Systems*, p. 107 611, 2021.

[119] R. Zreik, P. Latouche, and C. Bouveyron, "The dynamic random subgraph model for the clustering of evolving networks," *Computational Statistics*, vol. 32, no. 2, pp. 501–533, 2017.

[120] D. Zhuang, M. J. Chang, and M. Li, "Dynamo: Dynamic community detection by incrementally maximizing modularity," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[121] N. Dakiche, F. B.-S. Tayeb, Y. Slimani, and K. Benatchba, "Tracking community evolution in social networks: A survey," *Information Processing & Management*, vol. 56, no. 3, pp. 1084–1102, 2019.