Towards More Powerful Graph Representation Learning

Lingxiao Zhao

July 23, 2024

Doctoral Dissertation

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy in Machine Learning and Public Policy

Heinz College & Machine Learning Department
Carnegie Mellon University (CMU)
Pittsburgh, PA

Abstract

Graphs have been widely used in real-world to represent relations, and many data are naturally represented in graphs like social networks, protein structures, molecules, and transactions. Different from images and texts that have specific natural order of their components, graphs are unordered and permutation invariant, which introduce significant difficulties to learning good representations. Graph neural networks (GNNs) are proposed and consistently improved for graph representation learning. This thesis works on both node-level and graph-level representations and proposes solutions towards more powerful representation learning, with the goal of achieving foundation model on graphs.

In Part 1, I focus on node-level graph representation learning and tackle a critical issue known as "representation oversmoothing". To further understand the strengths of Graph Convolutional Networks (GCNs), we explore the relationship between neural networks and Principal Component Analysis (PCA). Our investigation reveals that the solution for Graph-regularized PCA aligns with the formulation of a single graph convolutional layer.

In Part 2, I delve into graph-level representation learning. Unike Multilayer Perceptrons (MLPs), which are universal function approximators for tabular data, GNNs have limited expressivity. I explore two directions to boost GNN's expressivity: using rooted subgraphs to achieve better local structure awareness; and exploring *unordered* higher order interactions to achieve comparable expressivity to their ordered counterparts but also gain great scalability for practical usage.

In Part 3, I focus on generative models for graphs. Unlike discriminative models, which require labels to learn task-dependent representations, generative models excel at unsupervised representation learning. However, graph generation presents unique challenges due to its unordered nature, which demands an approach that is independent of any specific ordering. To address this, I first examine diffusion models applied to categorical data, and propose to simplify and unify existing discrete-time and continuous-time discrete diffusion. Based on the unified discrete diffusion, I propose a partial-order-based diffusion model that combines autoregressive approach with diffusion model for graph generation. This method builds the foundation of generative pretraining on graphs.

In the last Part, I explore application of graph representation learning. Specifically, I work on graph-level anomaly detection (GLAD) which has many influential applications. As GLAD is rarely explored, I build a strong GNN based baseline "OCGIN", an evaluation platform with datasets and many non-GNN baselines. I also discover and study an issue called "performance flip". Later I design a specific model for detecting PwC's suspicious accounting transactions that requires handling attributed multi-graphs with metadata, which can effectively detect expert-guided anomalies.

Acknowledgements

First and foremost, I wish to extend my heartfelt thanks to my PhD advisor, Leman Akoglu, for her indispensable support. Without Leman, embarking on my PhD journey, let alone completing it, would have been inconceivable. As a master student transitioning from power systems to machine learning, it was Leman who advised and supported me when I almost gave up, instilling in me the confidence to pursue a PhD. Moreover, the freedom, encouragement, and unwavering support she provided throughout my doctoral studies have been critical in my development as an independent researcher. I am incredibly fortunate to have had the opportunity to delve into research problems that genuinely interest me. Also, I would like to thank my another co-advisor Aarti Singh, for her guidance, feedback, and support at Machine Learning Department. I also want to thank my committee members, Neil Shah and Andrej Risteski, for their valuable input and suggestions.

Secondly, I want to thank many of my mentors, Neil Shah, Haggai Maron, Pierre Liang, David Choi, Siheng Chen, and Christos Faloutsos, for their selfless support, assistance, and guidance, and without whom this thesis cannot be possible. I am so lucky to meet them at the path of my growth. I am especially grateful to Neil, who has advised me tirelessly through any difficulties along the way.

Besides, I thank Michelle Wirtz, Diane Stidle, Lori Geraci and Adrienne Mccorkle, for their prompt assistance and support throughout the duration of the program.

To my collaborators, Xuan Wu, Saurabh Sawlani, Tuan Le, Konstantinos Sotiropoulos, Jaemin Yoo, Xueying Ding, Jeremy Lee, Derek Lim, Joshua Robinson, Wei Jin, Jiliang Tang, Jiong Zhu, Yujun Yan, Danai Koutra, Yue Zhao, Shubhranshu Shekhar, Louis Härtel, for their input, collaboration, and support. Without them many of works cannot be finished.

To my friends and cohort, for their companionship and shared experiences in courses, activities, and the lovely but suffering journey.

Last but not the least, to my parents, and my families for their unconditional love, endless encouragement, and selfless dedication. Their sacrifices are the root of my achievements. Also, to my cats Mushu and Mulan, for their company and emotional support.

In the end, the PhD journey is long and filled with many ups and downs, achievements and failures, confusions and explorations, in research, life, and family. Looking back, I'm not sure if I could pass through this again, but I'm glad I had these unrestrained years for unrestricted study, pursuing intrinsic research that follows my heart and interests, loving without hesitation, and failing without fear. To myself, and wish I can continue pursue my passion for research and can love boldly with all my heart.

Contents

List of Figures

List of Tables

Chapter 0

Introduction

My fascination with graphs began during my undergraduate studies, where I frequently employed graph algorithms in mathematical modeling contests and studied circuits and power systems (my bachelor major) with graphs as their abstractions for analysis. At the beginning of my PhD in 2018, the field of graph neural networks (GNNs) was emerging, with pioneering work [KW17] in 2016 showcasing their promising potential in node classification and regression tasks. This, along with my passion for graphs, naturally led me to choose the area of GNNs as the start of my PhD journey, which involves designing new neural networks for processing graph structures. Unconsciously and inevitably, I have dedicated all my enthusiasm to this field, making graphs and GNNs central to my research journey. Fortunately, my passion, knowledge, understanding, and achievements have grown alongside the field itself, as I've witnessed GNNs evolve from a nascent field to a mature area of study.

This thesis addresses several fundamental challenges and advancements in GNNs as the field progresses towards maturity, encompassing theory, diverse applications and domains. Graphs are abstract elements used to model relationships or structures across various domains and problems. For example, social networks are represented as graphs to capture relationships or activities among individuals, where the main concerns are node-level problems such as individual behavior. In recommendation systems, graphs abstract users and products as nodes and user-product engagements as edges, with the main focus on predicting engagements (edge-level problems). Additionally, graphs can represent molecules, with atoms as nodes and chemical bonds as edges, where the primary concern is understanding molecule properties (graph-level tasks). Due to the diversity of tasks, domains, complications, and levels of abstraction, fully developing the field is akin to the blind men describing an elephant; it requires many stages of growth and understanding.

The field begins with *node-level* problems, which are also the focus of the **Part** I of the thesis. At early stage, graph network approaches like GCN [KW17] have started to outperform traditional propagation algorithms like lable propagation [ZGL03] and neural graph embedding approaches like DeepWalk [PARS14]. In the meantime, as the esteemed ResNet [He+16] clears the difficulty of training deep networks and demonstrates that making networks deeper enlarges model capacity and outperforms widen them, researchers explore deepening GNNs to further improve the performance in handling complex graph structures. However, counter-intuitive phenomenons are widely observed: increasing the number of layers in GNNs often leads to unimproved or even diminished performance. To address and explain the observation, [LHW18] proposes a hypothesis called "oversmoothing". This hypothesis suggests that the repeated message passing or graph convolutional layers can cause nodes' representations to become indistinguishable, thereby reducing performance when too many layers are stacked. **Chapter** 1 delves deeply into the oversmoothing hypothesis, presenting assumptions, metrics, and controlled experiments designed to quantify oversmoothing and its impact on performance in semi-supervised node classification tasks. Our comprehensive studies and evaluations demonstrate that both forms of oversmoothing, column-wise oversmoothing where all features become identical and row-wise oversmoothing where all nodes converge to a uniform representation, happen with the addition of more layers. As a follow up, the first normalization layer in GNNs, PairNorm, that normalizes the total pairwise distances among representations is presented to eliminate the shortcoming of deepening graph networks. Another intriguing problem is to figuring out the root of the effectiveness of graph convolutions on node-level problems, after which we can further improve model designs to achieve performance boost. Interestingly, we have found that while oversmoothing is harmful to node embeddings, proper amount of smoothing contributes to the success of graph convolution for node-level tassk. In **Chapter** 2, we studies the connection between graph convolution and graph-regularized PCA. Specifically, we establish a mathematical connection between the form of graph convolution and the close-form solution of graph-regularized PCA. The connection highlights that the effective smoothing or graph convolution power derives from the graph-based Laplacian regularization term. Moreover, increasing the depth of GCNs is analogous to enhancing the coefficient of the Laplacian regularization term. Building on these connections, one can analyze the limitations of GCNs through the lens of Laplacian regularization. For instance, my collaborators have delved into the homophily and heterophily issues in GNNs for node-level tasks [Zhu+20]. The heterophily issue of GNNs can be explained by the limitation of the Laplacian regularization, as its effectiveness is determined by how well the graph structure aligns with task labels. While node-level problems are the most widely studied problems in GNNs, there are still some theoretical questions not fully addressed. For example, the learning of node-level tasks is essentially a non-i.i.d. based learning, as all nodes are influenced by each other in the graph, where many theories from i.i.d. setting cannot be directly applied.

As the field advances, *graph-level* problems such as graph property prediction tasks have gained popularity. In these scenarios, a collection of graphs is presented, either with or without associated properties, and all graphs are assumed to be independently and identically distributed (i.i.d.). The **Part** II focuses on graph-level tasks. Given that each input is a graph, our goal is to develop powerful GNNs capable of modeling any function on graphs. Multi-Layer Perceptrons (MLPs), which are simple yet well-known, serve as universal function approximators for functions on high-dimensional vectors given enough depth and width. Back in early 2019, Chen et al. [Che+19a] proved that being a universal function approximator for functions on graphs is equivalent to being capable of solving the graph isomorphism test problem. Consequently, the expressiveness of GNNs has since been measured by their ability to solve the graph isomorphism test problem. However, as the graph isomorphism test is not known to be solvable in polynomial time [For96], GNNs cannot be universal function approximators for graph functions either, which dramatically distinguishes from MLPs on vector functions. In fact, [Xu+19] proves that the popular message-passing-based GNNs have limited expressiveness upper bounded by the first-order Weisfeiler-Leman algorithm (1-WL). This means that GNNs cannot distinguish graphs that the 1-WL algorithm fails to differentiate. As a consequence, GNNs cannot count the number of triangles and cycles that is important for many real-world problems. Improving the expressiveness of GNNs towards universal function approximators for function on graphs become the vital bottleneck for the field to advance. In **Chapter** 3, I explore the explicit usage of subgraphs to boost model expressiveness. Observing that typical GNNs, which aggregate only local first-hop information, are limited in discriminating non-isomorphic graphs, for the first time, I propose to generalize the star-like aggregation to subgraph aggregation. The insight results in GNNAsKernel [Zha+22c], a general framework with *theoretical guarantee* in expressiveness that enhances any GNN by convolving subgraphs with base GNN as kernel. This framework, incorporating both intra- and inter-subgraph interactions, has significantly improved performance across various graph task benchmarks with up to 60% error reduction. The direction of using subgraphs to enhance GNN expressiveness has since been a significant research direction, followed by many in the field. For example, Zhang et al. [Zha+23a] has proved that all subgraph GNNs with message passing base GNN have expressiveness upper bounded by 3-WL. Subgraph GNNs substantially enhance the expressiveness of existing GNN models, yet they are still constrained by the 3-WL test's upper bound. Moreover, the pursuit of increased expressiveness may not always be justifiable, as it can lead to considerable computational cost and possibly weaker generalization. There's a growing need in the community for GNNs that combine high expressiveness without limit and adjustable expressiveness. This approach would allow for a systematic evaluation of expressiveness's impact on real-world problems. In **Chapter** 4, I work on developing GNNs that are both practically efficient and progressively expressive, backed by theoretical guarantees. Recognizing the high expressiveness of higher-order interactions, my focus is on overcoming their primary limitation: the exponential computational cost associated with considering all k-order tuples. In [Zha+22b], I investigated the approach of minimizing the number of higher-order entities by focusing on sets instead of tuples. The shift from tuples to sets means disregarding the information embedded in the order and repetition of elements. Although this strategy significantly decreases the number of entities involved, assessing its theoretical impact on expressiveness is challenging. By applying theory from first-order logic and game theory, I demonstrate that, while there is a reduction in expressivity, a k-order set-based GNN still maintains expressivity that is not less powerful than a (k-1)-order tuple-based GNN. A new higher-order model, (k, c)-SetGNN, is developed based on the set version of higher-order WL. This work has achieved new state-of-the-art performance in many benchmarks with previously not achievable order k=10, significantly expands the possibility of higher order GNNs.

In addition to designing more expressive model architectures, developing unsupervised learning paradigms adapted to graphs is increasingly important to unlock the rich information inside the abundant amount of unlabeled data. Generative modeling, one of the most promising unsupervised learning approaches, has already revolutionized many areas such as language modeling and computer vision [Ope23], with groundbreaking models like GPT-4 and Claude 3 being widely used by hundreds of millions and adopted in various applications. Generative model essentially captures the full information of the training dataset, as with a perfectly trained generative model you can recover the training dataset with sampling from the model. Nevertheless, developing effective generative models for graph data remains challenging, given the unique characteristics of graphs such as permutation invariance and rich higher-order structural information. In **Part** III, we explore designing permutation-invariant generative model for graphs. The major difference between graphs and sequences, such as languages and image patches, is that graphs lack a natural ordering for their elements (nodes and edges). Any permutation of nodes and edges does not change the graph, making autoregressive approaches used for sequences less effective for graphs without compromising their generalization to arbitrary orderings. Recently, diffusion models, such as the one proposed by [HJA20], have gained increasing attention in the graph domain. These models are capable of modeling exchangeable probabilities, ensuring that the learned probability remains invariant to changes in the ordering of elements, under certain conditions. Nevertheless, graphs are essentially discrete objects where many sophisticatedly developed continuous-state diffusion models cannot be applied. Hence in **Chapter** 5, I first make several improvements to the current discrete-sate diffusion models to make it more effective and easy-to-use for graphs. Specifically, I developed a series of mathematical simplifications for both discrete-time and continuous-time discrete diffusion models, achieving exact and faster sampling for both. I also created a straightforward yet precise formulation for the variational lower bound, enhancing the training speed significantly. Our approach allows both forward and backward probabilities to accommodate any noise distribution, even for complex multi-element objects. Most importantly, *the designed diffusion model provides a single set of forward and backward procedures that unifies discrete- and continuous-time discrete diffusion*. With the effective discrete diffusion model developed, in chapter 6 I explore the possibility of combining autoregressive approach and diffusion model together to harness the effectiveness and efficiency of both while eliminating their shortcomings. Specifically, while the autoregressive approach struggles with generalizing to arbitrary orderings, the diffusion model often requires thousands of denoising steps. This significantly slows down both training and generation due to the difficulty of directly modeling the joint distribution of all edges and nodes. Recognizing that nodes in a graph, unlike elements in a set, have a partial order based on their connections, I explore breaking down the joint distribution into a series of simpler conditional distributions. This decomposition follows a blockwise autoregressive pattern, where each conditional distribution represents a simpler subset of nodes and edges. Crucially, to maintain permutation invariance in probability modeling, diffusion models are employed for each conditional distribution. These models provide the necessary symmetry breaking, vital for accurately predicting new connections in a block of structurally equivalent nodes. Furthermore, I also develop parallel training based on causal principles, akin to a causal transformer. This approach trains all blocks simultaneously, greatly enhances efficiency and usage.

In addition to fundamental model and methodology improvements, I have also worked on applying these models in practical scenarios, shown in the last **Part** IV. Specifically, I have focused on applying graph neural networks for structural-based anomaly detection. In **Chapter** 7, I developed the first GNN based method for graph-level anomaly detection (GLAD), established a comprehensive testbed with extensive baselines, and analyzed key issues in GLAD [ZA23]. Recognizing the challenges in hyperparameter setting for the unsupervised nature of GLAD, I further enhanced the method by incorporating automatic model selection and a more effective loss function through MMD pooling [Zha+22d]. Furthermore, in **Chapter** 8, we expand the GLAD model to process directed, multi-edge graphs, which are commonly found in financial systems. As a real-world study, we have applied our method to PwC's accounting dataset, with the goal of identifying fraud bookkeeping journals containing inaccurate transaction records.

To summarize, this thesis encloses my research explorations on various questions that I considered important for the field as it progresses toward maturity, varying from fundamental problems such as oversmoothing, expressiveness to models, learning methodologies, and applications. I am fortunate to have worked on questions that interest me the most without any restriction , and have the opportunity to grow alongside the field.

Part I

Node-level Representation Learning

Chapter 1

Oversmoothing in GNNs

Chapter based on: Lingxiao Zhao and Leman Akoglu. "PairNorm Tackling Oversmoothing in {GNN}s". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=rkecl1rtwB.

1.1 Introduction

Graph neural networks (GNNs) is a family of neural networks that can learn from graph structured data. Starting with the success of GCN [KW17] on achieving state-of-the-art performance on semi-supervised classification, several variants of GNNs have been developed for this task; including GraphSAGE [HYL17], GAT [Vel+18], SGC [Wu+19], and GMNN [QBT19] to name a few most recent ones.

A key issue with GNNs is their depth limitations. It has been observed that deeply stacking the layers often results in significant drops in performance for GNNs, such as GCN and GAT, even beyond just a few (2–4) layers. This drop is associated with a number of factors; including the vanishing gradients in back-propagation, overfitting due to the increasing number of parameters, as well as the phenomenon called oversmoothing. [LHW18] was the first to call attention to the oversmoothing problem. Having shown that the graph convolution is a type of *Laplacian smoothing*, they proved that after repeatedly applying Laplacian smoothing many times, the features of the nodes in the (connected) graph would converge to similar values—the issue coined as "*oversmoothing*". In effect, oversmoothing hurts classification performance by causing the node representations to be indistinguishable across different classes. Later, several others have alluded to the same problem [Xu+18; KBG19; Ron+19; Li+19a].

In this work, we address the oversmoothing problem in deep GNNs. Specifically, we propose (to the best of our knowledge) *the first normalization layer for GNNs* that is applied in-between intermediate layers during training. Our normalization has the effect of preventing the output features of distant nodes to be too similar or indistinguishable, while at the same time allowing those of connected nodes in the same cluster become more similar. We summarize our main contributions as follows.

In this work, we address the oversmoothing problem in deep GNNs. Specifically, we propose (to the best of our knowledge) *the first normalization layer for GNNs* that is applied in-between intermediate layers during training. Our normalization has the effect of preventing the output features of distant nodes to be too similar or indistinguishable, while at the same time allowing those of connected nodes in the same cluster become more similar. We summarize our main contributions as follows.

- **Normalization to Tackle Oversmoothing in GNNs:** We introduce a normalization scheme, called PairNorm, that makes GNNs significantly more robust to oversmoothing and as a result enables the training of deeper models without sacrificing performance. Our proposed scheme capitalizes on the understanding that most GNNs perform a special form of Laplacian smoothing, which makes node features more similar to one another. The key idea is to ensure that the total pairwise feature distances remains a *constant* across layers, which in turn leads to distant pairs having less similar features, preventing feature mixing across clusters.
- **Speed and Generality:** PairNorm is very straightforward to implement and introduces no additional parameters. It is simply applied to the output features of each layer (except the last one) consisting of simple operations, in particular centering and scaling, that are *linear* in the input size. Being a simple normalization step between layers, PairNorm is not specific to any particular GNN but rather applies broadly. In this work we use PairNorm to tackle oversmoothing for the GCN, GAT, and SGC models.
- **Use Case for Deeper GNNs:** While PairNorm prevents performance from dropping significantly with increasing number of layers, it does not necessarily yield increased performance in absolute terms. We find that this is because shallow architectures with no more than 2–4 layers is sufficient for the often-used benchmark datasets in the literature. In response, we motivate a real-world scenario wherein a notable portion of the nodes have *no* feature vectors. In such settings, nodes benefit from a larger range (i.e., neighborhood, hence a deeper GNN) to “recover” effective feature representations. Through extensive experiments, we show that GNNs employing our PairNorm significantly outperform the ‘vanilla’ GNNs when deeper models are beneficial to the classification task.

1.2 Related Work

Oversmoothing in GNNs: [LHW18] was the first to call attention to the oversmoothing problem. They proposed propagation-based co-training to introduce more labels into the training of the GNN, which helps increase its range (i.e., size of the neighborhood used). [Xu+18] introduced Jumping Knowledge Networks, which employ skip connections for multi-hop message passing and also enable different neighborhood ranges. [KBG19] proposed a propagation scheme based on personalized Pagerank that ensures locality (via teleports) which in turn prevents oversmoothing. [Li+19a] built on ideas from ResNet to use residual as well as dense connections to train deep GCNs. DropEdge [Ron+19] proposed to alleviate oversmoothing through message passing reduction via removing a certain fraction of edges at random from the input graph. Finally, [Fey19] proposed Just-Jump, a scheme that prevents new representations from being “washed out” by selectively aggregating those of neighbors’ from all previous layers. These are all specialized solutions that introduce additional parameters and/or a different network architecture.

Normalization Schemes for Deep-NNs: There exist various normalization schemes proposed for deep neural networks, including batch normalization [IS15], weight normalization [SK16], layer normalization [BKH16], and so on. Conceptually these have substantially different goals (e.g., reducing training time), and were not proposed for *graph* neural networks nor the oversmoothing problem therein. Important difference to note is that larger depth in regular neural-nets does not translate to more hops of propagation on a graph structure.

1.3 Understanding Oversmoothing

In this work, we consider the semi-supervised node classification (SSNC) problem on a graph. In the general setting, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ is given in which each node $i \in \mathcal{V}$ is associated with a feature vector $\mathbf{x}_i \in \mathbb{R}^d$ where $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^T$ denotes the feature matrix, and a subset $\mathcal{V}_l \subset \mathcal{V}$ of the nodes are labeled, i.e. $y_i \in \{1, \ldots, c\}$ for each $i \in \mathcal{V}_l$ where c is the number of classes. Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be the adjacency matrix and $\mathbf{D} = \text{diag}(deg_1, \ldots, deg_n) \in \mathbb{R}^{n\times n}$ be the degree matrix of $\mathcal{G}$. Let $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}$ denote the augmented adjacency and degree matrices with added self-loops on all nodes, respectively. Let $\tilde{\mathbf{A}}_{\text{sym}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ and $\tilde{\mathbf{A}}_{\text{rw}} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$ denote symmetrically and nonsymmetrically normalized adjacency matrices with self-loops.

The task is to learn a hypothesis that predicts y_i from $\mathbf{x}_i$ that generalizes to the unlabeled nodes $\mathcal{V}_u = \mathcal{V}\backslash\mathcal{V}_l$. In Section 1.4.2, we introduce a variant of this setting where only a subset $\mathcal{F} \subset \mathcal{V}$ of the nodes have feature vectors and the rest are missing.

1.3.1 The Oversmoothing Problem

Although GNNs like GCN and GAT achieve state-of-the-art results in a variety of graph-based tasks, these models are not very well-understood, especially why they work for the SSNC problem where only a small amount of training data is available. The success appears to be limited to shallow GNNs, where the performance gradually decreases with the increasing number of layers. This decrease is often attributed to three contributing factors: (1) overfitting due to increasing number of parameters, (2) difficulty of training due to vanishing gradients, and (3) oversmoothing due to many graph convolutions.

Among these, perhaps the least understood one is oversmoothing, which indeed lacks a formal definition. In their analysis of GCN's working mechanism, [LHW18] showed that the graph convolution of GCN is a special form of Laplacian smoothing. The standard form being $(\mathbf{I} - \gamma\mathbf{I})\mathbf{X} + \gamma\tilde{\mathbf{A}}_{\text{rw}}\mathbf{X}$, the graph convolution lets $\gamma = 1$ and uses the symmetrically normalized Laplacian to obtain $\tilde{\mathbf{X}} = \tilde{\mathbf{A}}_{\text{sym}}\mathbf{X}$, where the new features $\tilde{\mathbf{x}}$ of a node is the weighted average of its own and its neighbors' features. This smoothing allows the node representations within the same cluster become more similar, and in turn helps improve SSNC performance under the cluster assumption [CSZ06]. However when GCN goes deep, the performance can suffer from oversmoothing where node representations from different clusters become mixed up. Let us refer to this issue of node representations becoming too similar as *node-wise* oversmoothing.

Another way of thinking about oversmoothing is as follows. Repeatedly applying Laplacian smoothing too many times would drive node features to a stationary point, washing away all the information from these features. Let $\mathbf{x}_{\cdot j} \in \mathbb{R}^n$ denote the j-th column of $\mathbf{X}$. Then, for *any* $\mathbf{x}_{\cdot j} \in \mathbb{R}^n$:

$$\lim_{k\to\infty} \tilde{\mathbf{A}}^k_{\text{sym}}\mathbf{x}_{\cdot j} = \boldsymbol{\pi}_j \quad \text{and} \quad \frac{\boldsymbol{\pi}_j}{\|\boldsymbol{\pi}_j\|_1} = \boldsymbol{\pi} \ , \tag{1.1}$$

where the normalized solution $\boldsymbol{\pi} \in \mathbb{R}^n$ satisfies $\boldsymbol{\pi}_i = \frac{\sqrt{deg_i}}{\sum_i \sqrt{deg_i}}$ for all $i \in [n]$. Notice that $\boldsymbol{\pi}$ is independent of the values $\mathbf{x}_{\cdot j}$ of the input feature and is only a function of the graph structure (i.e., degree). In other words, (Laplacian) oversmoothing washes away the signal from *all* the features, making them indistinguishable. We will refer to this viewpoint as *feature-wise* oversmoothing.

To this end we propose two measures, row-diff and col-diff, to quantify these two types of oversmoothing. Let $\mathbf{H}^{(k)} \in \mathbb{R}^{n\times d}$ be the representation matrix after k graph convolutions, i.e. $\mathbf{H}^{(k)} = \tilde{\mathbf{A}}^k_{\text{sym}}\mathbf{X}$. Let $\mathbf{h}^{(k)}_i \in \mathbb{R}^d$ be the i-th row of $\mathbf{H}^{(k)}$ and $\mathbf{h}^{(k)}_{\cdot i} \in \mathbb{R}^n$

be the i-th column of $\mathbf{H}^{(k)}$. Then we define row-diff($\mathbf{H}^{(k)}$) and col-diff($\mathbf{H}^{(k)}$) as follows.

$$\text{row-diff}(\mathbf{H}^{(k)}) = \frac{1}{n^2} \sum_{i,j\in[n]} \left\| \mathbf{h}_i^{(k)} - \mathbf{h}_j^{(k)} \right\|_2 \tag{1.2}$$

$$\text{col-diff}(\mathbf{H}^{(k)}) = \frac{1}{d^2} \sum_{i,j\in[d]} \left\| \mathbf{h}_{\cdot i}^{(k)} / \|\mathbf{h}_{\cdot i}^{(k)}\|_1 - \mathbf{h}_{\cdot j}^{(k)} / \|\mathbf{h}_{\cdot j}^{(k)}\|_1 \right\|_2 \tag{1.3}$$

The row-diff measure is the average of all pairwise distances between the node features (i.e., rows of the representation matrix) and quantifies node-wise oversmoothing, whereas col-diff is the average of pairwise distances between (L_1-normalized[1]) columns of the representation matrix and quantifies feature-wise oversmoothing.

1.3.2 Studying Oversmoothing with SGC

Although oversmoothing can be a cause of performance drop with increasing number of layers in GCN, adding more layers also leads to more parameters (due to learned linear projections $\mathbf{W}^{(k)}$ at each layer k) which magnify the potential of overfitting. Furthermore, deeper models also make the training harder as backpropagation suffers from vanishing gradients.

In order to decouple the effect of oversmoothing from these other two factors, we study the oversmoothing problem using the SGC model [Wu+19]. (Results on other GNNs are presented in §1.5.) SGC is simplified from GCN by removing all projection parameters of graph convolution layers and all nonlinear activations between layers. The estimation of SGC is simply written as:

$$\widehat{\boldsymbol{Y}} = \text{softmax}(\tilde{\mathbf{A}}_{\text{sym}}^K \; \mathbf{X} \; \mathbf{W}) \tag{1.4}$$

where K is the number of graph convolutions, and $\mathbf{W} \in \mathbb{R}^{d\times c}$ denote the learnable parameters of a logistic regression classifier.

Note that SGC has a *fixed* number of parameters that does not depend on the number of graph convolutions (i.e. layers). In effect, it is guarded against the influence of overfitting and vanishing gradient problem with more layers. This leaves us only with oversmoothing as a possible cause of performance degradation with increasing K. Interestingly, the simplicity of SGC does not seem to be a sacrifice; it has been observed that it achieves similar or better accuracy in various relational classification tasks [Wu+19].

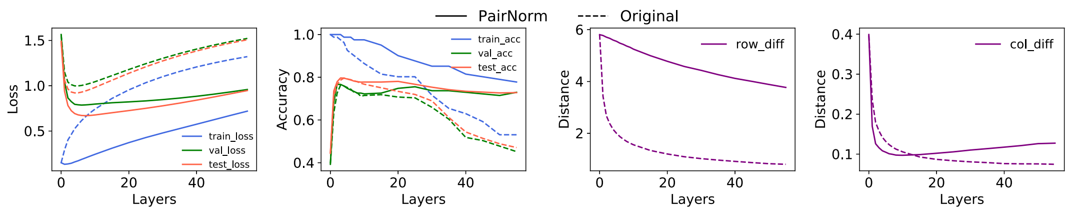


FIGURE 1.1: SGC's performance (dashed lines) with increasing graph convolutions (K) on `Cora` dataset (train/val/test split is 3%/10%/87%). For each K, we train SGC in 500 epochs, save the model with the best validation accuracy, and report all measures based on the saved model. Measures row-diff and col-diff are computed based on the final layer representation of the saved model.

[1]We normalize each column j as the Laplacian smoothing stationary point $\boldsymbol{\pi}_j$ is not scale-free. See Eq. (1.1).

Dashed lines in Figure 1.1 illustrate the performance of SGC on the `Cora` dataset as we increase the number of layers (K). The training (cross-entropy) loss monotonically increases with larger K, potentially because graph convolution mixes node representations with their neighbors' and makes them less distinguishable (training becomes harder). On the other hand, graph convolutions (i.e., smoothing) improve generalization ability, reducing the gap between training and validation/test loss up to $K = 4$, after which (over)smoothing begins to hurt performance. The row-diff and col-diff both continue decreasing monotonically with K, providing supporting evidence for oversmoothing.

1.4 Tackling Oversmoothing

1.4.1 Proposed PairNorm

We start by establishing a connection between graph convolution and an optimization problem, that is graph-regularized least squares (GRLS), as shown by [NM19]. Let $\bar{\mathbf{X}} \in \mathbb{R}^{n\times d}$ be a new node representation matrix, with $\bar{\mathbf{x}}_i \in \mathbb{R}^d$ depicting the i-th row of $\bar{\mathbf{X}}$. Then the GRLS problem is given as

$$\min_{\bar{\mathbf{X}}} \sum_{i\in\mathcal{V}} \|\bar{\mathbf{x}}_i - \mathbf{x}_i\|^2_{\tilde{\mathbf{D}}} + \sum_{(i,j)\in\mathcal{E}} \|\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j\|_2^2 \tag{1.5}$$

where $\|\mathbf{z}_i\|^2_{\tilde{\mathbf{D}}} = \mathbf{z}_i^T\tilde{\mathbf{D}}\mathbf{z}_i$. The first term can be seen as total degree-weighted least squares. The second is a graph-regularization term that measures the *variation* of the new features over the graph structure. The goal of the optimization problem can be stated as estimating new "denoised" features $\bar{\mathbf{x}}_i$'s that are not too far off of the input features $\mathbf{x}_i$'s and are *smooth* over the graph structure.

The GRLS problem has a closed form solution $\bar{\mathbf{X}} = (2\mathbf{I} - \tilde{\mathbf{A}}_{\text{rw}})^{-1}\mathbf{X}$, for which $\tilde{\mathbf{A}}_{\text{rw}}\mathbf{X}$ is the first-order Taylor approximation, that is $\tilde{\mathbf{A}}_{\text{rw}}\mathbf{X} \approx \bar{\mathbf{X}}$. By exchanging $\tilde{\mathbf{A}}_{\text{rw}}$ with $\tilde{\mathbf{A}}_{\text{sym}}$ we obtain the same form as the graph convolution, i.e., $\tilde{\mathbf{X}} = \tilde{\mathbf{A}}_{\text{sym}}\mathbf{X} \approx \bar{\mathbf{X}}$. As such, graph convolution can be viewed as an approximate solution of (1.5), where it minimizes the variation over the graph structure while keeping the new representations close to the original.

The optimization problem in (1.5) facilitates a closer look to the oversmoothing problem of graph convolution. Ideally, we want to obtain smoothing over nodes within the same cluster, however avoid smoothing over nodes from different clusters. The objective in (1.5) dictates only the first goal via the graph-regularization term. It is thus prone to oversmoothing when convolutions are applied repeatedly. To circumvent the issue and fulfill both goals simultaneously, we can add a negative term such as the sum of distances between disconnected pairs as follows.

$$\min_{\bar{\mathbf{X}}} \sum_{i\in\mathcal{V}} \|\bar{\mathbf{x}}_i - \mathbf{x}_i\|^2_{\tilde{\mathbf{D}}} + \sum_{(i,j)\in\mathcal{E}} \|\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j\|_2^2 - \lambda \sum_{(i,j)\notin\mathcal{E}} \|\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j\|_2^2 \tag{1.6}$$

where λ is a balancing scalar to account for different volume and importance of the two goals.[2] By deriving the closed-form solution of (1.6) and approximating it with first-order Taylor expansion, one can get a revised graph convolution operator with hyperparameter λ. In this paper, we take a different route. Instead of a completely new graph convolution operator, we propose a general and efficient "patch", called PairNorm, that can be applied to any form of graph convolution having the potential of oversmoothing.

[2]There exist other variants of (1.6) that achieve similar goals, and we leave the space for future exploration.

Let $\tilde{\mathbf{X}}$ (the output of graph convolution) and $\dot{\mathbf{X}}$ respectively be the input and output of PairNorm. Observing that the output of graph convolution $\tilde{\mathbf{X}} = \tilde{\mathbf{A}}_{\text{sym}}\mathbf{X}$ only achieves the first goal, PairNorm serves as a normalization layer that works on $\tilde{\mathbf{X}}$ to achieve the second goal of keeping disconnected pair representations farther off. Specifically, PairNorm normalizes $\tilde{\mathbf{X}}$ such that the total pairwise squared distance $\text{TPSD}(\dot{\mathbf{X}}) := \sum_{i,j\in[n]} \|\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j\|_2^2$ is the same as $\text{TPSD}(\mathbf{X})$. That is,

$$\sum_{(i,j)\in\mathcal{E}} \|\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j\|_2^2 \;+ \sum_{(i,j)\notin\mathcal{E}} \|\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j\|_2^2 \;=\; \sum_{(i,j)\in\mathcal{E}} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \;+ \sum_{(i,j)\notin\mathcal{E}} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \;. \tag{1.7}$$

By keeping the total pairwise squared distance *unchanged*, the term $\sum_{(i,j)\notin\mathcal{E}} \|\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j\|_2^2$ is guaranteed to be at least as large as the original value $\sum_{(i,j)\notin\mathcal{E}} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$ since the other term $\sum_{(i,j)\in\mathcal{E}} \|\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j\|_2^2 \approx \sum_{(i,j)\in\mathcal{E}} \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2^2$ is shrunk through the graph convolution.

In practice, instead of always tracking the original value $\text{TPSD}(\mathbf{X})$, we can maintain a *constant* TPSD value C across all layers, where C is a hyperparameter that could be tuned per dataset.

To normalize $\tilde{\mathbf{X}}$ to constant TPSD, we need to first compute $\text{TPSD}(\tilde{\mathbf{X}})$. Directly computing TPSD involves n^2 pairwise distances that is $\mathcal{O}(n^2 d)$, which can be time consuming for large datasets. Equivalently, normalization can be done via a two-step approach where TPSD is rewritten as

$$\text{TPSD}(\tilde{\mathbf{X}}) \;=\; \sum_{i,j\in[n]} \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2^2 \;=\; 2n^2\left(\frac{1}{n}\sum_{i=1}^{n}\|\tilde{\mathbf{x}}_i\|_2^2 - \|\frac{1}{n}\sum_{i=1}^{n}\tilde{\mathbf{x}}_i\|_2^2\right) \;. \tag{1.8}$$

Proof.

$$\begin{aligned}
\text{TPSD}(\tilde{\mathbf{X}}) &= \sum_{i,j\in[n]} \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2^2 = \sum_{i,j\in[n]} (\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j)^T(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j) \\
&= \sum_{i,j\in[n]} (\tilde{\mathbf{x}}_i^T\tilde{\mathbf{x}}_i + \tilde{\mathbf{x}}_j^T\tilde{\mathbf{x}}_j - 2\tilde{\mathbf{x}}_i^T\tilde{\mathbf{x}}_j) \\
&= 2n\sum_{i\in[n]} \tilde{\mathbf{x}}_i^T\tilde{\mathbf{x}}_i - 2\sum_{i,j\in[n]} \tilde{\mathbf{x}}_i^T\tilde{\mathbf{x}}_j \\
&= 2n\sum_{i\in[n]} \|\tilde{\mathbf{x}}_i\|_2^2 - 2\mathbf{1}^T\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T\mathbf{1} \\
&= 2n\sum_{i\in[n]} \|\tilde{\mathbf{x}}_i\|_2^2 - 2\|\mathbf{1}^T\tilde{\mathbf{X}}\|_2^2 \\
&= 2n^2\left(\frac{1}{n}\sum_{i=1}^{n}\|\tilde{\mathbf{x}}_i\|_2^2 - \|\frac{1}{n}\sum_{i=1}^{n}\tilde{\mathbf{x}}_i\|_2^2\right) \;.
\end{aligned} \tag{1.9}$$

□

The first term (ignoring the scale $2n^2$) in Eq. (1.8) represents the mean squared length of node representations, and the second term depicts the squared length of the mean of node representations. To simplify the computation of (1.8), we subtract the row-wise mean from each $\tilde{\mathbf{x}}_i$, i.e., $\tilde{\mathbf{x}}_i^c = \tilde{\mathbf{x}}_i - \frac{1}{n}\sum_i^n \tilde{\mathbf{x}}_i$ where $\tilde{\mathbf{x}}_i^c$ denotes the centered representation. Note that this shifting does *not* affect the TPSD, and furthermore drives the term $\|\frac{1}{n}\sum_{i=1}^n \tilde{\mathbf{x}}_i\|_2^2$ to zero, where computing $\text{TPSD}(\tilde{\mathbf{X}})$ boils down to calculating the squared Frobenius norm of $\tilde{\mathbf{X}}^c$ and overall takes $\mathcal{O}(nd)$. That is,

$$\text{TPSD}(\tilde{\mathbf{X}}) = \text{TPSD}(\tilde{\mathbf{X}}^c) = 2n\|\tilde{\mathbf{X}}^c\|_F^2 \;. \tag{1.10}$$

In summary, our proposed PairNorm (with input $\tilde{\mathbf{X}}$ and output $\dot{\mathbf{X}}$) can be written as a two-step, center-and-scale, normalization procedure:

$$\tilde{\mathbf{x}}_i^c = \tilde{\mathbf{x}}_i - \frac{1}{n}\sum_{i=1}^{n} \tilde{\mathbf{x}}_i \qquad \text{(Center)} \qquad (1.11)$$

$$\dot{\mathbf{x}}_i = s \cdot \frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\|\tilde{\mathbf{x}}_i^c\|_2^2}} = s\sqrt{n} \cdot \frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\|\tilde{\mathbf{X}}^c\|_F^2}} \qquad \text{(Scale)} \qquad (1.12)$$

After scaling the data remains centered, that is, $\|\sum_{i=1}^{n}\dot{\mathbf{x}}_i\|_2^2 = 0$. In Eq. (1.12), s is a hyperparameter that determines C. Specifically,

$$\text{TPSD}(\dot{\mathbf{X}}) = 2n\|\dot{\mathbf{X}}\|_F^2 = 2n\sum_i \|s \cdot \frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\frac{1}{n}\sum_i \|\tilde{\mathbf{x}}_i^c\|_2^2}}\|_2^2 = 2n\frac{s^2}{\frac{1}{n}\sum_i \|\tilde{\mathbf{x}}_i^c\|_2^2}\sum_i \|\tilde{\mathbf{x}}_i^c\|_2^2 = 2n^2s^2 \qquad (1.13)$$

Then, $\dot{\mathbf{X}} := \text{PairNorm}(\tilde{\mathbf{X}})$ has row-wise mean $\mathbf{0}$ (i.e., is centered) and constant total pairwise squared distance $C = 2n^2s^2$. An illustration of PairNorm is given in Figure 1.2. The output of PairNorm is input to the next convolution layer.

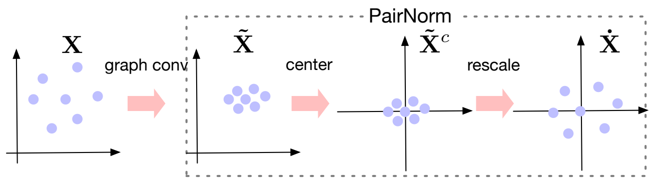


Figure 1.2: Illustration of PairNorm, comprising centering and rescaling steps.

We also derive a variant of PairNorm by replacing $\sum_{i=1}^{n}\|\tilde{\mathbf{x}}_i^c\|_2^2$ in Eq. (1.12) with $n\|\tilde{\mathbf{x}}_i^c\|_2^2$, such that the scaling step computes $\dot{\mathbf{x}}_i = s \cdot \frac{\tilde{\mathbf{x}}_i^c}{\|\tilde{\mathbf{x}}_i^c\|_2}$. We call it PairNorm-SI (for Scale Individually), which imposes more restriction on node representations, such that all have the same L_2-norm s. In practice we found that both PairNorm and PairNorm-SI work well for SGC, whereas PairNorm-SI provides better and more stable results for GCN and GAT. The reason why GCN and GAT require stricter normalization may be because they have more parameters and are more prone to overfitting. In all experiments, we employ PairNorm for SGC and PairNorm-SI for both GCN and GAT.

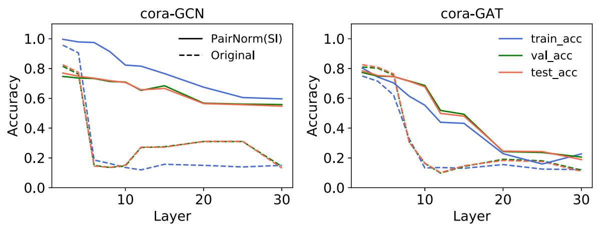


Figure 1.3: (best in color) Performance comparison of the original (dashed) vs. PairNorm-enhanced (solid) GCN and GAT models with increasing layers on `Cora`.

PairNorm is effective and efficient in solving the oversmoothing problem of GNNs. As a general normalization layer, it can be used for any GNN. Solid lines in Figure 1.1 present the performance of SGC on `Cora` with increasing number of layers, where we employ PairNorm after each graph convolution layer, as compared to 'vanilla' versions. Similarly, Figure 1.3 is for GCN and GAT (PairNorm is applied after the activation of each graph convolution). Note that the performance decay with PairNorm-at-work is much slower.

While PairNorm enables deeper models that are more robust to oversmoothing, it may seem odd that the overall test accuracy does not improve. In fact, the benchmark graph datasets often used in the literature require no more than 4 layers, after which performance decays (even if slowly). In the next section, we present a realistic use case setting for which deeper models are more likely to provide higher performance, where the benefit of PairNorm becomes apparent.

1.4.2 A Case Where Deeper GNNs are Beneficial

In general, oversmoothing gets increasingly more severe as the number of layers goes up. A task would benefit from employing PairNorm more if it required a large number of layers to achieve its best performance. To this effect we study the "missing feature setting", where a subset of the nodes *lack* feature vectors. Let $\mathcal{M} \subseteq \mathcal{V}_u$ be the set where $\forall m \in \mathcal{M}, \mathbf{x}_m = \emptyset$, i.e., all of their features are missing. We denote with $p = |\mathcal{M}|/|\mathcal{V}_u|$ the missing fraction. We call this variant of the task as semi-supervised node classification with missing vectors (SSNC-MV). Intuitively, one would require a larger number of propagation steps (hence, a deeper GNN) to be able to "recover" effective feature representations for these nodes.

SSNC-MV is a general and realistic problem that finds several applications in the real world. For example, the credit lending problem of identifying low- vs. high-risk customers (nodes) can be modeled as SSNC-MV where a large fraction of nodes do not exhibit any meaningful features (e.g., due to low-volume activity). In fact, many graph-based classification tasks with the cold-start issue (entity with no history) can be cast into SSNC-MV. To our knowledge, this is the *first* work to study the SSNC-MV problem using GNN models.

Figure 1.4 presents the performance of SGC, GCN, and GAT models on `Cora` with increasing number of layers, where we remove feature vectors from all the unlabeled nodes, i.e. $p = 1$. The models with PairNorm achieve a higher test accuracy compared to those without, which they typically reach at a larger number of layers.

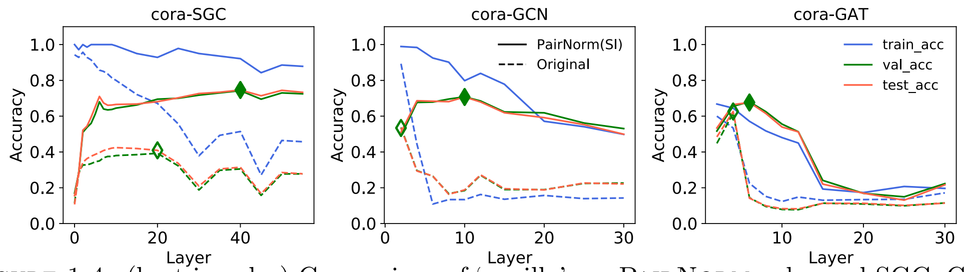


Figure 1.4: (best in color) Comparison of 'vanilla' vs. PairNorm-enhanced SGC, GCN, and GAT performance on `Cora` for $p = 1$. Green diamond symbols depict the layer at which validation accuracy peaks. PairNorm boosts overall performance by enabling more robust deep GNNs.

1.5 Experiments

In section 1.4 we have shown the robustness of PairNorm-enhanced models against increasing number of layers in SSNC problem. In this section we design extensive experiments to evaluate the effectiveness of PairNorm under the SSNC-MV setting, over SGC, GCN and GAT models.

1.5.1 Experiment Setup

Datasets. We use 4 well-known benchmark datasets in GNN domain: `Cora`, `Citeseer`, `Pubmed` [Sen+08], and `CoauthorCS` [Shc+18]. Their statistics are reported in Table 1.1. For `Cora`, `Citeseer` and `Pubmed`, we use the same dataset splits as [KW17], where all nodes outside train and validation are used as test set. For `CoauthorCS`, we randomly split all nodes into train/val/test as 3%/10%/87%, and keep the same split for all experiments.

TABLE 1.1: Dataset statistics.

Name	#Nodes	#Edges	#Features	#Classes	Label Rate
`Cora`	2708	5429	1433	7	0.052
`Citeseer`	3327	4732	3703	6	0.036
`Pubmed`	19717	44338	500	3	0.003
`CoauthorCS`	18333	81894	6805	15	0.030

Models. We use three different GNN models as our base model: SGC [Wu+19], GCN [KW17], and GAT [Vel+18]. We compare our PAIRNORM with residual connection method [He+16] over base models (except SGC since there is no "residual connected" SGC), as we surprisingly find it can slow down oversmoothing and benefit SSNC-MV problem. Similar to us, residual connection is a general technique that can be applied to any model without changing its architecture. We focus on the comparison between the base models and PAIRNORM-enhanced models, rather than achieving the state of the art performance for SSNC and SSNC-MV. There exist a few other work addressing oversmoothing [KBG19; LHW18; Ron+19; Xu+18] however they design specialized architectures and not simple "patch" procedures like PAIRNORM that can be applied on top of any GNN.

Hyperparameters. We search hyperparameter s of PAIRNORM in $\{0.1, 1, 10, 50, 100\}$ over validation set for SGC, while keeping it fixed at $s = 1$ for both GCN and GAT due to resource limitations. We set the #hidden units of GCN and GAT (#attention heads is set to 1) to 32 and 64 respectively for all datasets. Dropout with rate 0.6 and L_2 regularization with penalty $5 \cdot 10^{-4}$ are applied to GCN and GAT. For SGC, we vary number of layers in $\{1, 2, \dots 10, 15, \dots, 60\}$ and for GCN and GAT in $\{2, 4, \dots, 12, 15, 20, \dots, 30\}$.

Configurations. For PAIRNORM-enhanced models, we apply PAIRNORM after each graph convolution layer (i.e., after activation if any) in the base model. For residual-connected models with t skip steps, we connect the output of l-th layer to $(l + t)$-th, that is, $\mathbf{H}^{(l+t)}_{\text{new}} = \mathbf{H}^{(l+t)} + \mathbf{H}^{(l)}$ where $\mathbf{H}^{(l)}$ denotes the output of l-th graph convolution (after activation). For the SSNC-MV setting, we randomly erase p fraction of the feature vectors from nodes in validation and test sets (for which we input vector $\mathbf{0} \in \mathbb{R}^d$), whereas all training (labeled) nodes keep their original features (See 1.4.2). We run each experiment within 1000 epochs 5 times and report the average performance. We mainly use a single GTX-1080ti GPU, with some SGC experiments ran on an Intel i7-8700k CPU.

1.5.2 Experiment Results

We first show the global performance gain of applying PAIRNORM to SGC for SSNC-MV under varying feature missing rates as shown in Table 1.2. PAIRNORM-enhanced SGC performs similar or better over 0% missing, while it significantly outperforms vanilla SGC for most other settings, especially for larger missing rates. #L denotes the best number of layers for the model that yields the largest average validation accuracy (over 5 runs), for which we report the average test accuracy (Acc). Notice the larger #L values for SGC-PN compared to vanilla SGC, which shows the power of PAIRNORM for enabling "deep" SGC models by effectively tackling oversmoothing.

Similar to [Wu+19] who showed that the simple SGC model achieves comparable or better performance as other GNNs for various tasks, we found PAIRNORM-enhanced SGC to follow the same trend when compared with PAIRNORM-enhanced GCN and

GAT, for all SSNC-MV settings. Due to its simplicity and extreme efficiency, we believe PairNorm-enhanced SGC sets a strong baseline for the SSNC-MV problem.

Table 1.2: Comparison of 'vanilla' vs. PairNorm-enhanced SGC performance in `Cora`, `Citeseer`, `Pubmed`, and `CoauthorCS` for SSNC-MV problem, with missing rate ranging from 0% to 100%. Showing test accuracy at $\#L$ (K in Eq. 1.4) layers, at which model achieves best validation accuracy.

Missing Percentage		0%		20%		40%		60%		80%		100%	
Dataset	Method	Acc	#L	Acc	#L	Acc	#L	Acc	#L	Acc	#L	Acc	#L
`Cora`	SGC	**0.815**	4	**0.806**	5	0.786	3	0.742	4	0.733	3	0.423	15
	SGC-PN	0.811	7	0.799	7	**0.797**	7	**0.783**	20	**0.780**	25	**0.745**	40
`Citeseer`	SGC	0.689	10	0.684	6	**0.668**	8	**0.657**	9	0.565	8	0.290	2
	SGC-PN	**0.706**	3	**0.695**	3	0.653	4	0.641	5	**0.590**	50	**0.486**	50
`Pubmed`	SGC	0.754	1	0.748	1	0.723	4	0.746	2	0.659	3	0.399	35
	SGC-PN	**0.782**	9	**0.781**	7	**0.778**	60	**0.782**	7	**0.772**	60	**0.719**	40
`CoauthorCS`	SGC	0.914	1	0.898	2	0.877	2	0.824	2	0.751	4	0.318	2
	SGC-PN	**0.915**	2	**0.909**	2	**0.899**	3	**0.891**	4	**0.880**	8	**0.860**	20

We next employ PairNorm-si for GCN and GAT under the same setting, comparing it with the residual (skip) connections technique. Results are shown in Table 1.3 and Table 1.4 respectively for GCN and GAT. Due to space and resource limitations, we only show results for 0% and 100% missing rate scenarios. We observe similar trend for GCN and GAT: (1) vanilla model suffers from performance drop under SSNC-MV with increasing missing rate; (2) both residual connections and PairNorm-si enable deeper models and improve performance (note the larger #L and Acc); (3) GCN-PN and GAT-PN achieve performance that is comparable or better than just using skips; (4) performance can be further improved (albeit slightly) by using skips along with PairNorm-si.[3]

Table 1.3: Comparison of 'vanilla' and (PairNorm-si/ residual)-enhanced GCN performance on `Cora`, `Citeseer`, `Pubmed`, and `CoauthorCS` for SSNC-MV problem, with 0% and 100% feature missing rate. t represents the skip-step of residual connection.

Dataset	`Cora`				`Citeseer`				`Pubmed`				`CoauthorCS`			
Missing(%)	0%		100%		0%		100%		0%		100%		0%		100%	
Method	Acc	#L	Acc	#L	Acc	#L	Acc	#L	Acc	#L	Acc	#L	Acc	#L	Acc	#L
GCN	0.821	2	0.582	2	0.695	2	0.313	2	0.779	2	0.449	2	0.877	2	0.452	4
GCN-PN	0.790	2	$\underline{0.731}$	10	0.660	2	$\underline{0.498}$	8	0.780	30	$\underline{\mathbf{0.745}}$	25	0.910	2	$\underline{\mathbf{0.846}}$	12
GCN-t1	0.822	2	0.721	15	0.696	2	0.441	12	0.780	2	0.656	25	0.898	2	0.727	12
GCN-t1-PN	0.780	2	0.724	30	0.648	2	0.465	10	0.756	15	0.690	12	0.898	2	0.830	20
GCN-t2	0.820	2	0.722	10	0.691	2	0.432	20	0.779	2	0.645	20	0.882	4	0.630	20
GCN-t2-PN	0.785	4	**0.740**	30	0.650	2	**0.508**	12	0.770	15	0.725	30	0.911	2	0.839	20

1.6 Conclusion

We investigated the oversmoothing problem in GNNs and proposed PairNorm, a novel normalization layer that boosts the robustness of deep GNNs against oversmoothing. PairNorm is fast to compute, requires no change in network architecture nor any extra parameters, and can be applied to any GNN. Experiments on real-world

[3]Notice a slight performance drop when PairNorm is applied at 0% rate. For this setting, and the datasets we have, shallow networks are sufficient and smoothing through only a few (2-4) layers improves generalization ability for the SSNC problem (recall Figure 1.1 solid lines). PairNorm has a small reversing effect in these scenarios, hence the small performance drop.

TABLE 1.4: Comparison of 'vanilla' and (PAIRNORM-SI/ residual)-enhanced GAT performance on `Cora`, `Citeseer`, `Pubmed`, and `CoauthorCS` for SSNC-MV problem, with 0% and 100% feature missing rate. t represents the skip-step of residual connection.

Dataset	`Cora`				`Citeseer`				`Pubmed`				`CoauthorCS`			
Missing(%)	0%		100%		0%		100%		0%		100%		0%		100%	
Method	Acc	#L	Acc	#L	Acc	#L	Acc	#L	Acc	#L	Acc	#L	Acc	#L	Acc	#L
GAT	0.823	2	0.653	4	0.693	2	0.428	4	0.774	6	0.631	4	0.892	4	0.737	4
GAT-PN	0.787	2	<u>0.718</u>	6	0.670	2	<u>0.483</u>	4	0.774	12	**<u>0.714</u>**	10	0.916	2	<u>0.843</u>	8
GAT-t1	0.822	2	0.706	8	0.693	2	0.461	6	0.769	4	0.698	8	0.899	4	0.842	10
GAT-t1-PN	0.787	2	0.710	10	0.658	6	0.500	10	0.757	4	0.684	12	0.911	2	0.844	20
GAT-t2	0.820	2	0.691	8	s0.692	2	0.461	6	0.774	8	0.702	8	0.895	4	0.803	6
GAT-t2-PN	0.788	4	**0.738**	12	0.672	4	**0.517**	10	0.776	15	0.704	12	0.917	2	**0.855**	30

classification tasks showed the effectiveness of PAIRNORM, where it provides performance gains when the task benefits from more layers. Future work will explore other use cases of deeper GNNs that could further showcase PAIRNORM's advantages.

Chapter 2

GNN and Graph-Regularized PCA

Chapter based on: Lingxiao Zhao and Leman Akoglu. "Connecting graph convolutional networks and graph-regularized pca". In: *arXiv preprint arXiv:2006.12294* (2020).

2.1 Introduction

Graph neural networks (GNNs) are neural networks designed for the graph domain. Since the breakthrough of GCN [KW17], which notably improved performance on the semi-supervised node classification problem, many GNN variants have been proposed; including GAT [Vel+18], GraphSAGE [HYL17], DGI [Vel+19], GIN [Xu+19], PPNP and APPNP [KBG19], to name a few.

Despite the empirical successes of GNNs in both node-level and graph-level tasks, they remain not well understood due to limited systematic and theoretical analysis of GNNs. For example, researchers have found that GNNs, unlike their non-graph counterparts, suffer from performance degradation with increasing depth, their expressive power decaying exponentially in number of layers [OS20]. Such behavior is only partially explained by the oversmoothing phenomenon [LHW18; ZA20b]. Another surprising observation shows that a Simplified Graph Convolution model, named SGC [Wu+19], can achieve similar performance to various more complex GNNs on a variety of node classification tasks. Moreover, a simple baseline that does not utilize the graph structure altogether performs similar to state-of-the-art GNNs on graph classification tasks [Err+20]. These observations call attention to studies for a better understanding of GNNs [NM19; Mor+19; Xu+19; OS20; Lou20b; SR20]. (See Sec. 1.2 for more on understanding GNNs.)

Toward a systematic analysis and better understanding of GNNs, we establish a connection between the graph convolution operator of GCN (and PPNP) and Graph-regularized PCA (GPCA) [ZZ12], and show the similarity between GCN and stacking GPCA. This connection provides a deeper understanding of GCN's power and limitation. Empirically, we also find that GPCA performance matches that of many GNN baselines on benchmark semi-supervised node classification tasks. We argue that the simple GPCA should be a strong baseline in future. What is more, the unsupervised stacking GPCA can be viewed as "unsupervised GCN" and provides a straightforward, yet systematic way to initialize GCN training. We summarize our contributions as follows:

• **Connection between Graph Convolution and GPCA:** We establish the connection between the graph convolution operator of GCN (also PPNP) and the closed-form solution of graph-regularized PCA (GPCA) formulation. We demonstrate that a simple graph-regularized PCA paired with 1- or 2-layer MLP can achieve similar or even better results than state-of-the-art GNN baselines over several benchmark datasets. We further extend GPCA to (semi-)supervised setting which can generate embeddings using information of labels, which yields better performance on 3 out of 5 datasets. The outstanding performance of simple GPCA supports that the prowess of GCN on node classification task comes from graph based regularization. This motivates the study and design of other graph regularization techniques in the future.

• GPCANet: **New Stacking GPCA model:** Capitalizing on the connection between GPCA and graph convolution, we design a new GNN model called GPCANet shaped by (1) stacking multiple GPCA layers and nonlinear transformations, and (2) fine-tuning end-to-end via supervised training. GPCANet is a generalized GCN model with adjustable hyperparameters that control the strength of graph regularization of each layer. We show that with stronger regularization, we can train GPCANet with fewer (1–3) layers and achieve comparable performance to much deeper GCNs.

• **First initialization strategy for GNNs:** Capitalizing on the connection between GCN and GPCANet, we design a new strategy to initialize GCN training based on stacking GPCA, outperforming the popular Xaiver initialization [GB10]. We show that the GPCANet-initialization is extremely effective for training deeper GCNs, that significantly improves the convergence speed, performance, and robustness. Notably, GPCANet-initialization is general-purpose and also applies to other GNNs. To our knowledge, it is the first initialization method specifically designed for GNNs.

2.2 Related Work

Understanding GNNs. Our work concerns learning on a single graph, hence we limit discussion of related work to node-level GNNs. GCN's graph convolution is originally motivated from the approximation of graph filters in graph signal processing [KW17]. [NM19] show that graph convolution only performs low-pass filtering on original feature vectors, and also state a connection between graph filtering and Laplacian regularized least squares. Motivated by the oversmoothing phenomenon of graph convolution, [OS20] theoretically prove that GCN can only preserve information of node degrees and connected components when the number of layers goes to infinity, under some conditions of GCN weights. Recently several papers revisited the connection of graph convolution to graph-regularized optimization problem [Li+19b; Ma+20; PSH21; ZA20b; Zhu+21], which is originally discussed in graph signal processing [Shu+13]. More specifically, both [Ma+20] and [Zhu+21] relate graph-regularization optimization to several GNNs such as GCN [KW17], APPNP [KBG19], and GAT [Vel+18]. However, all previous work study these connections while ignoring the learnable parameters, which are essential for high-performance deep learning. Our work differs from these by establishing a stronger and closer connection to graph-regularized PCA that also takes learnable parameters into account.

Graph-regularized PCA. PCA and its variants are standard linear dimensionality reduction approaches. Several work extend PCA to graph-structured data, such as Graph-Laplacian PCA [Jia+13] and Manifold-regularized Matrix Factorization [ZZ12]. For other variants, see [Sha+16].

Stacking Models and Deep Learning. The connection between CNN and stacking PCA has been explored in PCANet [Cha+15], which demonstrated that the (unsupervised) simple stacking PCA works as well as supervised CNN over a large variety of vision tasks. The original PCANet is shallow and does not have nonlinear transformations, while PCANet+ [LTT17] overcomes these limitations and pushes the architecture much deeper. The idea of layerwise stacking for feature extraction is not new and was empirically observed to exhibit better representation ability in terms of classification. For a comprehensive review, we refer to [BCV13].

Initialization. Traditionally, neural networks (NNs) were initialized with random weights generated from Gaussian distribution with zero mean and a small standard deviation [KSH12]. As training deeper NNs became extremely difficult due to vanishing gradient and activation functions, [GB10] provided a specific weight initialization formula, named Xavier initialization, based on variance analysis without considering activation function. Xavier initialization is widely used for any type of NN even today, and it is the main initialization strategy used for GNNs. Later, [He+15] adapted Xavier initialization to ReLU activation by considering a multiplier. Taking another direction, [SMG13] analyzed the dynamics of training deep NNs and proposed random orthonormal initialization. [MM15] further improved orthonormal initialization for batch normalization [IS15]. Different from these data-independent approaches, others [Krä+16; Seu+17; Wag+13] have employed data-dependent techniques, like PCA, to initialize deep NNs. Although initialization has been widely studied for general NNs, no specific initialization has been proposed for GNNs. In this work, we propose a data-driven initialization technique (based on GPCA), specific to GNNs for the first time.

2.3 Graph Convolution and GPCA

2.3.1 Graph Convolution

Consider a node-attributed input graph $G = (V, E, X)$ with $|V| = n$ nodes and $|E| = m$ edges, where $X \in \mathbb{R}^{n\times d}$ denotes the node feature matrix with d features. Broadly, graph convolution operation convolves the features (or representations) over the graph structure.

GCN. Similar to other neural networks stacked with repeated layers, GCN contains multiple graph convolution layers each of which is followed by a nonlinear activation. Let $H^{(l)}$ be the l-th hidden layer representation, then, each GCN layer performs

$$H^{(l+1)} = \sigma(\tilde{A}_{\text{sym}} H^{(l)} W^{(l)}) \tag{2.1}$$

where $\tilde{A}_{\text{sym}} = \tilde{D}^{-\frac{1}{2}}(A+I)\tilde{D}^{-\frac{1}{2}}$ denotes the $n\times n$ symmetrically normalized adjacency matrix with self-loops, $\tilde{D}$ is the diagonal degree matrix where $\tilde{D}_{ii} = 1+\sum_{j=1}^{n} A_{ij}$, $W^{(l)}$ depicts the l-th layer parameters (to be learned), and σ is the nonlinear activation function. Formally, graph convolution is parameterized with W and maps an input X to a new representation Z as

$$Z = \tilde{A}_{\text{sym}} X W \ . \tag{2.2}$$

PPNP. For PPNP [KBG19], the features are first transformed by an MLP before convolving over the graph. Formally, the operation is revised as

$$Z = \mu \left(I - (1-\mu)\tilde{A}_{\text{sym}}\right)^{-1} \text{MLP}_W(X) = \left(I + \alpha\tilde{L}\right)^{-1} \text{MLP}_W(X) \tag{2.3}$$

where we replace μ with $\alpha = (1-\mu)/\mu$, $\tilde{L} := I - \tilde{A}_{\text{sym}}$ denotes the normalized graph Laplacian, and W depicts the learnable MLP parameters. As matrix inverse is expensive, an approximate version called APPNP that employs the power method [GV89] is often used in practice.

2.3.2 Graph-regularized PCA (GPCA)

As stated by [BCV13], “Although depth is an important part of the story, many *other priors* are interesting and can be conveniently captured when the problem is cast as one of learning a representation.” GPCA is one such representation learning technique with a *graph-based prior*.

Standard PCA learns k-dimensional projections $Z \in \mathbb{R}^{n\times k}$ of feature matrix $X \in \mathbb{R}^{n\times d}$, aiming to minimize the reconstruction error

$$\|X - ZW^T\|_F^2 \,, \tag{2.4}$$

subject to $W \in \mathbb{R}^{d\times k}$ being an orthonormal basis. GPCA extends this formalism to graph-structured data by additionally assuming either smoothing bases [Jia+13] or smoothing projections [ZZ12] over the graph. In this work we consider the latter case where low-dimensional projections are smooth over the input graph G, where $\tilde{L} = I - \tilde{A}_{\text{sym}}$ denotes its normalized Laplacian matrix. The objective formulation of GPCA is then given as

$$\min_{Z,W} \quad \|X - ZW^T\|_F^2 + \alpha \operatorname{Tr}(Z^T\tilde{L}Z) \qquad \text{s.t.} \quad W^TW = I \tag{2.5}$$

where α is a hyperparameter that balances reconstruction error and the variation of the projections over the graph. Note that the first part of Eq. (2.5), along with the constraint, corresponds to the objective of the original PCA, while the second part is a graph regularization term that aims to “smooth” the learned representations Z over the graph structure. As such, GPCA becomes the standard PCA when $\alpha = 0$.

Similar to PCA, the problem (2.5) is non-convex but has a closed-form solution [ZZ12]. Surprisingly, as we show, it has a close connection with the graph convolution formulation in Eq. (2.2). In the following, we give the GPCA solution and then detail its connection to graph convolution.

Theorem 2.3.1. *GPCA with formulation shown in (2.5) has the optimal solution (Z^*, W^*) following*

$$Z^* = (I + \alpha\tilde{L})^{-1}XW^* \;, \;\; \textit{and} \qquad W^* = (\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_k) \tag{2.6}$$

where $\mathbf{w}_1, ..., \mathbf{w}_k$ are the eigenvectors of $X^T(I+\alpha\tilde{L})^{-1}X$ corresponding to the largest k eigenvalues.

Proof. We give the proof in two steps.

Step 1: For a fixed W, Solve optimal Z^* as a function of W: When fixing W as constant, the problem becomes quadratic and convex. There is a unique solution, given by first-order optimal condition. Let ℓ denote the objective function as given in Eq. (2.5). Its gradient can be calculated as

$$\frac{\partial \ell}{\partial Z} = 2(I + \alpha\tilde{L})Z - 2XW \,. \tag{2.7}$$

Setting Eq. (2.7) to 0 leads to the solution $Z^* = (I + \alpha\tilde{L})^{-1}XW$.

Step 2: Replace Z with Z^*, Solve optimal W^*: Substituting Z in objective ℓ with $Z^* = (I + \alpha\tilde{L})^{-1}XW$, we reduce the optimization to

$$\min_{W, W^TW=I} \;\|X - (I+\alpha\tilde{L})^{-1}XWW^T\|_F^2 + \alpha \operatorname{Tr}\left[W^TX^T(I+\alpha\tilde{L})^{-1}\tilde{L}(I+\alpha\tilde{L})^{-1}XW\right] . \tag{2.8}$$

For this part only, let $M = (I + \alpha\tilde{L})^{-1}$ to simplify notation. We can show that Eq. (2.8) is equivalent to

$$\begin{aligned}\min_{W,W^TW=I} \quad & \operatorname{Tr}(XX^T + MXWW^TWW^TX^TM) \\ & - 2\operatorname{Tr}(MXWW^TX^T) + \alpha\operatorname{Tr}(W^TX^TM\tilde{L}MXW) \end{aligned} \tag{2.9}$$

Using the cyclic property of (Tr)ace (and plugging $(I + \alpha\tilde{L})^{-1}$ for M back), we can write it as

$$\max_{W,W^TW=I} \quad \operatorname{Tr}\left[W^TX^T(I+\alpha\tilde{L})^{-1}XW\right] . \tag{2.10}$$

Based on the spectral theorem of PSD matrices, the optimal solution W^* of problem Eq. (2.10) is the combination of eigenvectors, associated with the largest c eigenvalues of the graph-revised covariance matrix $X^T(I+\alpha\tilde{L})^{-1}X$. □

2.3.3 Connection between GCN and GPCA

Let $\Phi_\alpha := I + \alpha\tilde{L}$. The normalized Laplacian matrix $\tilde{L}$ has absolute eigenvalues bounded by 1, thus, all its positive powers have bounded operator norm. When $\alpha \leq 1$, Φ_α^{-1} can be decomposed into Taylor series as $(I+\alpha\tilde{L})^{-1} = I - \alpha\tilde{L} + \ldots + (-\alpha)^t\tilde{L}^t + \ldots$. The first-order truncated form (i.e. approximation) of the series is

$$(I+\alpha\tilde{L})^{-1} \approx I - \alpha\tilde{L} = (1-\alpha)I + \alpha\tilde{A}_{\text{sym}} . \tag{2.11}$$

When $\alpha = 1$, the first-order approximation of Z^* in Theorem 2.3.1 follows

$$Z^* \approx \tilde{A}_{\text{sym}}XW^* . \tag{2.12}$$

The (approximate) solution to GPCA in Eq. (2.12) matches the form of graph convolution operation in Eq. (2.2), with W^* plugged in as the eigenvectors of the matrix $X^T\Phi_\alpha^{-1}X$. In other words, there exists some parameter W^* with which GCN becomes the first-order approximation of GPCA.

To reiterate, a key contribution of this work is to show that the graph convolution operation in GCN can be viewed as the first-order approximation of GPCA with $\alpha = 1$ with a learnable W. Put differently, the first-order approximation of (unsupervised) GPCA with $\alpha = 1$ can be viewed as a graph convolution with a fixed, data-driven W. In other words, Notably, for $\alpha < 1$, Eq. (2.11) shows the connection between GPCA and graph convolution equipped with 1-step (scaled) residual connection.

2.3.4 Connection between PPNP and GPCA

Replacing the MLP in Eq. (2.3) with a single linear layer without activation results in $Z = \left(I + \alpha\tilde{L}\right)^{-1} XW$, which has exactly the same formulation as the solution Z^* in Theorem 2.3.1 Eq. (2.6). The connection states that the graph convolution in PPNP can be viewed as the GPCA solution with a learnable W. Interestingly, the empirical performance improvement of PPNP over GCN (see Table 2 in [KBG19]) may be explained through these connections that they have to GPCA; where PPNP relates to the exact solution of GPCA while GCN is related to its (first-order) approximation.

2.3.5 Supervised GPCA

The standard GPCA problem in (2.5) is unsupervised. Motiviated from LDA [BG98] and PLS [GK86], in this section we show how to extend it to the supervised setting, by learning embeddings that not only (1) provide good reconstruction and (2) vary smoothly over the graph structure, but also (3) highly correlate with the response variable(s). For simplicity of presentation, let $\mathbf{z} \in \mathbb{R}^d$ be a 1-d embedding and Y denote the response matrix (in the general case of multiple responses). We write the additional, i.e. (3)rd objective above, as[1]

$$\max_{\mathbf{z}} \quad \big[\mathrm{corr}(Y,\mathbf{z})\big]^T \big[\mathrm{corr}(Y,\mathbf{z})\big] \; \mathrm{var}(\mathbf{z}) \quad \equiv \quad \max_{\mathbf{z}} \; \mathbf{z}^T Y Y^T \mathbf{z} \tag{2.13}$$

The form of Eq. (2.13) and the variance-maximizing term $\mathrm{var}(\mathbf{z})$ are for mathematical convenience. Despite agnostic to labels, including $\mathrm{var}(\mathbf{z})$ is intuitive since an implicit objective of data projection (embedding) is to ensure that inherent variation in data is captured as much as possible. In general, we would aim to maximize the trace of $Z^T Y Y^T Z$ for multi-dimensional embeddings.

Interpretation. For semi-supervised node classification with c classes, let $\boldsymbol{L} \subset V$ denote the set of labeled nodes. For this task, $Y \in \{0,1\}^{n\times c}$ would encode the node labels where the v-th row of Y, denoted Y_v, depicts the one-hot encoded label for each $v \in \boldsymbol{L}$. For $u \in V \backslash \boldsymbol{L}$ with unknown labels, $Y_u = \mathbf{0}$, set as the c-dimensional all-zero vector. Then, $(YY^T)_{ij}$ is simply equal to 1 when nodes i and j share the same label, and otherwise 0 (either because they have different labels or labels are unknown). This term simply enforces the representations Z_i and Z_j of two same-labeled nodes to be similar. In a sense, YY^T adds "ghost" edges between the same-label nodes, further guiding the smoothness of their representations over this extended graph structure. We remark that earlier work [Gal+08] has heuristically introduced edges between same-label nodes to enhance a given graph for the node classification task. In this work, we have derived the theoretical underpinning for this strategy.

Supervised formulation. We have shown that requiring the embeddings to correlate with the known labels can be interpreted as additional smoothing over "ghost" edges between the same-label nodes in the graph. As such, we extend the GPCA problem in (2.5) to the (semi-)supervised setting as

$$\min_{Z,W} \quad \|X - ZW^T\|_F^2 + \alpha\,\mathrm{Tr}(Z^T \tilde{L}_{\text{spr}} Z) \qquad \text{s.t.} \quad W^T W = I \quad ; \tag{2.14}$$

$$\text{where } \tilde{L}_{\text{spr}} = I - \tilde{A}_{\text{spr}} \;, \quad \tilde{A}_{\text{spr}} = (1-\beta)\tilde{A}_{\text{sym}} + \beta D^{-\frac{1}{2}}(YY^T)D^{-\frac{1}{2}} \tag{2.15}$$

In Eq. Eq. (2.15), β is an additional hyperparameter for trading-off the graph-based regularization (i.e. smoothing) due to the actual input graph edges versus the ones introduced through YY^T between the nodes of the same label, and D is the diagonal matrix with $D_{ii} = \sum_{j=1}^{n}(YY^T)_{ij}$.

Theorem 2.3.2. *Supervised GPCA, as shown in (2.14) has the optimal solution (Z^*, W^*) following*

$$Z^* = (I + \alpha \tilde{L}_{\text{spr}})^{-1} X W^* \;, \; \textit{and} \qquad W^* = (\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_k) \tag{2.16}$$

where $\mathbf{w}_1, \ldots, \mathbf{w}_k$ are the top eigenvectors of the matrix $X^T(I+\alpha\tilde{L}_{\text{spr}})^{-1}X$, equivalently $X^T\big((1+\alpha)I - \big[\alpha(1-\beta)\tilde{A}_{sym} + \alpha\beta D^{-\frac{1}{2}}YY^TD^{-\frac{1}{2}}\big]\big)^{-1}X$, corresponding to the largest k eigenvalues.

Proof. The proof is similar to that of Theorem 2.3.1. □

2.3.6 Approximation and Complexity Analysis

According to formulations in Theorems 2.3.1 and 2.3.2, obtaining $Z^* \in \mathbb{R}^{n\times k}$ and $W^* \in \mathbb{R}^{d\times k}$ requires two demanding computations (1) the inverse of $\Phi_\alpha = (I + \alpha\mathbf{L}) \in \mathbb{R}^{n\times n}$, or in the supervised case $\Phi_\alpha = (I + \alpha\tilde{L}_{\text{spr}})$; and (2) top k eigenvectors of the

[1]For the optimization to be well-posed, constraints on $\mathbf{z}$ are required, omitted for simplicity of presentation.

matrix $X^T\Phi_\alpha^{-1}X \in \mathbb{R}^{d\times d}$. Eigen-decomposition takes $O(d^3)$ [PC99], which is scalable as d is usually small. Computing matrix inverse, on the other hand, can take $O(n^3)$ and require $O(n^2)$ memory, which would be infeasible for very large graphs.

To reduce computation and memory complexity, we instead approximately compute $F := \phi_\alpha^{-1}X$, which is a common term for both Z^* and W^*. We can equivalently write

$$(I+\alpha\mathbf{L})F = X \implies F + \alpha F = \alpha PF + X \implies F = \frac{\alpha}{1+\alpha}PF + \frac{1}{1+\alpha}X$$

for $P = \tilde{A}_{\text{sym}}$ in the unsupervised case and $P = (1-\beta)\tilde{A}_{\text{sym}} + \beta D^{-\frac{1}{2}}(YY^T)D^{-\frac{1}{2}}$ when supervised.

Then, we can iteratively (with total T iterations) use the power method [GV89] to compute F as

$$F^{(t+1)} \leftarrow \frac{\alpha}{1+\alpha}PF^{(t)} + \frac{1}{1+\alpha}X \tag{2.17}$$

where $t \in \{0, ..., T\}$ depicts the iteration and $F^{(0)} \in \mathbb{R}^{n\times d}$ is initialized as X (or randomly). For the supervised case, $PF^{(t)}$ is computed through a series of (from right to left) matrix-matrix products. This avoids the explicit construction of matrix YY^T in memory. Overall, solving for F takes $O(T(m+n)d)$ where m is the number of edges in the graph. The supervised case has an additional term $O(Td|\boldsymbol{L}|c)$ with c being the number of classes and $|\boldsymbol{L}| \leq n$ be the number of labeled nodes, which can also be upper-bounded by $O(T(m+n)d)$ when treating c as constant.

Having solved for F, we perform the matrix-matrix product $Z^* = FW^*$ in $O(ndk)$ and then the eigen-decomposition of X^TF in $O(d^3+nd^2) = O(nd^2)$ (for $n \geq d$). Assuming $O(d) = O(k)$, overall complexity for computing the 1-layer GPCA is given as $O(Tmd+Tnd+nd^2)$, which is *linear in the number of nodes and edges*. Note that empirically we found $5 \leq T \leq 10$ to be sufficient.

2.4 GPCANET: A Stacking GPCA Model

2.4.1 GPCANET

Thus far, we drew a connection between the geometrically motivated, manifold-based GPCA and the graph convolution operation of deep NN based GCN. Next we leverage this connection to design a new model called GPCANET that takes advantage of the relative strengths of each paradigm; namely, GPCA's ability to capture data variation and structure, and GCN's ability to capture multiple levels of abstraction (i.e. high-level concepts) through stacked layers and non-linearity.

In a nustshell, GPCANET is a stacking of multiple (unsupervised or supervised) GPCA layers and nonlinear transformations, which shares the same architecture as a multi-layer GCN. It consists of two main stages: (1) **Pre-training**, which *initializes* the layer-wise parameters through closed-form GPCA solutions, and (2) **End-to-end-training**, which *refines* these parameters through end-to-end gradient-based minimization of a global supervised loss criterion at the output layer.

We remark that GPCANET is *not* the same as GCN, as each layer uses the formulation in Thm.s 2.3.1 and 2.3.2 (with approximation shown in Sec. 2.3.6). In fact, when $\alpha = 1$ and $\beta = 0$, GPCANET is the GCN model initialized with GPCANET-initialization, which we discuss more in Sec. 2.4.2. In other words, GPCANET is a *generalized* GCN model with additional hyperparameters, α and β, controlling the strength of graph regularization based on the existing or "ghost" edges, respectively.

Algorithm 1 GPCANet **Forward Pass and Pre-training**

```
1: Input: graph $G = (V, E, X)$, GPCA hyper-parameter(s) $\alpha$ (and $\beta$ if supervised, $\beta = 0$ otherwise), #layers $L$, hidden layer sizes $\{d_1, \dots, d_L\}$, activation function $\sigma(\cdot)$, #approximation steps $T$
2: Output: pre-set layer-wise parameters $\{W^{(1)}, \dots, W^{(L)}\}$
3: Initialize $H^{(0)} := X$
4: for $l = 1$ to $L$ do
5:     Center $H^{(l-1)}$ by subtracting mean of row vectors
6:     $F \leftarrow H^{(l-1)}$
7:     for $t = 1$ to $T$ do
8:         $PF \leftarrow (1-\beta)\tilde{A}_{\text{sym}}F + \beta D^{-\frac{1}{2}}(YY^T)D^{-\frac{1}{2}}F$
9:         $F \leftarrow \frac{\alpha}{1+\alpha}PF + \frac{1}{1+\alpha}H^{(l-1)}$
10:    end for
11:    $W^{(l)} \leftarrow$ top $d_l$ eigenvectors of ${H^{(l-1)}}^T F$
12:    $H^{(l)} \leftarrow \sigma(FW^{(l)})$
13: end for
```

Forward Pass and Pre-training stage. During pre-training, weights of the l-th layer, denoted as $W^{(l)} \in \mathbb{R}^{d_{l-1} \times d_l}$, are pre-set (i.e. initialized) as the leading d_l eigenvectors of the matrix ${H^{(l-1)}}^T \Phi_\alpha^{-1} H^{(l-1)}$,[2] where $H^{(l-1)}$ is the representation as output by the $(l-1)$-th layer (with $H^{(0)} := X$), and Φ_α can be the unsupervised $(I + \alpha\mathbf{L})$ or the supervised $(I + \alpha\tilde{L}_{\text{spr}})$. The pre-training stage takes a single forward pass. Algo. 1 shows both forward pass during end-to-end-training and the pre-training procedure, where line 11 in blue is a step used *only* for pre-training.

Additional treatment for ReLU: Nonlinear transformations like ReLU improves model capacity, however at pre-training stage, it causes information loss as all negative values are truncated to 0. This hinders the advantage of using the leading d_l eigenvectors to initialize the weights so as to convey maximum variance (i.e. information) to the next layers. To address this issue, we instead use the leading $d_l/2$ eigenvectors $\{\mathbf{w}_i\}_{i=1}^{d_l/2}$ and their negatives $\{-\mathbf{w}_i\}_{i=1}^{d_l/2}$ to initialize $W^{(l)}$. Empirically we observe this always improves performance when using ReLU activation.

End-to-end training stage. Pre-training can be seen as an information-preserving initialization, as compared to an uninformative random initialization, after which we refine the layer-wise parameters via gradient-based optimization w.r.t. a supervised loss criterion at the output layer. Specifically for semi-supervised node classification, we perform an end-to-end training w.r.t. the cross-entropy loss on the labeled nodes. All parameters are updated jointly through backpropagation during this stage, with forward computation shown in Algo.1 (excluding line 11).

2.4.2 GPCANet-initialization for GCN

When we set $\alpha = 1$, $\beta = 0$, and approximate the matrix inverse $(I + \alpha\mathbf{L})^{-1}$ via first-order truncated Taylor expansion as shown in Eq. Eq. (2.11) , GPCANet has the same architecture with GCN. As such, we can use the pre-training stage of GPCANet to initialize GCN with only minor modification. Specifically, we replace lines 6 through 10 in Algo. 1 with the following single line:

$$F \leftarrow \tilde{A}_{\text{sym}} H^{(l-1)} \tag{2.18}$$

[2]If $d^{(l)}$ is greater than the number of eigenvectors, all eigenvectors are used, with additional vectors generated from random projection of eigenvectors.

The modified initialization is for GCN and is driven by the mathematical connection between GPCANET and GCN that we established. We expect that adapting it for other GNNs is also possible although we do not pursue this direction here.

2.5 Experiments

In this section we design extensive experiments to answer the following questions. (**Q1**) How does the simple, *unsupervised and shallow* GPCA compare to its multi-layer extension GPCANET, as well as to existing GNNs? (**Q2**) How does our extended, semi-supervised GPCA compare to the original, unsupervised GPCA? (**Q3**) Does GPCANET-initialization improve GCN accuracy and robustness?

2.5.1 Experimental Setup

Datasets. We focus on semi-supervised node classification (SSNC) and use 5 benchmark datasets: First three, CORA, CITESEER, PUBMED [Sen+08], are relatively small ($2K$ to $10K$ nodes) but widely-used citation graphs. For these we use the data splits in [KW17]. The others, ARXIV and PRODUCTS, are newest and much larger ($100K$ to $2000K$) node classification benchmarks from Open Graph Benchmark [Hu+20b], for which we use the official data splits. Data statistics can be found in Table 2.1.

TABLE 2.1: Statistics of used datasets.

DATASET	#NODES	#EDGES	#FEATURES	#CLASSES	TRAIN/VAL./TEST
CORA	2,708	5,429	1,433	7	5.2%/18.5%/36.9%
CITESEER	3,327	4,732	3,703	6	3.6%/15%/30%
PUBMED	19,717	44,338	500	3	0.3%/2.5%/5%
ARXIV	169,343	1,166,243	128	40	54%/18%/28%
PRODUCTS	2,449,029	61,859,140	100	47	8%/2%/90%

Baselines. We compare (unsupervised & semi-supervised) GPCA and GPCANET to state-of-the-art (SOTA) GNNs, including **GCN** [KW17], **APPNP** [KBG19], **GAT** [Vel+18], and GraphSAGE (**G-SAGE**) [HYL17].

Model configuration and training. For each dataset, we define a separate pool of values for the hyperparameters (HPs): learning rate, weight decay, number of layers, hidden size, dropout rate, and regularization trade-off terms α, β. For fair comparison, all models share the *same* HP pools during training.

We setup hyperparameters pool for each dataset, presented in Table 2.2. All methods use the *same* pool. The only exception is GPCA, as GPCA is just a 1-layer shallow model which can be trained with lager learning rate; we use 0.1 learning rate for it on all datasets.

Models are trained on every configuration across HP pools and picked based on validation performance. We use the Adam optimizer for all models. Learning rate is first manually tuned for each dataset to achieve stable training, and the same learning rate is fixed for all models—we empirically observed that learning rate is sensitive to datasets but insensitive to models. For GPCA and GPCANET, number of power iterations in Eq. (2.17) is always set to 5. All experiments use the maximum training epoch as 1000 and repeat 5 times. We mainly use a single GTX-1080ti GPU for small datasets CORA, CITESEER, and PUBMED. RTX-3090 GPU is used for ARXIV and PRODUCTS.

TABLE 2.2: Hyperparameters pool for each dataset, includes learning rate (LR), weight decay (WD), number of layers (#Layers), hidden size, dropout, α, and β. For ARXIV and PRODUCTS, weight decay is set as 0 because the dataset is large and no overfit happened. Same reason for choosing smaller dropout rate for them.

DATASET	LR	WD	#LAYERS	HIDDEN
CORA	0.001	[0.0005, 0.005, 0.05]	[2, 3, 5, 10, 15]	[128, 256]
CITESEER	0.001	[0.0005, 0.005, 0.05]	[2, 3, 5, 10, 15]	[128, 256]
PUBMED	0.001	[0.0005, 0.005, 0.05]	[2, 3, 5, 10, 15]	[128, 256]
ARXIV	0.005	0	[2, 3, 5, 10, 15]	[128, 256]
PRODUCTS	0.001	0	[2, 3, 5, 10, 15]	[128, 256]

DATASET	DROPOUT	α	β
CORA	[0, 0.5]	[1, 5, 10, 20, 50]	[0, 0.1, 0.2]
CITESEER	[0, 0.5]	[1, 5, 10, 20, 50]	[0, 0.1, 0.2]
PUBMED	[0, 0.5]	[1, 5, 10, 20, 50]	[0, 0.1, 0.2]
ARXIV	[0, 0.2]	[1, 5, 10, 20, 50]	0
PRODUCTS	[0, 0.1]	[1, 5, 10, 20, 50]	0

Mini-batch training. As nodes are not independent, GNN is mostly trained in full-batch under semi-supervised setting. We use full-batch training for all datasets except PRODUCTS, which is too large to fit into GPU memory during training. ClusterGCN [Chi+19], a subgraph based mini-batch training algorithm, is used to train GCN and GPCANET. For evaluation, we still use full-batch since a single forward pass can be conducted without memory issues. Initialization is also employed in full-batch.

Fair evaluation. Instead of picking the hyperparameter configurations manually, reported (test) performance is based on the *best* configuration selected using validation performance, where all models leverage the *same* hyperparameter pools. Further, each configuration from the pool is conducted 5 times to reduce randomness.

2.5.2 Q1: Performance of (Unsupervised) GPCA and GPCANET

GPCA. Having proved the mathematical connection between GPCA, GCN, and PPNP, we expect unsupervised GPCA ($\beta = 0$) to generate comparable representations. We perform GPCA with different $\alpha \in \{1, 5, 10, 20, 50\}$ to obtain node representations and pass those to a 1- or 2-layer MLP. We compare to GCN, APPNP, as well as other GNNs; GAT and G-SAGE.

The performance results are given in Table 2.4. Due to the scale of the largest two datasets, ARXIV and PRODUCTS, we list the reported performance at OGB-leaderboard[3] (depicted by *) for G-SAGE on both datasets, and that of (Cluster-)GAT on PRODUCTS.

We find that the simple 1-layer GPCA paired with MLP performs consistently better than the multi-layer GCN model across all 5 datasets. GPCA's performance is also comparable to or better than other SOTA GNNs. This is quite notable, given that GPCA is not only shallow but also unsupervised, whereas all other baselines are trained end-to-end, and with the exception of APPNP, they exhibit a multi-layer architecture. By carefully looking at the performance of GPCA with varying α (in Table 2.3), we find that different datasets have different best selected α^* (in Table 2.4 top to bottom: $\alpha^* = \{50, 5, 10, 20, 20\}$) but in general a relatively larger α (compared to graph convolution of GCN that is equivalent to $\alpha = 1$) is preferable for

[3]`https://ogb.stanford.edu/docs/leader_nodeprop/`

TABLE 2.3: Performance of unsupervised GPCA ($\beta = 0$) for varying α w.r.t. mean test accuracy and standard deviation (in parentheses). GPCA (best α) selects $\alpha \in \{1, 5, 10, 20, 50\}$ based on validation, whereas GPCA with specific α uses the specified fixed α.

	CORA	CITESEER	PUBMED	ARXIV	PRODUCTS
GPCA (BEST α)	81.10 (0.00)	71.80 (0.75)	78.78 (0.36)	71.86 (0.18)	79.23 (0.14)
GPCA-α=1	72.57 (0.79)	70.90 (0.58)	76.92 (0.30)	65.47 (0.26)	73.65 (0.07)
GPCA-α=5	80.95 (0.17)	71.80 (0.75)	**79.40** (0.29)	70.69 (0.11)	78.66 (0.09)
GPCA-α=10	**82.23** (0.58)	71.65 (0.53)	78.78 (0.36)	71.37 (0.09)	**79.24** (0.09)
GPCA-α=20	82.05 (0.54)	**72.15** (0.47)	78.15 (0.50)	**71.86** (0.18)	79.23 (0.14)
GPCA-α=50	81.10 (0.00)	71.50 (0.32)	78.00 (0.19)	71.48 (0.15	78.92 (0.10)

all datasets. Larger α implies stronger graph-regularization on the representations. The outstanding performance of the simple GPCA empirically confirms that the power of GNNs on the SSNC problem is mainly driven by graph regularization.

TABLE 2.4: Comparison btwn. unsupervised GPCA ($\beta = 0$), GPCANET, and existing (supervised) SOTA GNNs on 5 datasets, w.r.t. mean test accuracy and standard deviation (in parentheses) over 5 different seeds. Those marked with * are reported values at the OGB-leaderboard[3]. Highest mean performance is **in bold** and the second highest is <u>underlined</u>.

	GPCA	GPCANET	GCN	APPNP	GAT	G-SAGE
CORA	81.10 (0.00)	80.64 (0.33)	80.62 (0.90)	<u>81.35</u> (0.18)	79.27 (0.50)	**81.48** (0.83)
CITESEER	**71.80** (0.75)	<u>71.36</u> (0.21)	71.25 (0.05)	70.33 (0.75)	69.65 (0.59)	71.20 (0.92)
PUBMED	<u>78.78</u> (0.36)	78.52 (0.17)	78.42 (0.25)	**78.95** (0.36)	78.23 (0.54)	77.78 (0.29)
ARXIV	<u>71.86</u> (0.18)	**72.20** (0.15)	70.64 (0.17)	70.55 (0.27)	71.11 (0.11)	71.49*(0.27)
PRODUCTS	<u>79.23</u> (0.14)	**80.05** (0.29)	77.90 (0.33)	77.96 (0.34)	79.23*(0.78)	78.29*(0.16)

GPCANET. Compared to the 1-layer GPCA, GPCANET has a deeper architecture along with nonlinear activation function. Moreover, it employs hyperparameter α at every layer to control the degree of graph regularization. As each graph convolution has fixed level of graph regularization, one may hypothesize that increasing the number of layers (L) corresponds to increasing the degree of graph regularization. We empirically test this hypothesis using GPCANET, by varying both L (2 to 10) and α (0.1 to 10) to show their connection (hidden size is fixed as 128). The result is shown in Figure 2.1. The diagonal pattern (in dark blue) empirically suggests that increasing the number of layers has the same effect as increasing graph regularization via α.

The corresponding interaction between α and number of layers suggests that we can train a GPCANET with fewer number of layers yet achieve similar regularization by increasing α. Such a shallow model that in fact behaves like a deep one has the advantage of less memory requirement and faster training due to fewer parameters.

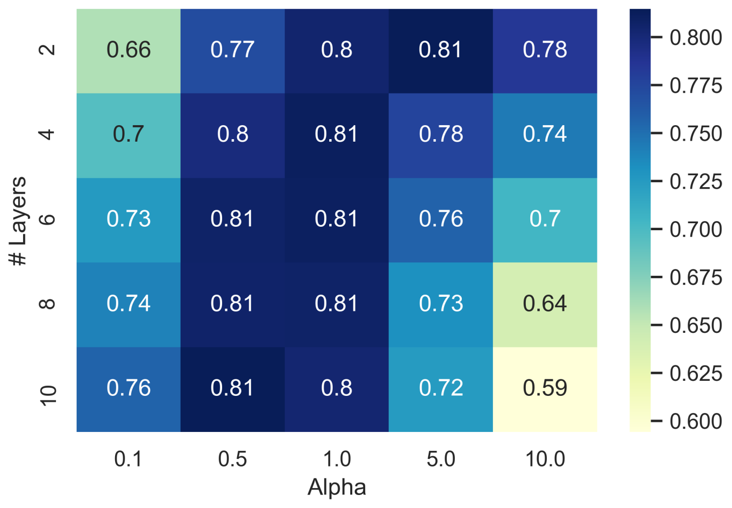


FIGURE 2.1: GPCANET performance (avg. over 5 seeds) with varying number of layers (L) and α on CORA. Increasing L has similar effect as increasing α. Results also hold for the other datasets.

To this end, we train 1–3-layer GPCANET with varying α, and select the best α and number of layers using validation set. We report test set performance in Table 2.4. We do not observe much improvement by GPCANET over other models on smaller datasets CORA, CITESEER, PUBMED, but notable gains on the larger ARXIV and PRODUCTS. As such, GPCANET enables shallow model training via tunable hyperparameter α, achieving comparable or better performance.

2.5.3 Q2: Unsupervised vs. Semi-supervised GPCA

TABLE 2.5: Comparison btwn. Supervised (S-)GPCA (β>0) and Unsupervised (U-)GPCA (β=0), w.r.t. mean test accuracy and standard deviation (in parentheses) over 5 different seeds. Also shown (bottom row) is the performance by the best method in Table 2.4. Highest mean performance is highlighted **in bold**.

	CORA	CITESEER	PUBMED
U-GPCA	81.10 (0.00)	71.80 (0.75)	78.78 (0.36)
S-GPCA (ALL β>0)	81.17 (0.27)	**73.20** (0.71)	**79.40** (0.69)
S-GPCA β=0.1	81.17 (0.27)	72.07 (0.37)	**79.40** (0.69)
S-GPCA β=0.2	**81.90** (0.00)	**73.20** (0.71)	78.73 (0.59)
TABLE 2.4 BEST	81.48 (0.83)	71.80 (0.75)	78.95 (0.36)

The representations generated by unsupervised GPCA does not use any label information from training data. In this work, we have extended GPCA to (semi-)supervised setting with an additional HP, namely $\beta \in [0, 1]$ that trades-off graph regularization due to the actual input graph edges versus the "ghost" ones added through YY^T. Overfitting can hurt performance when β is too large or when there is a distribution shift between the training and test sets. For ARXIV and PRODUCTS, we empirically observe that $\beta > 0$ always degrades performance, possibly because of the distribution difference between the training and test sets as described in OGB [Hu+20b]. Therefore we only study the effect of β on CORA, CITESEER and PUBMED. The pool for $\beta > 0$ is $\{0.1, 0.2\}$.

Results are shown in Table 2.5, where (ALL β>0) depicts the selected configuration for which S-GPCA achieves highest validation accuracy. The performance of the best method in Table 2.4, respectively of G-SAGE, (unsupervised) GPCA, and APPNP, is also shown for comparison. Notably, supervised GPCA provides a slight gain over

TABLE 2.6: Test set performance of GCN with Xaiver- versus GPCANET-initialization, w.r.t. varying number of layers (L) across all datasets. Each reported value is based on the best selected configuration on validation data. GPCANET-init. enables higher performance that is also stable with increasing depth.

DATASET	L=2	L=3	L=5	L=10	L=15
CORA XAIVER-INIT	80.62	**80.62**	79.40	76.37	66.07
CORA GPCANET-INIT	**81.67**	79.50	**80.90**	**79.82**	**78.00**
CITESEER XAIVER-INIT	71.25	**70.15**	**71.10**	61.90	57.40
CITESEER GPCANET-INIT	**71.27**	69.27	70.15	**68.67**	**67.87**
PUBMED XAIVER-INIT	**78.42**	**77.90**	77.07	77.00	45.80
PUBMED GPCANET-INIT	78.05	77.25	**78.07**	**77.80**	**78.03**
ARXIV XAIVER-INIT	69.61	**70.64**	70.33	68.32	61.68
ARXIV GPCANET-INIT	**69.76**	70.72	**70.52**	**69.77**	**66.28**
PRODUCTS XAIVER-INIT	77.90	78.65	78.08	76.27	74.70
PRODUCTS GPCANET-INIT	**78.13**	**78.71**	**78.22**	**77.47**	**75.90**

unsupervised GPCA across all 3 datasets, which also improves over the competing baseline methods.

2.5.4 Q3: GPCANET-initialization for GCN

Finally, we evaluate the effectiveness of GPCANET-initialization for GCN in terms of performance and robustness under different model sizes, i.e. number of layers L or number of training parameters. For comparison, Xavier initialization [GB10] is also used to initialize GCN.

We report the test set performance (averaged over 5 seeds) of the GCN model using both initializations in Table 2.6. The results show that GPCANET-initialization tends to outperform the widely-used Xavier initialization. The improvement grows with increasing number of layers, which is significant at large depths. Notably, GCN with GPCANET-initialization exhibits stable performance across all layers.

Besides study the mean performance, we further study whether GPCANET-initialization improves the training robustness, by reducing performance variation across different seeds. To this end, we first choose the best configuration for each initialization method based on validation performance, and train the GCN model with the chosen configuration using 100 random seeds.

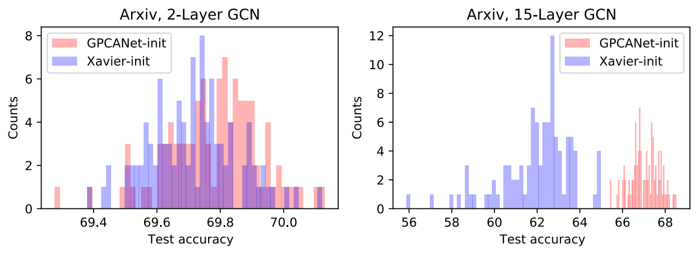


FIGURE 2.2: Comparison between Xavier-init. and GPCANET-init. in terms of test accuracy robustness over 100 seeds on ARXIV. GPCANET-init. enables robust training especially at larger depth.

In Figure 2.2 we present the histogram of test set accuracy over 100 runs with different seeds for ARXIV. (For results on other datasets, see Appendix. A6 in [ZA20a].) For both 2-layer and 15-layer GCN, GPCANET-initialization not only outperforms Xavier-initialization w.r.t. average performance, but also in terms of robustness, achieving much lower performance variation and few bad outliers, especially for deeper GCN. As such, it acts as a strong data-driven prior, facilitating the training of numerous parameters across many layers by identifying a promising region of the parameter space from which supervised fine-tuning is initiated.

2.6 Conclusion

In this work we have (1) discovered a mathematical connection between GPCA and graph convolution of GCN and PPNP; (2) extended GPCA to the (semi-)supervised setting; (3) proposed GPCAnet, by stacking GPCA and nonlinear activation, which is a generalized GCN model with an additional hyperparameter to control the degree of graph regularization, and (4) introduced the GPCAnet-initialization based on the established connection. Accordingly, we designed extensive experiments demonstrating that (*i*) the unsupervised shallow GPCA achieves comparable or better performance than GCN, APPNP, as well as other modern GNNs which suggests that graph convolution's power is mainly driven by graph regularization; (*ii*) semi-supervised GPCA helps improve performance and should be a powerful yet simple baseline in future research; (*iii*) GPCAnet enables the training of shallow models with competitive performance via increasing the degree of graph regularization at each layer, with reduced memory and training time cost; and finally (*iv*) GPCAnet-initialization acts as a strong data-driven prior for GCN training, enabling robust performance. Our methodological contributions (3) & 4) above) capitalize on the discovery of our theoretical findings (1) & 2)), shedding new light toward a better understanding and design of GNNs.

Part II

Graph-level Representation Learning

Chapter 3

Using Subgraphs to Boost Expressivity

Chapter based on: Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. "From Stars to Subgraphs: Uplifting Any GNN with Local Structure Awareness". In: *International Conference on Learning Representations.* 2022. URL: https://openreview.net/forum?id=Mspk_WYKoEH.

3.1 Introduction

Graphs are permutation invariant, combinatorial structures used to represent relational data, with wide applications ranging from drug discovery, social network analysis, image analysis to bioinformatics [Duv+15; Fan+19; Shi+19; Wu+20b]. In recent years, Graph Neural Networks (GNNs) have rapidly surpassed traditional methods like heuristically defined features and graph kernels to become the dominant approach for graph ML tasks.

Message Passing Neural Networks (MPNNs) [Gil+17] are the most common type of GNNs owing to their intuitiveness, effectiveness and efficiency. They follow a recursive aggregation mechanism where each node aggregates information from its immediate neighbors repeatedly. However, unlike simple multi-layer feedforward networks (MLPs) which are universal approximators of continuous functions [HSW89], MPNNs cannot approximate all permutation-invariant graph functions [Mar+19b]. In fact, their expressiveness is upper bounded by the first order Weisfeiler-Leman (1-WL) isomorphism test [Xu+19]. Importantly, researchers have shown that such 1-WL equivalent GNNs are not expressive, or powerful, enough to capture basic structural concepts, i.e., counting motifs such as cycles or triangles [Zhe+20; Arv+20] that are shown to be informative for bio- and chemo-informatics [Elt+19].

The weakness of MPNNs urges researchers to design more expressive GNNs, which are able to discriminate graphs from an isomorphism test perspective; [Che+19a] prove the equivalence between such tests and universal permutation invariant function approximation, which theoretically justifies it. As k-WL is strictly more expressive than 1-WL, many works [Mor+19; MRM20] try to incorporate k-WL in the design of more powerful GNNs, while others approach k-WL expressiveness indirectly from matrix invariant operations [Mar+19a; Mar+19b; KP19] and matrix language perspectives [Bal+21]. However, they require $O(k)$-order tensors to achieve k-WL expressiveness, and thus are not scalable or feasible for application on large, practical graphs. Besides, the bias-variance tradeoff between complexity and generalization [Nea+18] and the fact that almost all graphs (i.e. $O(2^{\binom{n}{2}})$ graphs on n vertices, [BES80]) can be distinguished by 1-WL challenge the necessity of developing such extremely expressive

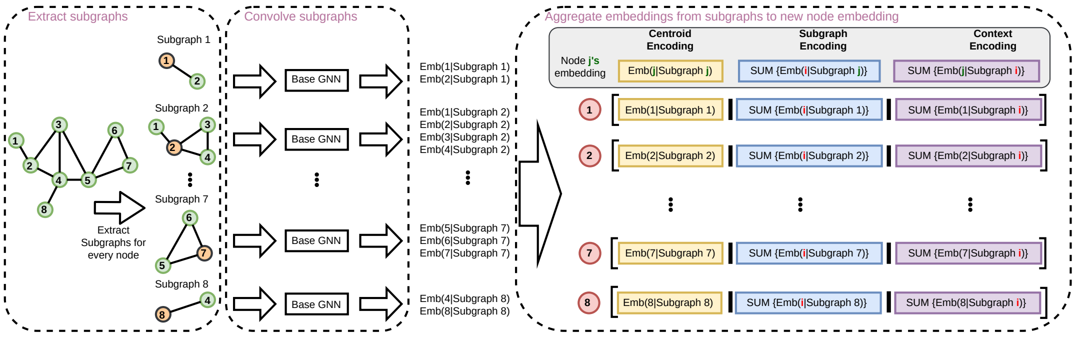


FIGURE 3.1: **Shown**: one GNN-AK$^+$ layer. For each layer, GNN-AK$^+$ first extracts n (# nodes) rooted subgraphs, and convolves all subgraphs with a base GNN as kernel, producing multiple rich subgraph-node embeddings of the form $\mathsf{Emb}(i \mid \mathrm{Sub}[j])$ (node i's embedding when applying a GNN kernel on subgraph j). From these, we extract and concatenate three encodings for a given node j: (i) centroid $\mathsf{Emb}(j \mid \mathrm{Sub}[j])$, (ii) subgraph $\sum_i \mathsf{Emb}(i \mid \mathrm{Sub}[j])$, and (iii) context $\sum_i \mathrm{Emb}(j \mid \mathrm{Sub}[i])$. GNN-AK$^+$ repeats the process for L layers, then sums all resulting node embeddings to compute the final graph embedding. As a weaker version, GNN-AK only contains encodings (i) and (ii).

models. In a complementary line of work, [Lou20b] sheds light on developing more powerful GNNs while maintaining linear scalability, finding that MPNNs can be universal approximators provided that nodes are sufficiently distinguishable. Relatedly, several works propose to add features to make nodes more distinguishable, such as identifiers [Lou20b], subgraph counts [Bou+20], distance encoding [Li+20], and random features [SYK21; Abb+21]. However, these methods either focus on handcrafted features which lose the premise of automatic learning, or create permutation sensitive features that hurt generalization.

Present Work. Our work stands between the two regimes of extremely expressive but unscalable k-order GNNs, and the limited expressiveness yet high scalability of MPNNs. Specifically, we propose a general framework that serves as a “wrapper” to uplift **any** GNN.

We observe that MPNNs' local neighbor aggregation follows a star pattern, where the representation of a node is characterized by applying an injective aggregator function as an encoder to the star subgraph (comprised of the central node and edges to neighbors). We propose a design which naturally generalizes from encoding the star to encoding a more flexibly defined subgraph, and we replace the standard injective aggregator with a GNN: in short, we characterize the new representation of a node by using a GNN to encode a locally induced encompassing subgraph, as shown in Fig.3.1. This uplifts GNN as a base model in effect by applying it on each *subgraph* instead of the whole input graph. This generalization is close to Convolutional Neural Networks (CNN) in computer vision: like the CNN that convolves image patches with a kernel to compute new pixel embeddings, our designed wrapper convolves subgraphs with a GNN to generate new node embeddings. Hence, we name our approach GNN-AK (GNN As Kernel). We show theoretically that GNN-AK is strictly more powerful than 1&2-WL with any MPNN as base model, and is not less powerful than 3-WL with PPGN [Mar+19a] used. We also give sufficient conditions under which GNN-AK can successfully distinguish two non-isomorphic graphs. Given this increase in expressive power, we discuss careful implementation strategies for GNN-AK, which allow us to carefully leverage multiple modalities of information from subgraph encoding, and resulting in an empirically more expressive version GNN-AK$^+$. As a result, GNN-AK and GNN-AK$^+$ induce a constant factor overhead in memory. To amplify our method's practicality, we further develop a subgraph sampling strategy inspired by Dropout

[Sri+14] to drastically reduce this overhead (1-3× in practice) without hurting performance. We conduct extensive experiments on 4 simulation datasets and 5 well-known real-world graph classification & regression benchmarks [Dwi+20; Hu+20b], to show significant and consistent practical benefits of our approach across different MPNNs and datasets. Specifically, GNN-AK$^+$ sets new state-of-the-art performance on ZINC, CIFAR10, and PATTERN – for example, on ZINC we see a relative error reduction of 60.3%, 50.5%, and 39.4% for base model being GCN [KW17], GIN [Xu+19], and (a variant of) PNA [Cor+20] respectively.

To summarize, our contributions are listed as follows:

- **A General GNN-AK Framework.** We propose GNN-AK (and enhanced GNN-AK$^+$), a general framework which uplifts any GNN by encoding local subgraph structure with a GNN.
- **Theoretical Findings.** We show that GNN-AK's expressiveness is strictly better than 1&2-WL, and is not less powerful than 3-WL. We analyze sufficient conditions for successful discrimination.
- **Effective and Efficient Realization.** We present effective implementations for GNN-AK and GNN-AK$^+$ to fully exploit all node embeddings within a subgraph. We design efficient online subgraph sampling to mitigate memory and runtime overhead while maintaining performance.
- **Experimental Results.** We show strong empirical results, demonstrating both expressivity improvements as well as practical performance gains where we achieve new state-of-the-art performance on several graph-level benchmarks.

Our implementation is easy-to-use, and directly accepts any GNN from PyG [FL19] for plug-and-play use. See code at https://github.com/GNNAsKernel/GNNAsKernel.

3.2 Related Work

Exploiting subgraph information in GNNs is not new; in fact, k-WL considers all k node subgraphs. [MOB18; Lee+19] exploit motif information within aggregation, and others [Bou+20; Bar+21] augment MPNN features with handcrafted subgraph based features. MixHop [AEH+19] directly aggregates k-hop information by using adjacency matrix powers, ignoring neighbor connections. Towards a meta-learning goal, G-meta [HZ20] applies GNNs on rooted subgraphs around each node to help transferring ability. Tahmasebi and Jegelka [TJ20] only theoretically justifies subgraph convolution with GNN by showing its ability in counting substructures. Zhengdao, Lei, Soledad, and Bruna [Zhe+20] also represent a node by encoding its local subgraph, however using non-scalable relational pooling. k-hop GNN [NDV20] uses k-egonet in a specially designed way: it encodes a rooted subgraph via sequentially passing messages from k-th hops in the subgraph to $k-1$ hops, until it reaches the root node, and use the root node as encoding of the subgraph. Ego-GNNs [SVH21] computes a context encoding with SGC [Wu+19] as the subgraph encoder, and only be studied on node-level tasks. Both k-hop GNN and Ego-GNNs can be viewed as a special case of GNN-AK. [You+21] designs ID-GNNs which inject node identity during message passing with the help of k-egonet, with k being the number of layers of GNN [HYL17]. Unlike GNN-AK which uses rooted subgraphs, [FYW20; TZK21; Bod+21a] design GNNs to use certain subgraph patterns (like cycles and paths) in message passing, however their preprocessing requires solving the subgraph isomorphism problem. [CMR21] explores reconstructing a graph from its subgraphs. A contemporary work [ZL21] also encodes rooted subgraphs with a base GNN but it essentially views a graph as a bag of subgraphs while GNN-AK modifies the 1-WL color refinement and has many iterations. Viewing graph as a bag of subgraphs is also explored in another contemporary work [Bev+22]. To summarize, our work differs by (i) proposing a general subgraph encoding framework motivated from theoretical Subgraph-1-WL for uplifting GNNs, and (ii) addressing scalability issues involved with using subgraphs, which poses significant challenges for subgraph-based methods in practice.

Improving Expressiveness of GNNs: Several works other than those mentioned in Sec.1.1 tackle expressive GNNs. Murphy, Srinivasan, Rao, and Ribeiro [Mur+19] achieve universality by summing permutation-sensitive functions across a combinatorial number of permutations, limiting feasibility. Dasoulas, Dos Santos, Scaman, and Virmaux [Das+20] adds node indicators to make them distinguishable, but at the cost of an invariant model, while Vignac, Loukas, and Frossard [VLF20] further addresses the invariance problem, but at the cost of quadratic time complexity. Corso, Cavalleri, Beaini, Liò, and Veličković [Cor+20] generalizes MPNN's default sum aggregator, but is still limited by 1-WL. Beani, Passaro, Létourneau, Hamilton, Corso, and Lió [Bea+21] generalizes spatial and spectral aggregation with >1-WL expressiveness, but using expensive eigendecomposition. Recently, [Bod+21b] introduce MPNNs over simplicial complexes that shares similar expressiveness as GNN-AK. [Yin+21] studies transformer with above 1-WL expressiveness. [AL21] surveys GNN expressiveness work.

Connections to CNN and k-WL: GNN-AK has a similar convolutional structure as CNN, and in fact historically many spatial GNNs are inspired by CNN; see Wu, Pan, Chen, Long, Zhang, and Philip [Wu+20b] for a detailed survey. The non-Euclidean nature of graphs makes such generalizations non-trivial. MoNet [Mon+17] introduces pseudo-coordinates for nodes, while PatchySAN [NAK16] learns to order and truncate neighboring nodes for convolution purposes. However, both methods aim to mimic the formulation of the CNN without admitting the inherent difference between graphs and images. In contrast, GNN-AK, generalizes CNN to graphs with a base GNN kernel, similar to how a CNN kernel encodes image patches. GNN-AK also shares connections with two variants of k-WL test algorithms: depth-k 1-dim WL [CFI92; Wei76] and deep WL [Arv+20]. The former recursively applies 1-WL to all size-k subgraphs, with slightly weaker expressiveness than k-WL, and the latter reduces the number of such subgraphs for the k-WL test. Instead of working on all $O(n^k)$ size-k subgraphs, we keep linear scalability by only applying 1-WL-equivalent MPNNs to $O(n)$ rooted subgraphs.

3.3 General Framework and Theory

We first introduce our setting and formalisms. Let $G = (\mathcal{V}, \mathcal{E})$ be a graph with node features $\mathbf{x}_i \in \mathbb{R}^d$, $\forall i \in \mathcal{V}$. We consider graph-level problems where the goal is to classify/regress a target y_G by learning a graph-level representation $\mathbf{h}_G$. Let $\mathcal{N}_k(v)$ be the set of nodes in the k-hop egonet rooted at node v. $\mathcal{N}(v) = \mathcal{N}_1(v) \backslash v$ denotes the immediate neighbors of node v. For $\mathcal{S} \subseteq \mathcal{V}$, let $G[\mathcal{S}]$ be the induced subgraph: $G[\mathcal{S}] = (\mathcal{S}, \{(i,j) \in \mathcal{E} | i \in \mathcal{S}, j \in \mathcal{S}\})$. Then $G[\mathcal{N}_k(v)]$ denotes the k-hop egonet rooted at node v. We also define $Star(v) = (\mathcal{N}_1(v), \{(v,j) \in \mathcal{E} | j \in \mathcal{N}(v)\})$ be the induced star-like subgraph around v. We use $\{\cdot\}$ denotes multiset, i.e. set that allows repetition.

Before presenting GNN-AK, we highlight the insights in designing GNN-AK and driving the expressiveness boost. **Insight 1: Generalizing star to subgraph.** In MPNNs, every node aggregates information from its immediate neighbors following a star pattern. Consequently, MPNNs fail to distinguish any non-isomorphic regular

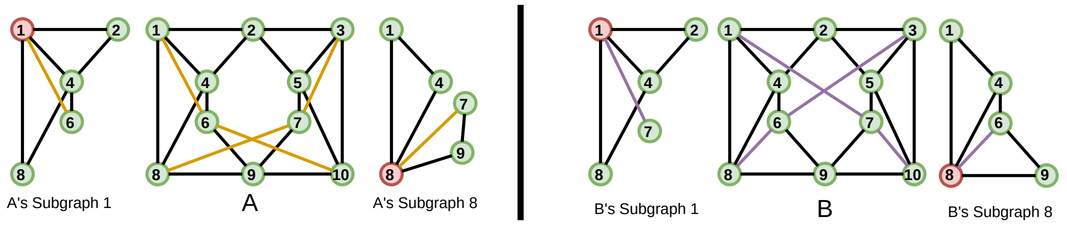


FIGURE 3.2: Two 4-regular graphs that cannot be distinguished by 1-WL. Colored edges are the difference between two graphs. Two 1-hop egonets are visualized while all other rooted egonets are ignored as they are same across graph A and graph B.

graphs where all stars are the same, since all nodes have the same degree. Even simply generalizing star to the induced, 1-hop egonet considers connections among neighbors, enabling distinguishing regular graphs. **Insight 2: Divide and conquer.** When two graphs are non-isomorphic, there exists a subgraph where this difference is captured (see Figure 3.2). Although a fixed-expressiveness GNN may not distinguish the two original graphs, it may distinguish the two *smaller* subgraphs, given that the required expressiveness for successful discrimination is proportional to graph size [Lou20a]. As such, GNN-AK divides the harder problem of encoding the whole graph to smaller and easier problems of encoding its subgraphs, and "conquers" the encoding with the base GNN.

3.3.1 From Stars to Subgraphs

We first take a close look at MPNNs, identifying their limitations and expressiveness bottleneck. MPNNs repeatedly update each node's embedding by aggregating embeddings from their neighbors a fixed number of times (layers) and computing a graph-level embedding $\mathbf{h}_G$ by global pooling. Let $\mathbf{h}_v^{(l)}$ denote the l-th layer embedding of node v. Then, MPNNs compute $\mathbf{h}_G$ by

$$\begin{aligned}\mathbf{h}_v^{(l+1)} &= \phi^{(l)}\Big(\mathbf{h}_v^{(l)}, f^{(l)}\Big(\big\{\mathbf{h}_u^{(l)} | u \in \mathcal{N}(v)\big\}\Big)\Big) \quad l = 0, ..., L-1 \\ \mathbf{h}_G &= \text{POOL}(\{\mathbf{h}_v^{(L)} | v \in \mathcal{V}\}) \end{aligned} \tag{3.1}$$

where $\mathbf{h}_i^{(0)} = \mathbf{x}_i$ is the original features, L is the number of layers, and $\phi^{(l)}$ and $f^{(l)}$ are the l-th layer update and aggregation functions. $\phi^{(l)}$, $f^{(l)}$ and POOL vary among different MPNNs and influence their expressiveness and performance. MPNNs achieve maximum expressiveness (1-WL) when all three functions are injective [Xu+19].

MPNNs' expressiveness upper bound follows from its close relation to the 1-WL isomorphism test [Mor+19]. Similar to MPNNs which repeatedly aggregate self and neighbor representations, at t-th iteration, for each node v, 1-WL test aggregates the node's own label (or color) $c_v^{(t)}$ and its neighbors' labels $\{c_u^{(t)} | u \in \mathcal{N}(v)\}$, and hashes this multi-set of labels $\left\{c_v^{(t)}, \{c_u^{(t)} | u \in \mathcal{N}(v)\}\right\}$ into a new, compressed label $c_v^{(t+1)}$. 1-WL outputs the set of all node labels $\left\{c_v^{(T)} | v \in \mathcal{V}\right\}$ as G's fingerprint, and decides two graphs to be non-isomorphic as soon as their fingerprints differ.

The hash process in 1-WL outputs a new label $c_v^{(t+1)}$ that uniquely characterizes the star graph $Star(v)$ around v, i.e. two nodes u, v are assigned different compressed labels only if $Star(u)$ and $Star(v)$ differ. Hence, it is easy to see that when two non-isomorphic unlabeled (i.e., all nodes have the same label) d-regular graphs have the same number of nodes, 1-WL cannot distinguish them. This failure limits the expressiveness of 1-WL, but also identifies its bottleneck: the star is not distinguishing enough. Instead, we propose to generalize the star $Star(v)$ to subgraphs, such as the egonet $G[\mathcal{N}_1(v)]$ and more generally k-hop egonet $G[\mathcal{N}_k(v)]$. This results in an improved version of 1-WL which we call *Subgraph-1-WL*. Formally,

Definition 3.3.1 (Subgraph-1-WL). Subgraph-1-WL generalizes the 1-WL graph isomorphism test algorithm by replacing color refinement (at iteration t) by $c_v^{(t+1)} = \text{HASH}(Star^{(t)}(v))$ with $c_v^{(t+1)} = \text{HASH}(G^{(t)}[\mathcal{N}_k(v)])$, $\forall v \in \mathcal{V}$ where $\text{HASH}(\cdot)$ is an injective function on graphs.

Note that an injective hash function for star graphs is equivalent to that for multisets, which is easy to derive [Zah+17]. In contrast, Subgraph-1-WL must hash a general subgraph , where an injective hash function for graphs is non-trivial (as hard as graph isomorphism). Thus, we derive a variant called Subgraph-1-WL* by using a weaker choice for $\text{HASH}(\cdot)$ – specifically, 1-WL. Effectively, we *nest* 1-WL inside Subgraph-1-WL. Formally,

Definition 3.3.2 (Subgraph-1-WL*). Subgraph-1-WL* is a less expressive variant of Subgraph-1-WL where $c_v^{(t+1)} = \text{1-WL}(G^{(t)}[\mathcal{N}_k(v)])$.

We further transfer Subgraph-1-WL to neural networks, resulting in GNN-AK whose expressiveness is upper bounded by Subgraph-1-WL. The natural transformation with maximum expressiveness is to replace the hash function with a universal subgraph encoder of $G[\mathcal{N}_k(v)]$, which is non-trivial as it implies solving the challenging graph isomorphism problem in the worst case. Analogous to using 1-WL as a weaker choice for $\text{HASH}(\cdot)$ inside Subgraph-1-WL*, we can use use any GNN (most practically, MPNN) as an encoder for subgraph $G[\mathcal{N}_k(v)]$. Let $G^{(l)}[\mathcal{N}_k(v)] = G[\mathcal{N}_k(v)|\mathbf{H}^{(l)}]$ be the attributed subgraph with hidden features $\mathbf{H}^{(l)}$ at the l-th layer. Then, GNN-AK computes $\mathbf{h}_G$ by

$$\mathbf{h}_v^{(l+1)} = \text{GNN}^{(l)}\Big(G^{(l)}[\mathcal{N}_k(v)]\Big) \quad l = 0, ..., L-1 \quad ; \quad \mathbf{h}_G = \text{POOL}(\{\mathbf{h}_v^{(L)} | v \in \mathcal{V}\}) \tag{3.2}$$

Notice that GNN-AK acts as a "wrapper" for any base GNN (mainly MPNN). This uplifts its expressiveness as well as practical performance as we demonstrate in the following sections.

3.3.2 Theory: Expressiveness Analysis

We next theoretically study the expressiveness of GNN-AK, by investigating the expressiveness of Subgraph-1-WL. We first establish that GNN-AK and Subgraph-1-WL have the same expressiveness under certain conditions. A GNN is able to distinguish two graphs if its embeddings for two graphs are not identical. A GNN is said to have the same expressiveness as a graph isomorphism test when for any two graphs the GNN outputs different embeddings if and only if (iff) the isomorphism test deems them non-isomorphic.

Theorem 3.3.3. *When the base model is an MPNN with sufficient number of layers and injective ϕ, f and POOL functions shown in Eq. (3.1), and MPNN-AK has an injective POOL function shown in Eq. (3.2), then MPNN-AK is as powerful as Subgraph-1-WL*.*

Proof. [Xu+19] proved that with sufficient number of layers and all injective functions, MPNN is as powerful as 1-WL. Then Eq. (3.2) outputs different vectors for two graphs iff Subgraph-1-WL* encodes different labels with $c_v^{(t+1)} = \text{1-WL}(G^{(t)}[\mathcal{N}_k(v)])$. With POOL in Eq. (3.2) also to be injective, MPNN-AK outputs different vectors iff Subgraph-1-WL* outputs different fingerprints for two graphs. Then MPNN-AK is as powerful as Subgraph-1-WL*. □

A more general version of Theorem 3.3.3 is that GNN-AK is as powerful as Subgraph-1-WL iff base GNN of GNN-AK is as powerful as the HASH function of Subgraph-1-WL in distinguishing subgraphs, following the same proof logic. The Theorem implies that we can characterize expressiveness of GNN-AK through studying Subgraph-1-WL and Subgraph-1-WL*.

Theorem 3.3.4. *Subgraph-1-WL* is strictly more powerful than 1&2-WL*[1].

Proof. We first prove that if two graphs are identified as isomorphic by Subgraph-1-WL*, they are also determined as isomorphic by 1-WL. Then we present a pair of non-isomorphic graphs that can be distinguished by Subgraph-1-WL* but not by 1-WL. Together these two imply that Subgraph-1-WL* is strictly more powerful than 1-WL. Comparing with 2-WL can be concluded from the fact that 1-WL and 2-WL are equivalent in expressiveness [Mar+19a]. In the proof we use 1-hop egonet subgraphs for Subgraph-1-WL*.

Assume graphs G and H have the same number of nodes (otherwise easily determined as non-isomorphic) and are two non-isomorphic graphs but Subgraph-1-WL* determines them as isomorphic. Then for any iteration t, set $\Big\{\text{1-WL}(G^{(t)}[\mathcal{N}_1(v)])|v \in \mathcal{V}_G\Big\}$ is the same as set $\Big\{\text{1-WL}(H^{(t)}[\mathcal{N}_1(v)])|v \in \mathcal{V}_H\Big\}$. Then there existing an ordering of nodes $v_1^G, ..., v_n^G$ and $v_1^H, ..., v_N^H$ with $N = |\mathcal{V}_G| = |\mathcal{V}_H|$, such that for any node order $i = 1, ..., N$, $\text{1-WL}(G^{(t)}[\mathcal{N}_1(v_i^G)]) = \text{1-WL}(H^{(t)}[\mathcal{N}_1(v_i^H)])$. This implies that structure $G[\mathcal{N}_1(v_i^G)]$ and $H[\mathcal{N}_1(v_i^H)]$ are not distinguishable by 1-WL. Hence $Star(v_i^G)$ and $Star(v_i^H)$ are hashed to the same label otherwise the 1-WL that includes the hashed result of $Star(v_i^G)$ and $Star(v_i^H)$ can also distinguish $G[\mathcal{N}_1(v_i^G)]$ and $H[\mathcal{N}_1(v_i^H)]$. Then for any iteration t and any node with order i, $\text{HASH}\Big(Star(v_i^{G^{(t)}})\Big) = \text{HASH}\Big(Star(v_i^{H^{(t)}})\Big)$ implies that 1-WL fails in distinguishing G and H. In fact if we replace the 1-WL hashing function in Subgraph-1-WL* to a stronger version $\text{HASH}\Big(\{\text{HASH}(Star(v^{G^{(t)}})), \text{1-WL}(G^{(t)}[\mathcal{N}_1(v)])\}\Big)$, this directly implies the above statement.

In Figure 3.2, two 4-regular graphs are presented that cannot be distinguished by 1-WL but can be distinguished by Subgraph-1-WL*. We visualize the 1-hop egonets that are structurally different among graphs A and B. It's easy to see that A's egonet $A[\mathcal{N}_1(1)]$ and B's egonet $B[\mathcal{N}_1(1)]$ can be distinguished by 1-WL, as degree distribution is not the same. Hence, A and B can be distinguished by Subgraph-1-WL*. □

A direct corollary from Theorem 3.3.3&3.3.4 is as follows, which is empirically verified in Table 3.2.

Corollary 3.3.5. *When MPNN is 1-WL expressive, MPNN-AK is strictly more powerful than 1&2-WL.*

Theorem 3.3.6. *When HASH(·) is 3-WL expressive, Subgraph-1-WL is no less powerful than 3-WL, that is, it can discriminate some graphs for which 3-WL fails.*

Proof. We prove by showing a pair of 3-WL failed non-isomorphic graphs can be distinguished by Subgraph-1-WL (see definition of "no less powerful" in [Zhe+20]: we call A is no/not less powerful than B if there exists a pair of non-isomorphic graphs that cannot be distinguished by B but can be distinguished by A.), assuming HASH(·)

[1] 1-WL and 2-WL are known to be equally powerful, see [AL21] and [Mar+19a].

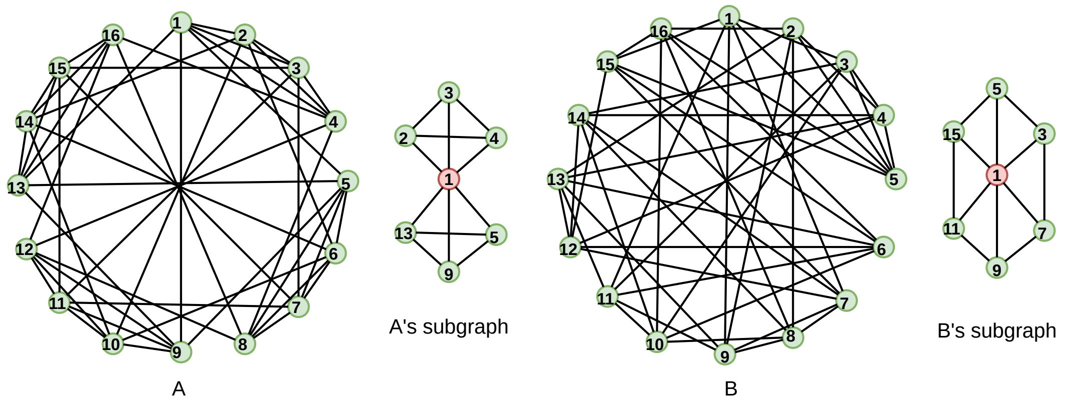


FIGURE 3.3: Two non-isomorphic strongly regular graphs that cannot be distinguished by 3-WL.

is 3-WL discriminative. Figure 3.3 shows two strongly regular graphs that can not be distinguished by 3-WL (any strongly regular graphs are not distinguishable by 3-WL [Arv+20]), along with their 1-hop egonet rooted subgraphs. Notice that all 1-hop egonet rooted subgraphs in A (also in B) are the same, resulting that Subgraph-1-WL can successfully distinguish A and B if HASH can distinguish the showed two 1-hop egonets. Now we prove that 3-WL can distinguish these two *subgraphs*. 3-WL constructs a coloring of 3-tuples of all vertices in a graph, and uses the histogram of colors of all k-tuples as fingerprint of the graph. Then, different 3-tuples correspond to different colors or bins in the histogram. As a triangle is a unique type of 3-tuple, at iteration 0 of 3-WL, the histogram of all 3-tuples counts the number of triangles in the graph. Notice that A's subgraph has $2 \times \binom{4}{3} = 8$ triangles, and B's subgraph contains 6 triangles. This implies that even at iteration 0, 3-WL can distinguish these two subgraphs. Hence, when HASH$(\cdot)$ is 3-WL expressive, Subgraph-1-WL can distinguish A and B. Therefore there exists a pair of 3-WL failed non-isomorphic graphs that can be distinguished by Subgraph-1-WL. □

A direct corollary from Theorem 3.3.3&3.3.6 is as follows, which is empirically verified in Table 3.2.

Corollary 3.3.7. *PPGN-AK can distinguish some 3-WL-failed non-isomorphic graphs.*

Theorem 3.3.8. *For any $k \geq 3$, there exists a pair of k-WL-failed graphs that **cannot** be distinguished by Subgraph-1-WL even with injective HASH$(\cdot)$ when t-hop egonets are used with $t \leq 4$.*

Theorem 3.3.8 is proven (see Appendix A.5 in [Zha+22c]) by observing that with limited t all rooted subgraphs of two non-isomorphic graphs from $CFI(k)$ family [CFI92] are isomorphic, i.e. local rooted subgraph is not enough to capture the "global" difference. This opens a future direction of generalizing rooted subgraph to general subgraph (as in k-WL) while keeping number of subgraphs in O$(|\mathcal{V}|)$.

Proposition 1 (sufficient conditions). For two non-isomorphic graphs G, H, Subgraph-1-WL with k-egonet can successfully distinguish them if: 1) for any node reordering $v_1^G, ..., v_{|\mathcal{V}_G|}^G$ and $v_1^H, ..., v_{|\mathcal{V}_H|}^H$, $\exists i \in [1, \max(|\mathcal{V}_G|, |\mathcal{V}_H|)]$ that $G[\mathcal{N}_k(v_i^G)]$ and $H[\mathcal{N}_k(v_i^H)]$ are non-isomorphic[2]; and 2) HASH$(\cdot)$ is discriminative enough that HASH$(G[\mathcal{N}_k(v_i^G)]) \neq$ HASH$(H[\mathcal{N}_k(v_i^H)])$.

[2]When $|\mathcal{V}_G| < |\mathcal{V}_H|$, $\forall i \in \{|\mathcal{V}_G|, |\mathcal{V}_G|+1, ..., |\mathcal{V}_H|\}$, let $G[\mathcal{N}_k(v_i^G)]$ denote an empty subgraph.

Proof. The proof is by contradiction. Let G and H be two non-isomorphic graphs and $|V_G| = |V_H|$ (if graph sizes are different then Subgraph-1-WL can trivially distinguish them). Now suppose that Prop. 1 is incorrect, i.e. Subgraph-1-WL cannot distinguish G and H even provided that the two conditions (1) and (2) are satisfied. Then, for any iteration of Subgraph-1-WL, G and H would have the *same* histogram of subgraph colors. Now we focus on iteration 0. Formally, let color $c^G_{v,i} = \text{HASH}(G[\mathcal{N}_k(v^G_i)])$ and $c^H_{v,i} = \text{HASH}(H[\mathcal{N}_k(v^H_i)])$ for a node order v (v^G_i maps index i to a node in G). According to Condition (1): for any node reordering $v^G_1, ..., v^G_{N_G}$ and $v^H_1, ..., v^H_{N_H}$, $\exists i \in [1, \max(N_G, N_H)]$ that $G[\mathcal{N}_k(v^G_i)]$ and $H[\mathcal{N}_k(v^G_i)]$ are non-isomorphic, then we know that $\exists i$ that $G[\mathcal{N}_k(v^G_i)]$ and $H[\mathcal{N}_k(v^H_i)]$ are non-isomorphic, hence $c^G_{v,i} \neq c^H_{v,i}$ since by Condition (2) HASH can distinguish these two subgraphs. However as two graphs have the same histogram of subgraph colors, there must be a $j \neq i$ such that $c^G_{v,i} = c^H_{v,j}$ and $c^G_{v,j} = c^H_{v,i}$. Then we can create a new node order m by swapping v^G_i and v^G_j, resulting $c^G_{m,i} = c^H_{m,i}$ and $c^G_{m,j} = c^H_{m,j}$. This process can be repeated until having a new node order w such that $\forall i \in 1, ..., |V_G|$, $c^G_{w,i} = c^H_{w,i}$. As HASH is discriminative enough according to Condition (2), this implies $\forall i \in 1, ..., |V_G|$, $G[\mathcal{N}_k(w^G_i)]$ and $H[\mathcal{N}_k(w^H_i)]$ are isomorphic, which contradicts with Condition (1). Thus, Prop. 1 must be true. $\square$

This implies subgraph size should be large enough to capture difference, but not larger which requires more expressive base model [Lou20b]. We empirically verify Prop. 1 in Table 3.3.

3.4 Concrete Realization

We first realize GNN-AK with two type of encodings, and then present an empirically more expressive version, GNN-AK$^+$, with (i) an additional *context encoding*, and (ii) a *subgraph pooling* design to incorporate distance-to-centroid, readily computed during subgraph extraction. Next, we discuss a random-walk based rooted subgraph extraction for graphs with small diameter to reduce memory footprint of k-hop egonets. We conclude this section with time and space complexity analysis.

Notation. Let $G = (\mathcal{V}, \mathcal{E})$ be the graph with $N = |\mathcal{V}|$, $G^{(l)}[\mathcal{N}_k(v)]$ be the k-hop egonet rooted at node $v \in \mathcal{V}$ in which $\mathbf{h}^{(l)}_u$ denotes node u's hidden representation for $u \in \mathcal{N}_k(v)$ at the l-th layer of GNN-AK. To simplify notation, we use $\text{Sub}^{(l)}[v]$ instead of $G^{(l)}[\mathcal{N}_k(v)]$ to indicate the the attribute-enriched induced subgraph for v. We consider all intermediate node embeddings across rooted subgraphs. Specifically, let $\mathsf{Emb}(i \mid \text{Sub}^{(l)}[j])$ denote node i's embedding when applying base $\text{GNN}^{(l)}$ on $\text{Sub}^{(l)}[j]$; we consider node embeddings for every $j \in \mathcal{V}$ and every $i \in \text{Sub}^{(l)}[j]$. Note that the base GNN can have multiple convolutional layers, and Emb refers to the node embeddings at the last layer before global pooling POOL_{GNN} that generates subgraph-level encoding.

Realization of GNN-AK. We can formally rewrite Eq. (3.2) as

$$\mathbf{h}^{(l+1)|\text{Subgraph}}_v = \text{GNN}^{(l)}\left(\text{Sub}^{(l)}[v]\right) := \text{POOL}_{\text{GNN}^{(l)}}\left(\left\{\mathsf{Emb}(i \mid \text{Sub}^{(l)}[v]) \mid i \in \mathcal{N}_k(v)\right\}\right) \tag{3.3}$$

We refer to the encoding of the rooted subgraph $\text{Sub}^{(l)}[v]$ in Eq. (3.3) as the *subgraph encoding*. Typical choices of $\text{POOL}_{\text{GNN}^{(l)}}$ are SUM and MEAN. As each rooted subgraph has a root node, $\text{POOL}_{\text{GNN}^{(l)}}$ can additionally be realized to differentiate the root node by self-concatenating its own representation, resulting in the following realization as each layer of GNN-AK:

$$\mathbf{h}_v^{(l+1)} = \text{FUSE}\Big(\mathbf{h}_v^{(l+1)|\text{Centroid}}, \mathbf{h}_v^{(l+1)|\text{Subgraph}}\Big) \quad \text{where } \mathbf{h}_v^{(l+1)|\text{Centroid}} := \mathsf{Emb}(v \mid \text{Sub}^{(l+1)}[v]) \tag{3.4}$$

where FUSE is concatenation or sum, and $\mathbf{h}_v^{(l)|\text{Centroid}}$ is referred to as the *centroid encoding*. The realization of GNN-AK in Eq.3.4 closely follows the theory in Sec.3.3.

Realization of GNN-AK$^+$. We further develop GNN-AK$^+$, which is more expressive than GNN-AK, based on two observations. **First**, we observe that Eq.3.4 does not fully exploit all information inside the rich intermediate embeddings generated for Eq.3.4, and propose an additional *context encoding*.

$$\mathbf{h}_v^{(l+1)|\text{Context}} := \text{POOL}_{\text{Context}}\Big(\big\{\mathsf{Emb}(v \mid \text{Sub}^{(l)}[j]) \mid \forall j \text{ s.t. } v \in \mathcal{N}_k(j)\big\}\Big) \tag{3.5}$$

Different from subgraph and centroid encodings, the context encoding captures views of node v from different subgraph contexts, or points-of-view. **Second**, GNN-AK extracts the rooted subgraph for every node with efficient k-hop propagation (complexity $O(k|\mathcal{E}|)$), along which the distance-to-centroid (D2C)[3] within each subgraph is readily recorded at no additional cost and can be used to augment node features; [Li+20] shows this theoretically improves expressiveness. Therefore, we propose to uses the D2C by default in two ways in GNN-AK$^+$: (i) augmenting hidden representation $\mathbf{h}_v^{(l)}$ by concatenating it with the encoding of D2C; (ii) using it to gate the subgraph and context encodings before $\text{POOL}_{\text{Subgraph}}$ and $\text{POOL}_{\text{Context}}$, with the intuition that embeddings of nodes far from v contribute differently from those close to v.

To formalize the gate mechanism guided by D2C, let $\mathbf{d}_{i|j}^{(l)}$ be the encoding of distance from node i to j at l-th layer[4]. Applying gating changes Eq. (3.5) to

$$\mathbf{h}_{\text{gated},v}^{(l+1)|\text{Context}} := \text{POOL}_{\text{Context}}\Big(\big\{\text{Sigmoid}(\mathbf{d}_{v|j}^{(l)}) \odot \mathsf{Emb}(v \mid \text{Sub}^{(l)}[j]) \mid \forall j \text{ s.t. } v \in \mathcal{N}_k(j)\big\}\Big) \tag{3.6}$$

where $\odot$ denotes element-wise multiplication. Similar changes apply to Eq. (3.3) to get $\mathbf{h}_{\text{gated},v}^{(l)|\text{Subgraph}}$.

Formally, each layer of GNN-AK$^+$ is defined as

$$\mathbf{h}_v^{(l+1)} = \text{FUSE}\Big(\mathbf{d}_{i|j}^{(l+1)}, \mathbf{h}_v^{(l+1)|\text{Centroid}}, \mathbf{h}_{\text{gated},v}^{(l+1)|\text{Subgraph}}, \mathbf{h}_{\text{gated},v}^{(l+1)|\text{Context}}\Big) \tag{3.7}$$

where FUSE is concatenation or sum. We illustrate in Figure 3.1 the l-th layer of GNN-AK($^+$).

Proposition 2. GNN-AK$^+$ is at least as powerful as GNN-AK.

Proof. When a pair of non-isomorphic graphs G and H cannot be distinguished by GNN-AK$^+$, for any layer l, the histogram of $h_v^{(l))}$ in G and H in Eq. 3.7 should be the same. Which implies that for any layer l, the histogram of $h_v^{(l))}$ in G and H in Eq. 3.4 should be the same. Then GNN-AK cannot distinguish G and H. So for any pair of non-isomorphic graphs, GNN-AK cannot distinguish them if GNN-AK$^+$ cannot distinguish them. Thus GNN-AK$^+$ is more powerful than GNN-AK. □

[3]We record D2C value for every node in every subgraph, and the value is categorical instead of continuous.

[4]The categorical D2C does not change across layers, but is encoded with different parameters in each layer.

Beyond k-egonet Subgraphs. The k-hop egonet (or k-egonet) is a natural choice in our framework, but can be too large when the input graph's diameter is small, as in social networks [Kle00], or when the graph is dense. To limit subgraph size, we also design a random-walk based subgraph extractor. Specifically, to extract a subgraph rooted at node v, we perform a fixed-length random walk starting at v, resulting in visited nodes $\mathcal{N}_{rw}(v)$ and their induced subgraph $G[\mathcal{N}_{rw}(v)]$. In practice, we use adaptive random walks as in Grover and Leskovec [GL16]. To reduce randomness, we use multiple truncated random walks and union the visited nodes as $\mathcal{N}_{rw}(v)$. Moreover, we employ online subgraph extraction during training that re-extracts subgraphs at every epoch, which further alleviates the effect of randomness via regularization.

Complexity Analysis. Assuming k-egonets as rooted subgraphs, and an MPNN as base model. For each graph $G = (\mathcal{V}, \mathcal{E})$, the subgraph extraction takes $O(k|\mathcal{E}|)$ runtime complexity, and outputs $|\mathcal{V}|$ subgraphs, which collectively can be represented as a union graph $\mathcal{G}_\cup = (\mathcal{V}_\cup, \mathcal{E}_\cup)$ with $|\mathcal{V}|$ disconnected components, where $|\mathcal{V}_\cup| = \sum_{v\in\mathcal{V}} |\mathcal{N}_k(v)|$ and $|\mathcal{E}_\cup| = \sum_{v\in\mathcal{V}} |\mathcal{E}_{G[\mathcal{N}_k(v)]}|$. GNN-AK($^+$) can be viewed as applying base GNN on the union graph. Assuming base GNN has $O(|\mathcal{V}| + |\mathcal{E}|)$ runtime and memory complexity, GNN-AK($^+$) has $O(|\mathcal{V}_\cup| + |\mathcal{E}_\cup|)$ runtime and memory cost. For rooted subgraphs of size s, GNN-AK($^+$) induces an $O(s)$ factor overhead over the base model.

3.5 Improving Scalability: SubgraphDrop

The complexity analysis reveals that GNN-AK($^+$) introduce a constant factor overhead (subgraph size) in runtime and memory over the base GNN. Subgraph size can be naturally reduced by choosing smaller k for k-egonet, or by ranking and truncating visited nodes in a random walk setting. However limiting to very small subgraphs tends to hurt performance as we empirically show in Table 3.7. Here, we present a different subsampling-based approach that carefully selects only a subset of the $|\mathcal{V}|$ rooted subgraphs. Further more, inspired from Dropout [Sri+14], we only drop subgraphs during training while still use all subgraphs when evaluation. Novel strategies are designed specifically for three type of encodings to eliminate the estimation bias between training and evaluation. We name it SubgraphDrop for dropping subgraphs during training. SubgraphDrop significantly reduces memory overhead while keeping performance nearly the same as training with all subgraphs. We first present subgraph sampling strategies, then introduce the designs of aligning training and evaluation. Fig. 3.4 provides a pictorial illustration.

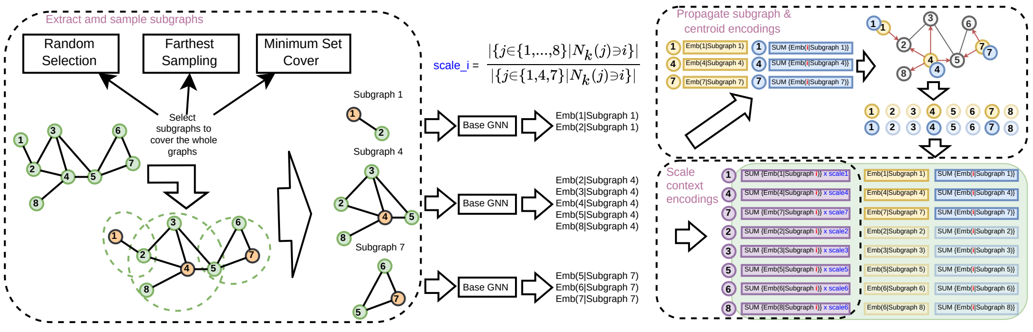


FIGURE 3.4: GNN-AK($^+$)-S with SubgraphDrop used in training. GNN-AK($^+$)-S first extracts subgraphs and subsamples $m \ll |\mathcal{V}|$ subgraphs to cover each node at least R times with multiple strategies. The base GNN is applied to compute all intermediate node embeddings in selected subgraphs. Context encodings are scaled to match evaluation. Subgraph and centroid encodings initially only exist for root nodes of selected subgraphs, and are propagated to estimate those of other nodes.

3.5.1 Subgraph Sampling Strategies

Intuitively, if u, v are directly connected in G, subgraphs $G[\mathcal{N}_k(u)]$ and $G[\mathcal{N}_k(v)]$ share a large overlap and may contain redundancy. With this intuition, we aim to sample only $m \ll |\mathcal{V}|$ minimally redundant subgraphs to reduce memory overhead. We propose three fast sampling strategies that select subgraphs to evenly cover the whole graph, where each node is covered by $\approx R$ (redundancy factor) selected subgraphs; R is a hyperparameter used as a sampling stopping condition. Then, GNN-AK($^+$)-S (with Sampling) has roughly R times the overhead of base model ($R \leq 3$ in practice). We remark that our subsampling strategies are randomized and fast, which are both desired characteristics for an online Dropout-like sampling in training.

- **Random sampling** selects subgraphs randomly until every node is covered $\geq R$ times.
- **Farthest sampling** selects subgraphs iteratively, starting at a random one and greedily selecting each subsequent one whose root node is farthest w.r.t. shortest path distance from those of already selected subgraphs, until every node is covered $\geq R$ times.
- **Min-set-cover sampling** initially selects a subgraph randomly, and follows the greedy minimum set cover algorithm to iteratively select the subgraph containing the maximum number of uncovered nodes, until every node is covered $\geq R$ times.

3.5.2 Training with SubgraphDrop

Although dropping redundant subgraphs greatly reduces overhead, it still loses information. Thus, as in Dropout [Sri+14], we only "drop" subgraphs during training while still using all of them during evaluation. Randomness in sampling strategies enforces that selected subgraphs differ across training epochs, preserving most information due to amortization. On the other hand, it makes it difficult for the three encodings during training to align with full-mode evaluation. Next, we propose an alignment procedure for each type of encoding.

Subgraph and Centroid Encoding in Training. Let $\mathcal{S} \subseteq \mathcal{V}$ be the set of root nodes of the selected subgraphs. When sampling during training, subgraph and centroid encoding can only be computed for nodes $v \in \mathcal{S}$, following Eq. (3.3) and Eq. (3.4), resulting incomplete subgraph encodings $\{\mathbf{h}_v^{(l)|\text{Subgraph}} | v \in \mathcal{S}\}$ and centroid encodings $\{\mathbf{h}_v^{(l)|\text{Centroid}} | v \in \mathcal{S}\}$. To estimate uncomputed encodings of $u \in \mathcal{V} \setminus \mathcal{S}$, we propose to *propagate* encodings from $\mathcal{S}$ to $\mathcal{V} \setminus \mathcal{S}$. Formally, let $k_{max} = \max_{u \in \mathcal{V} \setminus \mathcal{S}} \text{Dist}(u, \mathcal{S})$ where $\text{Dist}(u, \mathcal{S}) = \min_{v \in \mathcal{S}} \text{ShortestPathDistance}(u, v)$. Then we partition $\mathcal{U} = \mathcal{V} \setminus \mathcal{S}$ into $\{\mathcal{U}_1, ..., \mathcal{U}_{k_{max}}\}$ with $\mathcal{U}_d = \{u \in \mathcal{U} | \text{Dist}(u, \mathcal{S}) = d\}$. We propose to spread vector encodings of $\mathcal{S}$ out iteratively, i.e. compute vectors of $\mathcal{U}_d$ from $\mathcal{U}_{d-1}$. Formally, we have

$$\mathbf{h}_u^{(l)|\text{Subgraph}} = \text{Mean}\big(\{\mathbf{h}_v^{(l)|\text{Subgraph}} | v \in \mathcal{U}_{d-1}, (u, v) \in \mathcal{E}\}\big) \text{ for } d = 1, 2 \ldots k_{max}, \forall u \in \mathcal{U}_d, \tag{3.8}$$

$$\mathbf{h}_u^{(l)|\text{Centroid}} = \text{Mean}\big(\{\mathbf{h}_v^{(l)|\text{Centroid}} | v \in \mathcal{U}_{d-1}, (u, v) \in \mathcal{E}\}\big) \text{ for } d = 1, 2 \ldots k_{max}, \forall u \in \mathcal{U}_d, \tag{3.9}$$

Context Encoding in Training. Following Eq. (3.5), context encodings can be computed for every node $v \in \mathcal{V}$ as each node is covered at least R times during training with SubgraphDrop. However when $\text{POOL}_{\text{Context}}$ is $\text{SUM}(\cdot)$, the scale of $\mathbf{h}_v^{(l)|\text{Context}}$ is smaller than the one in full-mode evaluation. Thus, we scale the context encodings up to align with full-mode evaluation. Formally,

$$\mathbf{h}_v^{(l)|\text{Context}} = \frac{|\{j \in \mathcal{V} | \mathcal{N}_k(j) \ni v\}|}{|\{j \in \mathcal{S} | \mathcal{N}_k(j) \ni v\}|} \times \text{SUM}\Big(\big\{\mathsf{Emb}(v \mid \text{Sub}^{(l)}[j]) \mid \forall j \in \mathcal{S} \text{ s.t. } v \in \mathcal{N}_k(j)\big\}\Big) \tag{3.10}$$

When $\text{POOL}_{\text{Context}}$ is $\text{MEAN}(\cdot)$, the context encoding is computed without any modification.

3.6 Experiments

In this section we (1) empirically verify the expressiveness benefit of GNN-AK($^+$) on 4 simulation datasets; (2) show GNN-AK($^+$) boosts practical performance significantly on 5 real-world datasets; (3) demonstrate the effectiveness of SubgraphDrop; (4) conduct ablation studies of concrete designs. We mainly report the performance of GNN-AK$^+$, while still keep the performance of GNN-AK with GIN as base model for reference, as it is fully explained by our theory.

TABLE 3.1: Dataset statistics.

Dataset	Task Semantic	# Cls./Tasks	# Graphs	Ave. # Nodes	Ave. # Edges
EXP	Distinguish 1-WL failed graphs	2	1200	44.4	110.2
SR25	Distinguish 3-WL failed graphs	15	15	25	300
CountingSub.	Regress num. of substructures	4	1500 / 1000 / 2500	18.8	62.6
GraphProp.	Regress global graph properties	3	5120 / 640 / 1280	19.5	101.1
ZINC-12K	Regress molecular property	1	10000 / 1000 / 1000	23.1	49.8
CIFAR10	10-class classification	10	45000 / 5000 / 10000	117.6	1129.8
PATTERN	Recognize certain subgraphs	2	10000 / 2000 / 2000	118.9	6079.8
MolHIV	1-way binary classification	1	32901 / 4113 / 4113	25.5	54.1
MolPCBA	128-way binary classification	128	350343 / 43793 / 43793	25.6	55.4
MUTAG	Recognize mutagenic compounds	2	188	17.93	19.79
PTC-MR	Classify chemical compounds	2	344	14.29	14.69
PROTEINS	Classify Enzyme & Non-enzyme	2	1113	39.06	72.82
NCI1	Classify molecular	2	4110	29.87	32.30
IMDB-B	Classify movie	2	1000	19.77	96.53
RDT-B	Classify reddit thread	2	2000	429.63	497.75

Simulation Datasets: 1) EXP [Abb+21] contains 600 pairs of 1&2-WL failed graphs that are splited into two classes where each graph of a pair is assigned to two different classes. 2) SR25 [Bal+21] has 15 strongly regular graphs (3-WL failed) with 25 nodes each. SR25 is translated to a 15 way classification problem with the goal of mapping each graph into a different class. 3) Substructure counting (i.e. triangle, tailed triangle, star and 4-cycle) problems on random graph dataset [Zhe+20]. 4) Graph property regression (i.e. connectedness, diameter, radius) tasks on random graph dataset [Cor+20]. All simulation datasets are used to empirically verify the expressiveness of GNN-AK($^+$). **Large Real-world Datasets:** ZINC-12K, CIFAR10, PATTER from Benchmarking GNNs [Dwi+20] and MolHIV, and MolPCBA from Open Graph Benchmark [Hu+20b]. **Small Real-world Datasets**: MUTAG, PTC, PROTEINS, NCI1, IMDB, and REDDIT from TUDatset [Mor+20]. See Table 3.1 for all dataset statistics.

Baselines. We use GCN [KW17], GIN [Xu+19], PNA* [5] [Cor+20], and 3-WL powerful PPGN [Mar+19a] directly, which also server as base model of GNN-AK to

[5]PNA* is a variant of PNA that changes from using degree to scale embeddings to encoding degree and concatenate to node embeddings. This eliminates the need of computing average degree of datasets in PNA.

see its general uplift effect. GatedGCN [Dwi+20], DGN [Bea+21], PNA [Cor+20], GSN[Bou+20], HIMP [FYW20], and CIN [Bod+21a] are referenced directly from literature for real-world datasets comparison.

Hyperparameter and model configuration. To reduce the search space, we search hyperparameters in a two-phase approach: First, we search common ones (hidden size from [64, 128], number of layers L from [2,4,5,6], (sub)graph pooling from [SUM, MEAN] for each dataset using GIN based on validation performance, and fix it for any other GNN and GNN-AK($^+$). While hyperparameters may not be optimal for other GNN models, the evaluation is fair as there is no benefit for GNN-AK($^+$). Next, we search GNN-AK($^+$) exclusive ones (encoding types) over validation set using GIN-AK($^+$) and keep them fixed for other GNN-AK($^+$). We use a 1-layer base model for GNN-AK($^+$), with exceptions that we tune number of layers of base model (while keeping total number of layers fixed) for GNN-AK in simulation datasets (presented in Table 3.2). We use 3-hop egonets for GNN-AK($^+$), with exceptions that CIFAR10 uses 2-hop egonet due to memory constraint; PATTERN and RDT-B use random walk based subgraph with walk length=10 and repeat times=5 as their graphs are dense. For GNN-AK($^+$)-S, $R = 3$ is set as default. We use farthest sampling for molecular datasets ZINC, MolHIV, and MolPCBA; to speed up further, random sampling is used for CIFAR10 whose graphs are k-NN graphs; min-set-cover sampling is used for PATTERN to adapt random walk based subgraph. We use Batch Normalization and ReLU activation in all models. For optimization we use Adam with learning rate 0.001 and weight decay 0. All experiments are repeated 3 times to calculate mean and standard derivation. All experiments are conducted on RTX-A6000 GPUs.

3.6.1 Empirical Verification of Expressiveness

TABLE 3.2: Simulation dataset performance: *GNN-AK($^+$) boosts base GNN across tasks, empirically verifying expressiveness lift.* (ACC: accuracy, MAE: mean absolute error, OOM: out of memory)

Method	EXP	SR25	Counting Substructures (MAE)				Graph Properties ($\log_{10}$(MAE))		
	(ACC)	(ACC)	Triangle	Tailed Tri.	Star	4-Cycle	IsConnected	Diameter	Radius
GCN	50%	6.67%	0.4186	0.3248	0.1798	0.2822	-1.7057	-2.4705	-3.9316
GCN-AK$^+$	**100%**	6.67%	0.0137	0.0134	0.0174	0.0183	-2.6705	-3.9102	-5.1136
GIN	50%	6.67%	0.3569	0.2373	0.0224	0.2185	-1.9239	-3.3079	-4.7584
GIN-AK	**100%**	6.67%	0.0934	0.0751	0.0168	0.0726	-1.9934	-3.7573	-5.0100
GIN-AK$^+$	**100%**	6.67%	0.0123	0.0112	0.0150	0.0126	-2.7513	-3.9687	-5.1846
PNA*	50%	6.67%	0.3532	0.2648	0.1278	0.2430	-1.9395	-3.4382	-4.9470
PNA*-AK$^+$	**100%**	6.67%	0.0118	0.0138	0.0166	0.0132	-2.6189	-3.9011	-5.2026
PPGN	100%	6.67%	0.0089	0.0096	0.0148	0.0090	-1.9804	-3.6147	-5.0878
PPGN-AK$^+$	100%	**100%**	OOM	OOM	OOM	OOM	OOM	OOM	OOM

Table 3.2 presents the results on simulation datasets. To save space we present GNN-AK$^+$ with different base models but only one one version of GNN-AK: GIN-AK. All GNN-AK($^+$) variants perform perfectly on EXP, while only PPGN alone do so previously. Moreover, PPGN-AK$^+$ reaches perfect accuracy on SR25, while PPGN fails. Similarly, GNN-AK($^+$) consistently boosts all MPNNs for substructure and graph property prediction (PPGN-AK$^+$ is OOM as it is quadratic in input size).

In Table 3.3 we look into PPGN-AK's performance on SR25 as a function of k-egonets ($k \in [1, 2]$), as well as the number of PPGN (inner) layers and (outer) iterations for PPGN-AK. We find that at least 2 inner layers is needed with 1-egonet subgraphs to achieve top performance. With 2-egonets more inner layers helps, although performance is sub-par, attributed to PPGN's disability to distinguish larger (sub)graphs, aligning with Proposition 1 (Sec. 3.3.2).

Table 3.3: PPGN-AK expressiveness on SR25.

Base PPGN's #L	1-hop egonet			2-hop egonet		
PPGN-AK's #L	1	2	3	1	2	3
1	26.67%	100%	100%	26.67%	26.67%	46.67%
2	33.33%	100%	100%	26.67%	26.67%	53.33%
3	26.67%	100%	100%	33.33%	26.67%	53.33%

3.6.2 Comparing with SOTA and Generality

Having studied expressiveness tasks, we turn to performance on real-world datasets, as shown in Table 3.4. We observe similar performance lifts across all datasets and base GNNs (we omit PPGN due to scalability), demonstrating our framework's generality. Remarkably, GNN-AK$^+$ sets new SOTA performance for several benchmarks – ZINC, CIFAR10, and PATTERN – with a relative error reduction of 60.3%, 50.5%, and 39.4% for base model being GCN, GIN, and PNA* respectively.

Table 3.4: Real-world dataset performance: *GNN-AK$^+$ achieves SOTA performance for ZINC-12K, CIFAR10, and PATTERN.* (OOM: out of memory, −: missing values from literature)

Method	ZINC-12K (MAE)	CIFAR10 (ACC)	PATTERN (ACC)	MolHIV (ROC)	MolPCBA (AP)
GatedGCN	0.363 ± 0.009	69.37 ± 0.48	84.480 ± 0.122	−	−
HIMP	0.151 ± 0.006	−	−	0.7880 ± 0.0082	−
PNA	0.188 ± 0.004	70.47 ± 0.72	86.567 ± 0.075	0.7905 ± 0.0132	0.2838 ± 0.0035
DGN	0.168 ± 0.003	72.84 ± 0.42	86.680 ± 0.034	0.7970 ± 0.0097	0.2885 ± 0.0030
GSN	0.115 ± 0.012	−	−	0.7799 ± 0.0100	−
CIN	**0.079 ± 0.006**	−	−	**0.8094 ± 0.0057**	−
GCN	0.321 ± 0.009	58.39 ± 0.73	85.602 ± 0.046	0.7422 ± 0.0175	0.2385 ± 0.0019
GCN-AK$^+$	0.127 ± 0.004	72.70 ± 0.29	**86.887 ± 0.009**	0.7928 ± 0.0101	0.2846 ± 0.0002
GIN	0.163 ± 0.004	59.82 ± 0.33	85.732 ± 0.023	0.7881 ± 0.0119	0.2682 ± 0.0006
GIN-AK	0.094 ± 0.005	67.51 ± 0.21	86.803 ± 0.044	0.7829 ± 0.0121	0.2740 ± 0.0032
GIN-AK$^+$	**0.080 ± 0.001**	72.19 ± 0.13	86.850 ± 0.057	0.7961 ± 0.0119	**0.2930 ± 0.0044**
PNA*	0.140 ± 0.006	73.11 ± 0.11	85.441 ± 0.009	0.7905 ± 0.0102	0.2737 ± 0.0009
PNA*-AK$^+$	0.085 ± 0.003	OOM	OOM	0.7880 ± 0.0153	0.2885 ± 0.0006
GCN-AK$^+$-S	0.127 ± 0.001	71.93 ± 0.47	86.805 ± 0.046	0.7825 ± 0.0098	0.2840 ± 0.0036
GIN-AK$^+$-S	0.083 ± 0.001	72.39 ± 0.38	86.811 ± 0.013	0.7822 ± 0.0075	0.2916 ± 0.0029
PNA*-AK$^+$-S	0.082 ± 0.000	**74.79 ± 0.18**	86.676 ± 0.022	0.7821 ± 0.0143	0.2880 ± 0.0012

We also report additional results on several smaller datasets from TUDataset [Mor+20], with their statistics reported in last block of Table 3.1. The training setting and evaluation procedure follows [Xu+19] exactly, where we perform 10-fold cross-validation and report the average and standard deviation of validation accuracy across the 10 folds within the cross-validation. We take results of existing baselines directly from [Bod+21a] with their method labeled as CIN, for references to all baselines see [Bod+21a]. The result is shown in Table 3.5.

We mark that the performance of GIN-AK$^+$ over IMDB-B is not improved because each graph in the dataset is an egonet, hence all nodes have the same rooted subgraph – the whole graph. The performance of MUTAG and PTC is very unstable, given these datasets are too small: 188 and 344, respectively, and the evaluation method is based on average 10 validation curves over 10 folds.

3.6.3 Scaling up by Subsampling

In some cases, GNN-AK ($^+$)'s overhead leads to OOM, especially for complex models like PNA* that are resource-demanding. Sampling with SubgraphDrop enables training using practical resources. Notably, GNN-AK$^+$-S models, shown at the end of Table

TABLE 3.5: Results on TU Datasets. First section contains methods of graph kernels, second section has GNNs, and third is the method in [Bod+21a]. The top two are highlighted by **First**, **Second**, **Third**.

Dataset	MUTAG	PTC	PROTEINS	NCI1	IMDB-B	RDT-B
RWK	79.2±2.1	55.9±0.3	59.6±0.1	>3 days	N/A	N/A
GK ($k = 3$)	81.4±1.7	55.7±0.5	71.4±0.31	62.5±0.3	N/A	N/A
PK	76.0±2.7	59.5±2.4	73.7±0.7	82.5±0.5	N/A	N/A
WL kernel	90.4±5.7	59.9±4.3	75.0±3.1	**86.0±1.8**	73.8±3.9	81.0±3.1
DCNN	N/A	N/A	61.3±1.6	56.6±1.0	49.1±1.4	N/A
DGCNN	85.8±1.8	58.6±2.5	75.5±0.9	74.4±0.5	70.0±0.9	N/A
IGN	83.9±13.0	58.5±6.9	76.6±5.5	74.3±2.7	72.0±5.5	N/A
GIN	89.4±5.6	64.6±7.0	76.2±2.8	82.7±1.7	75.1±5.1	**92.4±2.5**
PPGNs	9 0.6±8.7	66.2±6.6	**77.2±4.7**	83.2±1.1	73.0±5.8	N/A
Natural GN	89.4±1.6	66.8±1.7	71.7±1.0	82.4±1.3	73.5±2.0	N/A
GSN	**92.2 ± 7.5**	**68.2 ± 7.2**	76.6 ± 5.0	83.5 ± 2.0	**77.8 ± 3.3**	N/A
SIN	N/A	N/A	76.4 ± 3.3	82.7 ± 2.1	**75.6 ± 3.2**	92.2 ± 1.0
CIN	**92.7 ± 6.1**	**68.2 ± 5.6**	**77.0 ± 4.3**	**83.6 ± 1.4**	**75.6 ± 3.7**	**92.4 ± 2.1**
GIN-AK$^+$	**91.3 ± 7.0**	**67.7 ± 8.8**	**77.1 ± 5.7**	**85.0 ± 2.0**	75.0 ± 4.2	**94.8 ± 0.8**

3.4, do not compromise and can *even improve* performance as compared to their non-sampled counterpart, in alignment with Dropout's benefits [Sri+14]. Next we evaluate resource-savings, specifically on ZINC-12K and CIFAR10. Table 3.6 shows that GIN-AK$^+$-S with varying R provides an effective handle to trade off resources with performance. Importantly, the rate in which performance decays with smaller R is much lower than the rates at which runtime and memory decrease.

TABLE 3.6: Resource analysis of SubgraphDrop.

Dataset		GIN-AK$^+$-S					GIN-AK$^+$	GIN
		R=1	R=2	R=3	R=4	R=5		
ZINC-12K	MAE	0.1216	0.0929	0.0846	0.0852	0.0854	0.0806	0.1630
	Runtime (S/Epoch)	10.8	11.2	12.0	12.4	12.5	9.4	6.0
	Memory (MB)	392	811	1392	1722	1861	1911	124
CIFAR10	ACC	71.68	72.07	72.39	72.20	72.32	72.19	59.82
	Runtime (S/Epoch)	80.7	89.1	100.5	110.9	119.7	241.1	55.0
	Memory (MB)	2576	4578	6359	8716	10805	30296	801

3.6.4 Ablation Study

We present ablation results on various structural components of GNN-AK. Table 3.7 shows the performance of GIN-AK for varying size egonets with k. We find that larger subgraphs tend to yield improvement, although runtime-performance trade-off may vary by dataset. Notably, simply 1-egonets are enough for CIFAR10 to uplift performance of the base GIN considerably.

TABLE 3.7: Effect of various k-egonet size.

k of GIN-AK$^+$	ZINC-12K	CIFAR10
GIN	0.163 ± 0.004	59.82 ± 0.33
$k = 1$	0.147 ± 0.006	71.37 ± 0.28
$k = 2$	0.120 ± 0.005	72.19 ± 0.13
$k = 3$	0.080 ± 0.001	OOM

TABLE 3.8: Effect of various encodings

Ablation of GIN-AK$^+$	ZINC-12K	CIFAR10
Full	0.080 ± 0.001	72.19 ± 0.13
w/o Subgraph encoding	0.086 ± 0.001	67.76 ± 0.29
w/o Centroid encoding	0.084 ± 0.003	72.20 ± 0.96
w/o Context encoding	0.088 ± 0.003	69.25 ± 0.30
w/o Distance-to-Centroid	0.085 ± 0.001	71.91 ± 0.22

Next Table 3.8 illustrates the added benefit of various node encodings. Compared to the full design, eliminating any of the subgraph, centroid, or context encodings (Eq.s (3.3)–(3.5)) yields notably inferior results. Encoding without distance awareness is also subpar. These justify the design choices in our framework and verify the practical benefits of our design.

As GNN-AK$^+$ is not directly explained by Subgraph-1-WL* (though D2C is supported by Subgraph-1-WL, by strengthening HASH), we conduct additional ablation studies over GNN-AK$^+$ with different combinations of removing context encoding and D2C, shown in Table 3.9. Note that all experiments in Table 3.9 are using a 1-layer base GIN model. We summarize the observations as follows.

- For real-world datasets (ZINC-12K, CIFAR10), We observe that the largest performance improvement over base model comes from wrapping base GNN with Eq.3.2 (the performance of GIN-AK), while adding context encoding and D2C monotonically improves performance of GNN-AK$^+$.
- For substructure counting and graph property regression tasks, we observe D2C significantly increases the performance of GIN-AK$^+$ w/o Ctx (supported by the theory of Subgraph-1-WL where the HASH is required to be injective). Specifically, the cost-free D2C feature enhances the expressiveness of the base 1-layer GIN model (similar to the benefit of distance encoding shown in [Li+20]), resulting in a more expressive GNN-AK$^+$, which lies between Subgraph-1-WL and Subgraph-1-WL*. We leave the exact theoretical analysis of D2C's expressiveness benefit in future work. Notice that GIN-AK improves the performance over the base model but not in a large margin, we next show this is due to the insufficiency of the number of layers of base GIN.

TABLE 3.9: Study GNN-AK without context encoding (Ctx) and without distance-to-centroid (D2C). Base model is **1-layer** GIN for all methods.

Method	ZINC-12K	CIFAR10	EXP	SR25	Counting Substructures (MAE)				Graph Properties ($\log_{10}$(MAE))		
	(MAE)	(ACC)	(ACC)	(ACC)	Triangle	Tailed Tri.	Star	4-Cycle	IsConnected	Diameter	Radius
GIN	0.163	59.82	50%	6.67%	0.3569	0.2373	0.0224	0.2185	-1.9239	-3.3079	-4.7584
GIN-AK	0.094	67.51	100%	6.67%	0.2311	0.1805	0.0207	0.1911	-1.9574	-3.6925	-5.0574
GIN-AK$^+$ w/o Ctx	0.088	69.25	100%	6.67%	0.0130	0.0108	0.0177	0.0131	-2.7083	-3.9257	-5.2784
GIN-AK$^+$ w/o D2C	0.085	71.91	100%	6.67%	0.1746	0.1449	0.0193	0.1467	-2.0521	-3.6980	-5.0984
GIN-AK$^+$	0.080	72.19	100%	6.67%	0.0123	0.0112	0.0150	0.0126	-2.7513	-3.9687	-5.1846

TABLE 3.10: Study the effect of base model's number of layers while keeping total number of layers in GNN-AK fixed. Different effect is observed for GNN-AK and GNN-AK without D2C.

Method	GIN-AK's	Base GIN's	Counting Substructures (MAE)				Graph Properties ($\log_{10}$(MAE))		
	#Layers	#Layers	Triangle	Tailed Tri.	Star	4-Cycle	IsConnected	Diameter	Radius
GIN	0	6	0.3569	0.2373	0.0224	0.2185	-1.9239	-3.3079	-4.7584
GIN-AK	6	1	0.2311	0.1805	0.0207	0.1911	-1.9574	-3.6925	-5.0574
	3	2	0.1556	0.1275	0.0172	0.1419	-1.9134	-3.7573	-5.0100
	2	3	0.1064	0.0819	0.0168	0.1071	-1.9259	-3.7243	-4.9257
	1	6	0.0934	0.0751	0.0216	0.0726	-1.9916	-3.6555	-4.9249
GIN-AK$^+$ w/o D2C	6	1	0.1746	0.1449	0.0193	0.1467	-2.0521	-3.6980	-5.0984
	3	2	0.1244	0.1052	0.0219	0.1121	-2.1538	-3.7305	-4.9250
	2	3	0.1021	0.0830	**0.0162**	0.0986	**-2.2268**	**-3.7585**	**-5.1044**
	1	6	**0.0885**	**0.0696**	0.0174	**0.0668**	-2.0541	-3.6834	-4.8428
GIN-AK$^+$ with D2C	6	1	0.0123	0.0112	0.0150	0.0126	**-2.7513**	**-3.9687**	**-5.1846**
	3	2	0.0116	0.0100	0.0168	0.0122	-2.6827	-3.8407	-5.1034
	2	3	0.0119	0.0102	0.0146	0.0127	-2.6197	-3.8745	-5.1177
	1	6	0.0131	0.0123	0.0174	0.0162	-2.5938	-3.7978	-5.0492

According to Prop. 1, the base model must be discriminative enough such that $\text{HASH}(G[\mathcal{N}_k(v_i^G)]) \neq \text{HASH}(H[\mathcal{N}_k(v_i^H)])$, for Subgraph-1-WL with k-egonet to enjoy expressiveness benefits. In addition to using D2C to increase the expressiveness of base model, another way is to practically increase the number of layers of the base model (akin to increasing the number of iterations of 1-WL, as in the definition of Subgraph-1-WL*). We study the effect of base model's number of layers in GNN-AK$^+$, with and without D2C in Table 3.10.

- Firstly, GNN-AK$^+$ with D2C is insensitive to the depth of base model, with 1-layer base model being enough to achieve great performance in counting substructures and the best performance in regressing graph properties. We hypothesize that D2C-enhanced 1-layer GIN base model is discriminative enough for subgraphs in the dataset, and without expressiveness bottleneck of base model, increasing GNN-AK$^+$'s depth benefits expressiveness, akin to increasing iterations of Subgraph-1-WL. Besides, unlike counting substructure that needs local information within subgraphs, regressing graph properties needs the graph's global information which can only be accessed with increasing GNN-AK$^+$'s (outer) depth.
- Secondly, GNN-AK($^+$) without D2C suffers from a trade-off between the base model's expressiveness (depth of base model) and the number of GNN-AK($^+$) layers (outer depth), which is clearly observed in regressing graph properties. We hypothesize that without D2C the 1-layer GIN base model is not discriminative enough for subgraphs in the dataset, and with this bottleneck of base model, GNN-AK($^+$) cannot benefit from increasing the outer depth. Hence the number of layers of the base model are important for the expressiveness of GNN-AK($^+$) when D2C is not used.

3.7 Conclusion

Our work introduces a new, general-purpose framework called GNN-As-Kernel (GNN-AK) to uplift the expressiveness of any GNN, with the key idea of employing a base GNN as a kernel on induced subgraphs of the input graph, generalizing from the star-pattern aggregation of classical MPNNs. Our approach provides an expressiveness and performance boost, while retaining practical scalability of MPNNs—a highly sought-after middle ground between the two regimes of scalable yet less-expressive MPNNs and high-expressive yet practically infeasible and poorly-generalizing k-order designs. We theoretically studied the expressiveness of GNN-AK, provided a concrete design and the more powerful GNN-AK$^+$, introducing SubgraphDrop for shrinking runtime and memory footprint. Extensive experiments on both simulated and real-world benchmark datasets empirically justified that GNN-AK($^+$) (i) uplifts base GNN expressiveness for multiple base GNN choices (e.g. over 1&2-WL for MPNNs, and over 3-WL for PPGN), (ii) which translates to performance gains with SOTA results on graph-level benchmarks, (iii) while retaining scalability to practical graphs.

Chapter 4

Using Unordered High Order Interactions

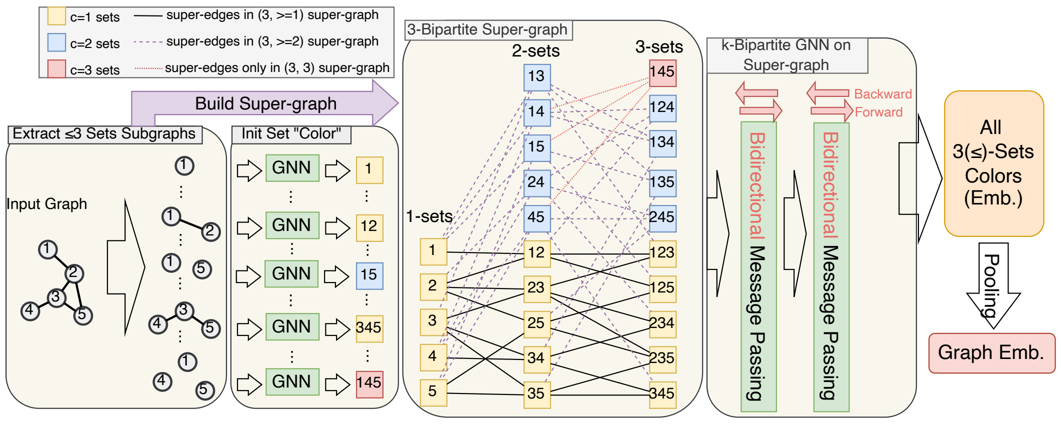
Chapter based on: Lingxiao Zhao, Louis Härtel, Neil Shah, and Leman Akoglu. "A Practical, Progressively-Expressive GNN". in: *36th Conference on Neural Information Processing Systems*. 2022. URL: https://openreview.net/forum?id=WBv9Z6qpA8x.

4.1 Introduction

In recent years, graph neural networks (GNNs) have gained considerable attention [Wu+20a; Wu+20b] for their ability to tackle various node-level [KW17; Vel+18], link-level [Zha+20; San+21] and graph-level [Bod+21a; Mar+19a] learning tasks, given their ability to learn rich representations for complex graph-structured data. The common template for designing GNNs follows the message passing paradigm; these so-called message-passing neural networks (MPNNs) are built by stacking layers which encompass feature transformation and aggregation operations over the input graph [Ma+20; Gil+17]. Despite their advantages, MPNNs have several limitations including oversmoothing [ZA20b; Che+20a; OS20], oversquashing [AY21; Top+21], inability to distinguish node identities [You+21] and positions [YYL19], and *expressive power* [Xu+19].

Since Xu et al.'s [Xu+19] seminal work showed that MPNNs are at most as powerful as the first-order Weisfeiler-Leman (1-WL) test in the graph isomorphism (GI) testing framework, there have been several follow-up works on improving the understanding of GNN expressiveness [Arv+20; Che+20b]. In response, the community proposed many GNN models to overcome such limitations [Sat20; AL21; Zha+22c]. Several of these aim to design powerful higher-order GNNs which are increasingly expressive [Mor+19; Mar+19c; KP19; Bod+21a] by inspiration from the k-WL hierarchy [She+11].

A key limitation of such higher-order models that reach beyond 1-WL is their poor scalability; in fact, these models can only be applied to small graphs with small k in practice [Mar+19a; Mar+19c; Mor+19] due to combinatorial growth in complexity. On the other hand lies an open question on whether practical graph learning tasks indeed need such complex, and extremely expressive GNN models. Historically, Babai et al. [BES80] showed that almost all graphs on n nodes can be distinguished by 1-WL. In other contexts like node classification, researchers have encountered superior generalization performance with graph-augmented multi-layer perceptron (GA-MLP) models [Ros+20; KBG19] compared to MPNNs, despite the former being strictly less expressive than the latter [CCB20]. Considering that increased model complexity has



c=1 sets
c=2 sets
c=3 sets
super-edges in (3, >=1) super-graph
super-edges in (3, >=2) super-graph
super-edges only in (3, 3) super-graph
Build Super-graph
Extract ≤3 Sets Subgraphs
Input Graph
Init Set "Color"
GNN
3-Bipartite Super-graph
1-sets
2-sets
3-sets
k-Bipartite GNN on Super-graph
Backward
Forward
Bidirectional Message Passing
All 3(≤)-Sets Colors (Emb.)
Pooling
Graph Emb.

Figure 4.1: Main steps of $(k, c)(\leq)$-SetGNN. Given a graph and (k, c), we build the (k, c)-bipartite super-graph (in middle) containing sets with up to k nodes and c connected components in the induced subgraph, on which a base GNN assigns initial "colors". Bidirectional bipartite GNN layers with frw.-bckw. message passing learn set embeddings, pooled into graph embedding. The size of super-graph, and accordingly its expressiveness, grows progressively with increasing k and c. The 2-bipartite message passing generalizes normal GNN, edge GNN, and line graph (see Appendix. A12 in [Zha+22b]).

negative implications in terms of overfitting and generalization [Haw04], it is worth re-evaluating continued efforts to pursue maximally expressive GNNs in practice. Ideally, one could study these models' generalization performance on various real-world tasks by increasing k (expressiveness) in the k-WL hierarchy. Yet, given the impracticality of this too-coarse "ruler" of expressiveness, which becomes infeasible beyond $k>3$, one desires a more fine-grained "ruler", that is, a new hierarchy whose expressiveness grows more gradually. Such a hierarchy could enable us to gradually build more expressive models which admit improved scaling, and avoid unnecessary leaps in model complexity for tasks which do not require them, guarding against overfitting.

Present Work. In this paper, we propose such a hierarchy, and an associated *practical progressively-expressive GNN model*, called $(k, c)(\leq)$-SetGNN, whose expressiveness can be modulated through k and c. In a nutshell, we take inspiration from k-WL, yet achieve practicality through three design ideas which simplify key bottlenecks of scaling k-WL: **First**, we move away from k-tuples of the original k-WL to k-multisets (unordered), and then to $k(\leq)$-sets. We demonstrate that these steps drastically reduce the number nodes in the k-WL graph while retaining significant expressiveness. (Sec.s 4.3.2–4.3.3) **Second**, by considering the underlying sparsity of an input graph, we reduce scope to $k, c(\leq)$-sets that consist of $\leq k$ nodes whose induced subgraph is comprised of $\leq c \leq k$ connected components. This also yields massive reduction in the number of nodes while improving practicality; i.e. small values of c on sparse graphs can allow one to increase k well beyond 3 in practice. (Sec. 4.3.4) **Third**, we design a super-graph architecture that consists of a sequence of $k-1$ *bipartite* graphs over which we learn embeddings for our $k, c(\leq)$-sets, using bidirectional message passing. These embeddings can be pooled to yield a final graph embedding.. (Sec.s 4.4.1–4.4.2) We also speed up initializing "colors" for the $k, c(\leq)$-sets, for $c > 1$, substantially reducing computation and memory. (Sec. 4.4.3) Fig. 4.1 overviews of our proposed framework. Experimentally, our $(k, c)(\leq)$-SetGNN outperforms existing state-of-the-art GNNs on simulated expressiveness as well as real-world graph-level tasks, achieving new bests on substructure counting and ZINC-12K, respectively. We show that generalization performance reflects increasing expressiveness by k and c. Our proposed scalable designs allow us to train models with e.g. k=10 or c=3 with practical running time and memory requirements.

4.2 Related Work

MPNN limitations. Given the understanding that MPNNs have expressiveness upper-bounded by 1-WL [Xu+19], several researchers investigated what else MPNNs *cannot* learn. To this end, Chen et al. [Che+20b] showed that MPNNs cannot count induced connected substructures of 3 or more nodes, while along with [Che+20b], Arvind et al. [Arv+20] showed that MPNNs can only count star-shaped patterns. Loukas [Lou20b] further proved several results regarding decision problems on graphs (e.g. subgraph verification, cycle detection), finding that MPNNs cannot solve these problems unless strict conditions of depth and width are met. Moreover, Garg et al. [GJJ20] showed that many standard MPNNs cannot compute properties such as longest or shortest cycles, diameters or existence of cliques.

Improving expressivity. Several works aim to improve expressiveness limitations of MPNNs. One approach is to inject features into the MPNN aggregation, motivated by Loukas [Lou20b] who showed that MPNNs can be universal approximators when nodes are sufficiently distinguishable. Sato et al. [SYK21] show that injecting random features can better enable MPNNs to solve problems like minimum dominating set and maximum matching. You et al. [You+21] inject cycle counts as node features, motivated by the limitation of MPNNs not being able to count cycles. Others proposed utilizing subgraph isomorphism counting to empower MPNNs with substructure information they cannot learn [Bou+20; Bod+21a]. Earlier work by You et al. [YYL19] adds positional features to distinguish node which naïve MPNN embeddings would not. Recently, several works also propose utilizing subgraph aggregation; Zhang et al. [ZL21], Zhao et al. [Zha+22c] and Bevilacqua et al. [Bev+22] propose subgraph variants of the WL test which are no less powerful than 3-WL. Recently a following up work [Fra+22] shows that rooted subgraph based extension with 1-WL as kernel is bounded by 3-WL. The community is yet exploring several distinct avenues in overcoming limitations: Murphy et al. [Mur+19] propose a relational pooling mechanism which sums over all permutations of a permutation-sensitive function to achieve above-1-WL expressiveness. Balcilar et al. [Bal+21] generalizes spatial and spectral MPNNs and shows that instances of spectral MPNNs are more powerful than 1-WL. Azizian et al. [AL21] and Geerts et al. [GR22] unify expressivity and approximation ability results for existing GNNs.

***k*-WL-inspired GNNs.** The k-WL test captures higher-order interactions in graph data by considering all k-tuples, i.e., size k ordered multisets defined over the set of nodes. While highly expressive, it does not scale to practical graphs beyond a very small k ($k = 3$ pushes the practical limit even for small graphs). However, several works propose designs which can achieve k-WL in theory and strive to make them practical. Maron et al. [Mar+19c] proposes a general class of invariant graph networks k-IGNs having exact k-WL expressivity [Gee20], while being not scalable. Morris et al. have several work on k-WL and its variants like k-LWL [MKM17], k-GNN [Mor+19], and δ-k-WL-GNN [MRM20]. Both k-LWL and k-GNN use a variant of k-WL that only considers k-sets and are strictly weaker than k-WL but much more practical. In our paper we claim that $k(\leq)$-sets should be used with additional designs to keep the best expressivity while remaining practical. The δ-k-WL-GNN works with k-tuples but sparsifies the connections among tuples by considering locality. Qian et al. [Qia+22] extends the k-tuple to subgraph network but has same scalability bottleneck as k-WL. A recent concurrent work SpeqNet [Mor+22] proposes to reduce number of tuples by restricting the number of connected components of tuples' induced subgraphs. The idea is independently explored in our paper in Sec.4.3.4. All four variants proposed by Morris et al. do not have realizations beyond $k > 3$ in their experiments while we manage to reach $k = 10$. Interestingly, a recent work [Kim+22] links graph transformer with k-IGN that having better (or equal) expressivity, however is still not scalable.

4.3 A practical progressively-expressive isomorphism test: $(k, c)(\leq)$-SetWL

We first motivate our method $(k, c)(\leq)$-SetGNN from the GI testing perspective by introducing $(k, c)(\leq)$-SetWL. We first introduce notation and background of k-WL (Sec. 4.3.1). Next, we show how to increase the practicality of k-WL without reducing much of its expressivity, by removing node ordering (Sec. 4.3.2), node repetitions (Sec. 4.3.3), and leveraging graph sparsity (Sec. 4.3.4). We then give the complexity analysis (Sec. 4.3.5). We close the section with extending the idea to k-FWL which is as expressive as $(k+1)$-WL (Sec. 4.3.6). All proofs can be found in Appendix of [Zha+22b].

Notation: Let $G = (V(G), E(G), l_G)$ be an undirected, colored graph with nodes $V(G)$, edges $E(G)$, and a color-labeling function $l_G : V(G) \to C$ where $C = \{c_1, ..., c_d\}$ denotes a set of d distinct colors. Let $[n] = \{1, 2, 3, ..., n\}$. Let $(\cdot)$ denote a *tuple* (ordered multiset), $\{\{\cdot\}\}$ denote a *multiset* (set which allows repetition), and $\{\cdot\}$ denote a *set*. We define $\overrightarrow{\boldsymbol{v}} = (v_1, ..., v_k)$ as a k-tuple, $\tilde{\boldsymbol{v}} = \{\{v_1, ..., v_k\}\}$ as a k-multiset, and $\hat{\boldsymbol{v}} = \{v_1, ..., v_k\}$ as a k-set. Let $\overrightarrow{\boldsymbol{v}}[x/i] = (v_1, ..., v_{i-1}, x, v_{i+1}, ..., v_k)$ denote replacing the i-th element in $\overrightarrow{\boldsymbol{v}}$ with x, and analogously for $\tilde{\boldsymbol{v}}[x/i]$ and $\hat{\boldsymbol{v}}[x/i]$ (assume mutlisets and sets are represented with the canonical ordering). When $v_i \in V(G)$, let $G[\overrightarrow{\boldsymbol{v}}]$, $G[\tilde{\boldsymbol{v}}]$, $G[\hat{\boldsymbol{v}}]$ denote the induced subgraph on G with nodes inside $\overrightarrow{\boldsymbol{v}}, \tilde{\boldsymbol{v}}, \hat{\boldsymbol{v}}$ respectively (keeping repetitions). An isomorphism between two graphs G and G' is a bijective mapping $p : V(G) \to V(G')$ which satisfies $\forall u, v \in V(G), (u, v) \in E(G) \iff (p(u), p(v)) \in E(G')$ and $\forall u \in V(G), l_G(u) = l_{G'}(p(u))$. Two graphs G, G' are isomorphic if there exists an isomorphism between them, which we denote as $G \cong G'$.

4.3.1 Preliminaries: the k-Weisfeiler-Leman (k-WL) Graph Isomorphism Test

The 1-dimensional Weisfeiler-Leman (1-WL) test, also known as color refinement [RC77] algorithm, is a widely celebrated approach to (approximately) test GI. Although extremely fast and effective for most graphs (1-WL can provide canonical forms for all but $n^{-1/7}$ fraction of n-vertex graphs [BES80]), it fails to distinguish members of many important graph families, such as regular graphs. The more powerful k-WL algorithm first appeared in [Wei76], and extends coloring of vertices (or 1-tuples) to that of k-tuples. k-WL is progressively expressive with increasing k, and can distinguish any finite set of graphs given a sufficiently large k. k-WL has many interesting connections to logic, games, and linear equations [CFI92; GO15]. Another variant of k-WL is called k-FWL (Folklore WL), such that $(k + 1)$-WL is equally expressive as k-FWL [CFI92; Gro21]. Our work focuses on k-WL.

k-WL iteratively recolors all n^k k-tuples defined on a graph G with n nodes. At iteration 0, each k-tuple $\overrightarrow{\boldsymbol{v}} = (v_1, ..., v_k) \in V(G)^k$ is initialized with a color as its *atomic type* $\boldsymbol{at_k}(G, \overrightarrow{\boldsymbol{v}})$. Assume G has d colors, then $\boldsymbol{at_k}(G, \overrightarrow{\boldsymbol{v}}) \in \{0, 1\}^{2\binom{k}{2}+kd}$ is an ordered vector encoding. The first $\binom{k}{2}$ entries indicate whether $v_i = v_j$, $\forall i, j, 1 \leq i < j \leq k$, which is the node repetition information. The second $\binom{k}{2}$ entries indicate whether $(v_i, v_j) \in E(G)$. The last kd entries one-hot encode the initial color of v_i, $\forall i, 1 \leq i \leq k$. Importantly, $\boldsymbol{at_k}(G, \overrightarrow{\boldsymbol{v}}) = \boldsymbol{at_k}(G', \overrightarrow{\boldsymbol{v'}})$ if and only if $v_i \mapsto v'_i$ is an isomorphism from $G[\overrightarrow{\boldsymbol{v}}]$ to $G'[\overrightarrow{\boldsymbol{v'}}]$. Let $\boldsymbol{wl}_k^{(t)}(G, \overrightarrow{\boldsymbol{v}})$ denote the color of k-tuple $\overrightarrow{\boldsymbol{v}}$ on graph G at t-th iteration of k-WL, where colors are initialized with $\boldsymbol{wl}_k^{(0)}(G, \overrightarrow{\boldsymbol{v}}) = \boldsymbol{at_k}(G, \overrightarrow{\boldsymbol{v}})$.

At the t-th iteration, k-WL updates the color of each $\overrightarrow{\boldsymbol{v}} \in V(G)^k$ according to

$$\boldsymbol{wl}_k^{(t+1)}(G,\overrightarrow{\boldsymbol{v}}) = \text{HASH}\Big(\boldsymbol{wl}_k^{(t)}(G,\overrightarrow{\boldsymbol{v}}), \Big\{\!\!\Big\{\boldsymbol{wl}_k^{(t)}(G,\overrightarrow{\boldsymbol{v}}[x/1])\Big| x\in V(G)\Big\}\!\!\Big\}, ..., \Big\{\!\!\Big\{\boldsymbol{wl}_k^{(t)}(G,\overrightarrow{\boldsymbol{v}}[x/k])\Big| x\in V(G)\Big\}\!\!\Big\}\Big) \tag{4.1}$$

Let $\boldsymbol{gwl}_k^{(t)}(G)$ denote the encoding of G at t-th iteration of k-WL. Then,

$$\boldsymbol{gwl}_k^{(t)}(G) = \text{HASH}\Big(\Big\{\!\!\Big\{\boldsymbol{wl}_k^{(t)}(G, \overrightarrow{\boldsymbol{v}})\Big| \overrightarrow{\boldsymbol{v}} \in V(G)^k\Big\}\!\!\Big\}\Big) \tag{4.2}$$

Two k-tuples $\overrightarrow{\boldsymbol{v}}$ and $\overrightarrow{\boldsymbol{u}}$ are connected by an edge if $|\{i \in [k]|v_i = u_i\}| = k - 1$, or informally if they share $(k-1)$ entries. Then, k-WL defines a super-graph $S_{k\text{-wl}}(G)$ with its nodes being all k-tuples in G, and edges defined as above. Eq. (4.1) defines the rule of color refinement on $S_{k\text{-wl}}(G)$. Intuitively, k-WL is akin to (but more powerful as it orders subgroups of neighbors) running 1-WL algorithm on the supergraph $S_{k\text{-wl}}(G)$. As $t \to \infty$, the color $\boldsymbol{wl}_k^{(t+1)}(G, \overrightarrow{\boldsymbol{v}})$ converges to a stable value, denoted as $\boldsymbol{wl}_k^{(\infty)}(G, \overrightarrow{\boldsymbol{v}})$ and the corresponding stable graph color denoted as $\boldsymbol{gwl}_k^{(\infty)}(G)$. For two non-isomorphic graphs G, G', k-WL can successfully distinguish them if $\boldsymbol{gwl}_k^{(\infty)}(G) \neq \boldsymbol{gwl}_k^{(\infty)}(G')$. The expressivity of $\boldsymbol{wl}_k^{(t)}$ can be exactly characterized by first-order logic with counting quantifiers. Let $\mathbf{C}_k^t$ denote all first-order formulas with at most k variables and t-depth counting quantifiers, then $\boldsymbol{wl}_k^{(t)}(G, \overrightarrow{\boldsymbol{v}}) = \boldsymbol{wl}_k^{(t)}(G', \overrightarrow{\boldsymbol{v'}}) \iff \forall \phi \in \mathbf{C}_k^t, \phi(G, \overrightarrow{\boldsymbol{v}}) = \phi(G', \overrightarrow{\boldsymbol{v'}})$. Additionally, there is a t-step bijective pebble game that are equivalent to t-iteration k-WL in expressivity. See Appendix in [Zha+22b] for the pebble game characterization of k-WL.

Despite its power, k-WL uses all n^k tuples and has $O(kn^k)$ complexity at each iteration.

4.3.2 From k-WL to k-MultisetWL: Removing Ordering

Our first proposed adaptation to k-WL is to *remove ordering* in each k-tuple, i.e. changing k-tuples to k-multisets. This greatly reduces the number of supernodes to consider by $O(k!)$ times.

Let $\tilde{\boldsymbol{v}} = \{\!\{v_1, ..., v_k\}\!\}$ be the corresponding multiset of tuple $\overrightarrow{\boldsymbol{v}} = (v_1, ..., v_k)$. We introduce a canonical order function on G, $o_G : V(G) \to [n]$, and a corresponding indexing function $o_G^{-1}(\tilde{\boldsymbol{v}}, i)$, which returns the i-th element of $v_1, ..., v_k$ sorted according to o_G. Let $\boldsymbol{mwl}_k^{(t)}(G, \tilde{\boldsymbol{v}})$ denote the color of the k-multiset $\tilde{\boldsymbol{v}}$ at t-th iteration of k-MultisetWL, formally defined next.

At $t = 0$, we initialize the color of $\tilde{\boldsymbol{v}}$ as $\boldsymbol{mwl}_k^{(0)}(G, \tilde{\boldsymbol{v}}) = \text{HASH}\big(\{\!\{\boldsymbol{at}_k(G, p(\tilde{\boldsymbol{v}}))|p \in \text{perm[k]}\}\!\}\big)$ where perm[k][1] denotes the set of all permutation mappings of k elements. It can be shown that $\boldsymbol{mwl}_k^{(0)}(G, \tilde{\boldsymbol{v}}) = \boldsymbol{mwl}_k^{(0)}(G', \tilde{\boldsymbol{v}}')$ if and only if $G[\tilde{\boldsymbol{v}}]$ and $G'[\tilde{\boldsymbol{v}}']$ are isomorphic.

[1]This function should also consider repeated elements; we omit this for clarity of presentation.

At t-th iteration, k-MultisetWL updates the color of every k-multiset by

$$\boldsymbol{mwl}_k^{(t+1)}(G,\tilde{\boldsymbol{v}}) = \text{HASH}\Big(\boldsymbol{mwl}_k^{(t)}(G,\tilde{\boldsymbol{v}}), \Big\{\!\!\Big\{ \{\!\{\boldsymbol{mwl}_k^{(t)}(G,\tilde{\boldsymbol{v}}[x/o_G^{-1}(\tilde{\boldsymbol{v}},1)])\big| x \in V(G)\}\!\}, \quad ..., \{\!\{\boldsymbol{mwl}_k^{(t)}(G,\tilde{\boldsymbol{v}}[x/o_G^{-1}(\tilde{\boldsymbol{v}},k)])\big| x \in V(G)\}\!\} \Big\}\!\!\Big\}\Big) \tag{4.3}$$

Where $\tilde{\boldsymbol{v}}[x/o_G^{-1}(\tilde{\boldsymbol{v}},i)])$ denotes replacing the i-th (ordered by o_G) element of the multiset with $x \in V(G)$. Let $S_{k\text{-mwl}}(G)$ denote the super-graph defined by k-MultisetWL. Similar to Eq. (4.2), the graph level encoding is $\boldsymbol{gmwl}_k^{(t)}(G) = \text{HASH}(\{\!\{\boldsymbol{mwl}_k^{(t)}(G,\tilde{\boldsymbol{v}})\big|\forall \tilde{\boldsymbol{v}} \in V(S_{k\text{-mwl}}(G))\}\!\})$.

Interestingly, although k-MultisetWL has significantly fewer number of node groups than k-WL, we show it is no less powerful than k-1-WL in terms of distinguishing graphs, while being upper bounded by k-WL in distingushing both node groups and graphs.

Theorem 4.3.1. *Let $k \geq 1$ and $\boldsymbol{wl}_k^{(t)}(G,\tilde{\boldsymbol{v}}) := \{\!\{\boldsymbol{wl}_k^{(t)}(G,p(\tilde{\boldsymbol{v}}))|p \in perm[k]\}\!\}$. For all $t \in \mathbb{N}$ and all graphs G, G': k-MultisetWL is upper bounded by k-WL in distinguishing multisets $G, \tilde{\boldsymbol{v}}$ and $G', \tilde{\boldsymbol{v}}'$ at t-th iteration, i.e. $\boldsymbol{wl}_k^{(t)}(G,\tilde{\boldsymbol{v}}) = \boldsymbol{wl}_k^{(t)}(G',\tilde{\boldsymbol{v}}') \implies \boldsymbol{mwl}_k^{(t)}(G,\tilde{\boldsymbol{v}}) = \boldsymbol{mwl}_k^{(t)}(G',\tilde{\boldsymbol{v}}')$.*

Theorem 4.3.2. *k-MultisetWL is no less powerful than (k-1)-WL in distinguishing graphs: for any $k \geq 3$ there exists graphs that can be distinguished by k-MultisetWL but not by (k-1)-WL.*

Theorem 4.3.2 is proved by using a variant of a series of CFI [CFI92] graphs which cannot be distinguished by k-WL. This theorem shows that k-MultisetWL is indeed very powerful and finding counter examples of k-WL distinguishable graphs that cannot be distinguished by k-MultisetWL is very hard. Hence we conjecture that k-MultisetWL may have the same expressivity as k-WL in distinguishing undirected graphs. Additionally, the theorem also implies that k-MultisetWL is strictly more powerful than $(k-1)$-MultisetWL.

We next give a pebble game characterization of k-MultisetWL, which is named doubly bijective k-pebble game presented in Appendix of [Zha+22b]. The game is used in the proof of Theorem 4.3.2.

Theorem 4.3.3. *k-MultisetWL has the same expressivity as the doubly bijective k-pebble game.*

4.3.3 From k-MultisetWL to $k(\leq)$-SetWL: Removing Repetition

Next, we propose further *removing repetition* inside any k-multiset, i.e. transforming k-multisets to $k(\leq)$-sets. We assume elements of k-multiset $\tilde{\boldsymbol{v}}$ and $k(\leq)$-set $\hat{\boldsymbol{v}}$ in G are sorted based on the ordering function o_G, and omit o_G for clarity. Let $s(\cdot)$ transform a multiset to set by removing repeats, and let $r(\cdot)$ return a tuple with the number of repeats for each distinct element in a multiset. Specifically, let $\hat{\boldsymbol{v}} = s(\tilde{\boldsymbol{v}})$ and $\hat{\boldsymbol{n}} = r(\tilde{\boldsymbol{v}})$, then $m := |\hat{\boldsymbol{v}}| = |\hat{\boldsymbol{n}}| \in [k]$ denotes the number of distinct elements in k-multiset $\tilde{\boldsymbol{v}}$, and $\forall i \in [m]$, $\hat{\boldsymbol{v}}_i$ is the i-th distinct element with $\hat{\boldsymbol{n}}_i$ repetitions. Clearly there is an injective mapping between $\tilde{\boldsymbol{v}}$ and $(\hat{\boldsymbol{v}}, \hat{\boldsymbol{n}})$; let f be the inverse mapping such that $\tilde{\boldsymbol{v}} = f(s(\tilde{\boldsymbol{v}}), r(\tilde{\boldsymbol{v}}))$. Equivalently, each m-set $\hat{\boldsymbol{v}}$ can be mapped with a multiset of k-multisets: $\hat{\boldsymbol{v}} \leftrightarrow \{\!\{\tilde{\boldsymbol{v}} = f(\hat{\boldsymbol{v}}, \hat{\boldsymbol{n}}) \mid \sum_{i=1}^m \hat{\boldsymbol{n}}_i = k, \forall i\ \hat{\boldsymbol{n}}_i \geq 1\}\!\}$. Based on this relationship, we extend the connection among k-multisets to $k(\leq)$-sets: given $m_1, m_2 \in [k]$, a m_1-set $\hat{\boldsymbol{v}}$ is connected to a m_2-set $\hat{\boldsymbol{u}}$ if and only if $\exists \hat{\boldsymbol{n}}_v, \hat{\boldsymbol{n}}_u$, $f(\hat{\boldsymbol{v}}, \hat{\boldsymbol{n}}_v)$ is connected with $f(\hat{\boldsymbol{u}}, \hat{\boldsymbol{n}}_u)$ in k-MULTISETWL. Let $S_{k\text{-swl}}(G)$ denote the defined super-graph on G by $k(\leq)$-SETWL. It can be shown that this is equivalent to either (1) $(|m_1 - m_2| = 1) \wedge (|\hat{\boldsymbol{v}} \cap \hat{\boldsymbol{u}}| = \min(m_1, m_2))$ or (2) $(m_1 = m_2) \wedge (|\hat{\boldsymbol{v}} \cap \hat{\boldsymbol{u}}| = m_1 - 1)$ is true. Notice that $S_{k\text{-swl}}(G)$ contains a sequence of $k-1$ bipartite graphs with each reflecting the connections among the $(m-1)$-sets and the m-sets. It also contains $k-1$ subgraphs, i.e. the connections among the m-sets for $m = 2, ..., k$. Later on we will show that these $k-1$ subgraphs can be ignored without affecting $k(\leq)$-SETWL.

Let $\boldsymbol{swl}_k^{(t)}(G, \hat{\boldsymbol{v}})$ denote the color of m-set $\hat{\boldsymbol{v}}$ at t-th iteration of $k(\leq)$-SETWL. Now we formally define $k(\leq)$-SETWL. At $t = 0$, we initialize the color of a m-set $\hat{\boldsymbol{v}}$ ($m \in [k]$) as:

$$\boldsymbol{swl}_k^{(0)}(G, \hat{\boldsymbol{v}}) = \text{HASH}\big(\{\!\{\boldsymbol{mwl}_k^{(0)}(G, f(\hat{\boldsymbol{v}}, \hat{\boldsymbol{n}})) \;\big|\; \hat{\boldsymbol{n}}_1 + ... + \hat{\boldsymbol{n}}_m = k, \forall i\ \hat{\boldsymbol{n}}_i \geq 1\}\!\}\big) \tag{4.4}$$

Clearly $\boldsymbol{swl}_k^{(0)}(G, \hat{\boldsymbol{v}}) = \boldsymbol{swl}_k^{(0)}(G', \hat{\boldsymbol{v}}')$ if and only if $G[\hat{\boldsymbol{v}}]$ and $G'[\hat{\boldsymbol{v}}']$ are isomorphic. At t-th iteration, $k(\leq)$-SETWL updates the color of every m-set $\hat{\boldsymbol{v}}$ by

$$\begin{aligned}\boldsymbol{swl}_k^{(t+1)}(G, \hat{\boldsymbol{v}}) = \text{HASH}\Big(&\boldsymbol{swl}_k^{(t)}(G, \hat{\boldsymbol{v}}), \{\!\{\boldsymbol{swl}_k^{(t)}(G, \hat{\boldsymbol{v}} \cup \{x\}) \mid x \in V(G) \setminus \hat{\boldsymbol{v}}\}\!\}, \{\!\{\boldsymbol{swl}_k^{(t)}(G, \hat{\boldsymbol{v}} \setminus x) \mid x \in \hat{\boldsymbol{v}}\}\!\},\\ &\Big\{\!\!\Big\{ \{\!\{\boldsymbol{swl}_k^{(t)}(G, \hat{\boldsymbol{v}}[x/o_G^{-1}(\hat{\boldsymbol{v}}, 1)]) \mid x \in V(G) \setminus \hat{\boldsymbol{v}}\}\!\}, ..., \{\!\{\boldsymbol{swl}_k^{(t)}(G, \hat{\boldsymbol{v}}[x/o_G^{-1}(\hat{\boldsymbol{v}}, m)]) \mid x \in V(G) \setminus \hat{\boldsymbol{v}}\}\!\} \Big\}\!\!\Big\}\Big)\end{aligned} \tag{4.5}$$

Notice that when $m = 1$ and $m = k$, the third and second part of the hashing input is an empty multiset, respectively. Similar to Eq. (4.2), we formulate the graph level encoding as $\boldsymbol{gswl}_k^{(t)}(G) = \text{HASH}(\{\!\{\boldsymbol{swl}_k^{(t)}(G, \hat{\boldsymbol{v}}) \mid \forall \hat{\boldsymbol{v}} \in V(S_{k\text{-swl}}(G))\}\!\})$.

To characterize the relationship of their expressivity, we first extend k-MULTISETWL on sets by defining the color of a m-set $\hat{\boldsymbol{v}}$ on k-MULTISETWL as $\boldsymbol{mwl}_k^{(t)}(G, \hat{\boldsymbol{v}}) := \{\!\{\boldsymbol{mwl}_k^{(t)}(G, f(\hat{\boldsymbol{v}}, \hat{\boldsymbol{n}})) \mid \sum_{i=1}^m \hat{\boldsymbol{v}}_i = k, \forall i\ \hat{\boldsymbol{n}}_i \geq 1\}\!\}$. We prove that k-MULTISETWL is at least as expressive as $k(\leq)$-SETWL in terms of separating node sets and graphs.

Theorem 4.3.4. *Let $k \geq 1$, then $\forall t \in \mathbb{N}$ and all graphs G, G': $\boldsymbol{mwl}_k^{(t)}(G, \hat{\boldsymbol{v}}) = \boldsymbol{mwl}_k^{(t)}(G', \hat{\boldsymbol{v}}') \Longrightarrow \boldsymbol{swl}_k^{(t)}(G, \hat{\boldsymbol{v}}) = \boldsymbol{swl}_k^{(t)}(G', \hat{\boldsymbol{v}}')$.*

We also conjecture that k-MULTISETWL and $k(\leq)$-SETWL could be equally expressive, and leave it to future work. As a single m-set corresponds to $\binom{k-1}{m-1}$ k-multisets, moving from k-MULTISETWL to $k(\leq)$-SETWL further reduces the computational cost greatly.

4.3.4 From $k(\leq)$-SETWL to $(k, c)(\leq)$-SETWL: Accounting for Sparsity

Notice that for two arbitrary graphs G and G' with equal number of nodes, the number of $k(\leq)$-sets and the connections among all $k(\leq)$-sets in $k(\leq)$-SETWL are exactly the same, regardless of whether they are dense or sparse. We next propose to *account for the sparsity* of a graph G to further reduce the complexity of $k(\leq)$-SETWL. As the graph structure is encoded inside every m-set, when the graph becomes sparser, there would be more sparse m-sets with a potentially large number of disconnected components. Based on the hypothesis that the induced subgraph over a set (of nodes) with fewer disconnected components naturally contains more structural information, we propose to restrict the $k(\leq)$-sets to be $(k, c)(\leq)$-sets: all sets with at most k nodes and at most c connected components in its induced subgraph. Let $S_{k,c\text{-swl}}(G)$ denote the super-graph defined by $(k, c)(\leq)$-SETWL, then $S_{k,c\text{-swl}}(G) = S_{k\text{-swl}}(G)[\{\hat{\boldsymbol{v}} | \#\text{components}(G[\hat{\boldsymbol{v}}]) \leq c\}]$, which is the induced subgraph on the super-graph defined by $k(\leq)$-SETWL. Fortunately, $S_{k,c\text{-swl}}(G)$ can be efficiently and recursively constructed based on $S_{(k-1,c)\text{-swl}}(G)$, and the construction algorithm is inside the Appendix of [Zha+22b]. $(k,c)(\leq)$-SetWL can be defined similarly to $k(\leq)$-SetWL (Eq. (4.4) and Eq. (4.5)), however while removing all colors of sets that do not exist on $S_{k,c\text{-swl}}(G)$.

$(k,c)(\leq)$-SetWL is progressively expressive with increasing k and c, and when $c = k$, $(k,c)(\leq)$-SetWL becomes the same as $k(\leq)$-SetWL, as all $k(\leq)$-sets are then considered. Let $\boldsymbol{swl}_{k,c}^{(t)}(G,\hat{\boldsymbol{v}})$ denote the color of a $(k,c)(\leq)$-set $\hat{\boldsymbol{v}}$ on t-th iteration of $(k,c)(\leq)$-SetWL, then $\boldsymbol{gswl}_{k,c}^{(t)}(G) = \text{HASH}(\{\!\{\boldsymbol{swl}_{k,c}^{(t)}(G,\hat{\boldsymbol{v}}) \big| \forall \hat{\boldsymbol{v}} \in S_{k,c\text{-swl}}(G)\}\!\})$.

Theorem 4.3.5. *Let $k \geq 1$, then $\forall t \in \mathbb{N}$ and all graphs G, G':*

(1) when $1 \leq c_1 < c_2 \leq k$, if G, G' cannot be distinguished by $(k,c_2)(\leq)$-SetWL, they cannot be distinguished by $(k,c_1)(\leq)$-SetWL

(2) when $k_1 < k_2$, $\forall c \leq k_1$, if G, G' cannot be distinguished by $(k_2,c)(\leq)$-SetWL, they cannot be distinguished by $(k_1,c)(\leq)$-SetWL

4.3.5 Complexity Analysis

All color refinement algorithms described above run on a super-graph; thus, their complexity at each iteration is linear to the number of supernodes and number of edges of the super-graph. Instead of using big$\mathcal{O}$ notation that ignores constant factors, we compare the exact number of supernodes and edges. Let G be the input graph with n nodes and average degree d. For k-WL, there are n^k supernodes and each has $n*k$ number of neighbors, hence $S_{k\text{-wl}}(G)$ has n^k supernodes and $kn^{k+1}/2$ edges. For $m \in [k]$, there are $\binom{n}{m}$ m-sets and each connects to m number of $(m-1)$-sets. So $S_{k\text{-swl}}$ has $\sum_{i=1}^{k}\binom{n}{i} \leq \binom{n}{k}\frac{n-k+1}{n-2k+1}$ supernodes and $\sum_{i=2}^{k} i\binom{n}{i} = n\sum_{i=1}^{k-1}\binom{n-1}{i} \leq n\binom{n-1}{k-1}\frac{n-k+1}{n-2k+2}$ edges (derivation in Appendix). Here we ignore edges within m-sets for any $m \in [k]$ as they can be reconstructed from the bipartite connections among $(m-1)$-sets and m-sets, described in detail in Sec. 4.4.1. Consider e.g., $n = 30, k = 5$; we get $\frac{|V(S_{k\text{-wl}}(G))|}{|V(S_{k\text{-swl}}(G))|} = 139$, $\frac{|E(S_{k\text{-wl}}(G))|}{|E(S_{k\text{-swl}}(G))|} = 2182$. Directly analyzing the savings by restricting number of components is not possible without assuming a graph family. In Sec. 4.5.3 we measured the scalability for $(k,c)(\leq)$-SetGNN with different number of components directly on sparse graphs, where $(k,c)(\leq)$-SetWL shares similar scalability.

4.3.6 Set version of k-FWL

k-FWL is a stronger GI algorithm and it has the same expressivity as k+1-WL [Gro21], in this section we also demonstrate how to extend the set to k-FWL to get $k(\leq)$-SetFWL. Let $\boldsymbol{fwl}_k^{(t)}(G,\overrightarrow{\boldsymbol{v}})$ denote the color of k-tuple $\overrightarrow{\boldsymbol{v}}$ at t-th iteration of k-FWL. Then k-FWL is initialized the same as the k-WL, i.e. $\boldsymbol{fwl}_k^{(0)}(G,\overrightarrow{\boldsymbol{v}}) = \boldsymbol{wl}_k^{(0)}(G,\overrightarrow{\boldsymbol{v}})$. At t-th iteration, k-FWL updates colors with

$$\boldsymbol{fwl}_k^{(t+1)}(G,\overrightarrow{\boldsymbol{v}})=\text{HASH}\Big(\boldsymbol{fwl}_k^{(t)}(G,\overrightarrow{\boldsymbol{v}}),\Big\{\!\!\Big\{\Big(\boldsymbol{fwl}_k^{(t)}(G,\overrightarrow{\boldsymbol{v}}[x/1]),...,\boldsymbol{fwl}_k^{(t)}(G,\overrightarrow{\boldsymbol{v}}[x/k])\Big)\Big|x\in V(G)\Big\}\!\!\Big\}\Big) \tag{4.6}$$

Let $\boldsymbol{sfwl}_k^{(t)}(G,\hat{\boldsymbol{v}})$ denote the color of m-set $\hat{\boldsymbol{v}}$ at t-th iteration of $k(\leq)$-SetFWL. Then at t-th iteration it updates with

$$\boldsymbol{sfwl}_k^{(t+1)}(G,\hat{\boldsymbol{v}})=\text{HASH}\Big(\boldsymbol{sfwl}_k^{(t)}(G,\hat{\boldsymbol{v}}),\Big\{\!\!\Big\{\{\!\{\boldsymbol{sfwl}_k^{(t)}(G,\hat{\boldsymbol{v}}[x/1]),...,\boldsymbol{sfwl}_k^{(t)}(G,\hat{\boldsymbol{v}}[x/m])\}\!\}\Big|x\in V(G)\Big\}\!\!\Big\}\Big) \tag{4.7}$$

The $k(\leq)$-SetFWL should have better expressivity than $k(\leq)$-SetWL. We show in the next section that $k(\leq)$-SetWL can be further improved with less computation through an intermediate step while this is nontrivial for $k(\leq)$-SetFWL. We leave it to future work of studying $k(\leq)$-SetFWL.

4.4 A practical progressively-expressive GNN: $(k, c)(\leq)$-SETGNN

In this section we transform $(k, c)(\leq)$-SETWL to a GNN model by replacing the HASH function in Eq. (4.5) with a combination of MLP and DeepSet [Zah+17], given they are universal function approximators for vectors and sets, respectively [HSW89; Zah+17]. After the transformation, we propose two additional improvements (Sec. 4.4.2 and Sec. 4.4.3) to further improve scalability. We work on vector-attributed graphs. Let $G = (V(G), E(G), X)$ be an undirected graph with node features $\mathbf{x}_i \in \mathbb{R}^d, \forall i \in V(G)$. Proofs of all theorems in this section can be found in Appendix of [Zha+22b].

4.4.1 From $(k, c)(\leq)$-SetWL to $(k, c)(\leq)$-SetGNN

$(k, c)(\leq)$-SETWL defines a super-graph $S_{k,c\text{-swl}}$, which aligns with Eq. (4.5). We first rewrite Eq. (4.5) to reflect its connection to $S_{k,c\text{-swl}}$. For a supernode $\hat{\boldsymbol{v}}$ in $S_{k,c\text{-swl}}$, let $\mathcal{N}^G_{\text{left}}(\hat{\boldsymbol{v}}) = \{\hat{\boldsymbol{u}} \mid \hat{\boldsymbol{u}} \in S_{k,c\text{-swl}}, \hat{\boldsymbol{u}} \leftrightarrow \hat{\boldsymbol{v}} \text{ and } |\hat{\boldsymbol{u}}| = |\hat{\boldsymbol{v}}| - 1\}$, and $\mathcal{N}^G_{\text{right}}(\hat{\boldsymbol{v}}) = \{\hat{\boldsymbol{u}} \mid \hat{\boldsymbol{u}} \in S_{k,c\text{-swl}}, \hat{\boldsymbol{u}} \leftrightarrow \hat{\boldsymbol{v}} \text{ and } |\hat{\boldsymbol{u}}| = |\hat{\boldsymbol{v}}| + 1\}$. Then we can rewrite Eq. (4.5) for $(k, c)(\leq)$-SETWL as

$$\boldsymbol{swl}^{(t+1)}_{k,c}(G, \hat{\boldsymbol{v}}) = \Big(\boldsymbol{swl}^{(t)}_{k,c}(G, \hat{\boldsymbol{v}}), \boldsymbol{swl}^{(t+\frac{1}{2})}_{k,c}(G, \hat{\boldsymbol{v}}), \{\!\{\boldsymbol{swl}^{(t)}_{k,c}(G, \hat{\boldsymbol{u}}) \mid \hat{\boldsymbol{u}} \in \mathcal{N}^G_{\text{left}}(\hat{\boldsymbol{v}})\}\!\}, \{\!\{\boldsymbol{swl}^{(t+\frac{1}{2})}_{k,c}(G, \hat{\boldsymbol{u}}) \mid \hat{\boldsymbol{u}} \in \mathcal{N}^G_{\text{left}}(\hat{\boldsymbol{v}})\}\!\}\Big) \tag{4.8}$$

where $\boldsymbol{swl}^{(t+\frac{1}{2})}_{k,c}(G, \hat{\boldsymbol{v}}) := \{\!\{\boldsymbol{swl}^{(t)}_{k,c}(G, \hat{\boldsymbol{u}}) \mid \hat{\boldsymbol{u}} \in \mathcal{N}^G_{\text{right}}(\hat{\boldsymbol{v}})\}\!\}$. Notice that we omit HASH and apply it implicitly. Eq. (4.8) essentially splits the computation of Eq. (4.5) into two steps and avoids repeated computation via caching the explicit $t+\frac{1}{2}$ step. It also implies that the connection among m-sets for any $m \in [k]$ can be reconstructed from the bipartite graph among m-sets and $(m-1)$-sets.

Next we formulate $(k, c)(\leq)$-SETGNN formally. Let $h^{(t)}(\hat{\boldsymbol{v}}) \in \mathbb{R}^{d_t}$ denote the vector representation of supernode $\hat{\boldsymbol{v}}$ on the t-th iteration of $(k, c)(\leq)$-SETGNN. For any input graph G, it initializes representations of supernodes by

$$h^{(0)}(\hat{\boldsymbol{v}}) = \text{BaseGNN}(G[\hat{\boldsymbol{v}}]) \tag{4.9}$$

where the BaseGNN can be any GNN model. Theoretically the BaseGNN should be chosen to encode non-isomorphic induced subgraphs distinctly, mimicking HASH. Based on empirical tests of several GNNs on all (11,117) possible 8-node non-isomorphic graphs [Bal+21], and given GIN [Xu+19] is simple and fast with nearly 100% separation rate, we use GIN as our BaseGNN in experiments. Notice that we also concatenate an encoding of $|\hat{\boldsymbol{v}}|$ to the initial representation $h^{(0)}(\hat{\boldsymbol{v}})$ [2].

$(k, c)(\leq)$-SETGNN iteratively updates representations of all supernodes by

$$h^{(t+\frac{1}{2})}(\hat{\boldsymbol{v}}) = \sum_{\hat{\boldsymbol{u}} \in \mathcal{N}^G_{\text{right}}(\hat{\boldsymbol{v}})} \text{MLP}^{(t+\frac{1}{2})}(h^{(t)}(\hat{\boldsymbol{u}})) \tag{4.10}$$

$$h^{(t+1)}(\hat{\boldsymbol{v}}) = \text{MLP}^{(t)}\Big(h^{(t)}(\hat{\boldsymbol{v}}), h^{(t+\frac{1}{2})}(\hat{\boldsymbol{v}}), \sum_{\hat{\boldsymbol{u}} \in \mathcal{N}^G_{\text{left}}(\hat{\boldsymbol{v}})} \text{MLP}^{(t)}_A(h^{(t)}(\hat{\boldsymbol{u}})), \sum_{\hat{\boldsymbol{u}} \in \mathcal{N}^G_{\text{left}}(\hat{\boldsymbol{v}})} \text{MLP}^{(t)}_B(h^{(t+\frac{1}{2})}(\hat{\boldsymbol{u}}))\Big) \tag{4.11}$$

Then after T iterations, we compute the graph level encoding as

$$h^{(T)}(G) = \text{POOL}(\{\!\{h^{(T)}(\hat{\boldsymbol{v}}) \mid \hat{\boldsymbol{v}} \in V(S_{k,c\text{-swl}})\}\!\}) \tag{4.12}$$

[2] We view the size of $\hat{\boldsymbol{v}}$ discrete/categorical and pass it to an embedding layer to get encoding.

where POOL can be chosen as summation. We visualize the steps in Figure 4.1. Under mild conditions, $(k,c)(\leq)$-SetGNN has the same expressivity as $(k,c)(\leq)$-SetWL.

Theorem 4.4.1. *When (i) BaseGNN can distinguish any non-isomorhpic graphs with at most k nodes, (ii) all MLPs have sufficient depth and width, and (iii) POOL is an injective function, then for any $t \in \mathbb{N}$, t-layer $(k,c)(\leq)$-SetGNN is as expressive as $(k,c)(\leq)$-SetWL at the t-th iteration.*

The following facts can be derived easily from Theorem 4.3.5 and Theorem 4.4.1.

Corollary 4.4.2. $(k,c)(\leq)$-SetGNN *is progressively-expressive with increasing k and c, that is,*
(1) when $c_1 > c_2$, $(k,c_1)(\leq)$-SetGNN is more expressive than $(k,c_2)(\leq)$-SetGNN, and
(2) when $k_1 > k_2$, $(k_1,c)(\leq)$-SetGNN is more expressive than $(k_2,c)(\leq)$-SetGNN.

4.4.2 Bidirectional Sequential Message Passing

The t-th layer of $(k,c)(\leq)$-SetGNN (Eq. (4.10) and Eq. (4.11)) are essentially propagating information back and forth on the super-graph $S_{k,c\text{-swl}}(G)$, which is a sequence of $k-1$ bipartite graphs (see the middle of Figure 4.1), in *parallel* for all supernodes. We propose to change it to bidirectional blockwise *sequential* message passing, which we call $(k,c)(\leq)$-SetGNN*, defined as follows.

$$m = k-1 \text{ to } 1, \forall m\text{-set } \hat{\boldsymbol{v}}, h^{(t+\frac{1}{2})}(\hat{\boldsymbol{v}}) = \text{MLP}^{(t)}_{m,1}\Big(h^{(t)}(\hat{\boldsymbol{v}}), \sum_{\hat{\boldsymbol{u}}\in\mathcal{N}^G_{\text{right}}(\hat{\boldsymbol{v}})} \text{MLP}^{(t)}_{m,2}(h^{(t+\frac{1}{2})}(\hat{\boldsymbol{u}}))\Big) \tag{4.13}$$

$$m = 2 \text{ to } k, \forall m\text{-set } \hat{\boldsymbol{v}}, h^{(t+1)}(\hat{\boldsymbol{v}}) = \text{MLP}^{(t+\frac{1}{2})}_{m,1}\Big(h^{(t+\frac{1}{2})}(\hat{\boldsymbol{v}}), \sum_{\hat{\boldsymbol{u}}\in\mathcal{N}^G_{\text{left}}(\hat{\boldsymbol{v}})} \text{MLP}^{(t+\frac{1}{2})}_{m,2}(h^{(t+1)}(\hat{\boldsymbol{u}}))\Big) \tag{4.14}$$

Notice that $(k,c)(\leq)$-SetGNN* has lower memory usage, as $(k,c)(\leq)$-SetGNN load the complete supergraph ($k-1$ bipartites) while $(k,c)(\leq)$-SetGNN* loads 1 out of $k-1$ bipartites at a time, which is beneficial for limited-size GPUs. What is more, for a small, finite t it is even more expressive than $(k,c)(\leq)$-SetGNN. We provide both implementation of parallel and sequential message passing in the official github repository, while only report the performance of $(k,c)(\leq)$-SetGNN* given its efficiency and better expressivity.

Theorem 4.4.3. *For any $t \in \mathbb{N}$, the t-layer $(k,c)(\leq)$-SetGNN* is more expressive than the t-layer $(k,c)(\leq)$-SetGNN. As $\lim_{t\to\infty}$, $(k,c)(\leq)$-SetGNN is as expressive as $(k,c)(\leq)$-SetGNN*.*

4.4.3 Improving Supernode Initialization

Next we describe how to improve the supernode initialization (Eq. (4.9)) and extensively reduce computational and memory overhead for $c > 1$ without losing any expressivity. We achieve this by the fact that a graph with c components can be viewed as a set of c connected components. Formally,

Theorem 4.4.4. *Let G be a graph with c connected components $C_1, ..., C_c$, and G' be a graph also with c connected components $C'_1, ..., C'_c$, then G and G' are isomorphic if and only if $\exists p : [c] \to [c]$, s.t. $\forall i \in [c]$, C_i and $C'_{p(i)}$ are isomorphic.*

The theorem implies that we only need to apply Eq. (4.9) to all supernodes with a single component, and for any c-components $\hat{\boldsymbol{v}}$ with components $\hat{\boldsymbol{u}}_1, ..., \hat{\boldsymbol{u}}_c$, we can get the encoding by $h^{(0)}(\hat{\boldsymbol{v}}) = \text{DeepSet}(\{h^{(0)}(\hat{\boldsymbol{u}}_1), ..., h^{(0)}(\hat{\boldsymbol{u}}_c)\})$ without passing its induced subgraph to BaseGNN. This eliminates the heavy computation of passing a large number of induced subgraphs. We give the algorithm of building connection between $\hat{\boldsymbol{v}}$ to its single components in Appendix of [Zha+22b].

4.5 Experiments

We design experiments to answer the following questions. **Q1. Performance:** How does $(k,c)(\leq)$-SetGNN* compare to SOTA expressive GNNs? **Q2. Varying k and c:** How does the progressively increasing expressiveness reflect on generalization performance? **Q3. Computational requirements:** Is $(k,c)(\leq)$-SetGNN feasible on practical graphs w.r.t. running time and memory usage?

4.5.1 Setup

Datasets. To inspect the expressive power, we use four different types of simulation datasets: **1)** EXP [Abb+21] contains 600 pairs of 1&2-WL failed graphs which we split into two where graphs in each pair is assigned to two different classes; **2)** SR25 [Bal+21] has 15 strongly regular graphs (3-WL failed) with 25 nodes each, which we transform to a 15-way classification task; **3)** Substructure counting (i.e. triangle, tailed triangle, star and 4-cycle) tasks on random graph dataset [Che+20b]; **4)** Graph property regression (i.e. connectedness, diameter, radius) tasks on random graph dataset [Cor+20]. We also evaluate performance on two real world graph learning tasks: **5)** ZINC-12K [Dwi+20], and **6)** QM9 [Wu+18] for molecular property regression. See Table 4.1 for detailed dataset statistics.

Table 4.1: Dataset statistics.

Dataset	Task	# Cls./Tasks	# Graphs	Avg. # Nodes	# Edges
EXP [Abb+21]	Distinguish 1-WL failed graphs	2	1200	44.4	110.2
SR25 [Bal+21]	Distinguish 3-WL failed graphs	15	15	25	300
CountingSub. [Che+20b]	Regress num. of substructures	4	1500 / 1000 / 2500	18.8	62.6
GraphProp. [Cor+20]	Regress global graph properties	3	5120 / 640 / 1280	19.5	101.1
ZINC-12K [Dwi+20]	Regress molecular property	1	10000 / 1000 / 1000	23.1	49.8
QM9 [Wu+18]	Regress molecular properties	19[3]	130831	18.0	37.3

Baselines. We use GCN [KW17], GIN [Xu+19], PNA* [Cor+20], PPGN [Mar+19a], PF-GNN [DDL22], and GNN-AK [Zha+22c] as baselines on the simulation datasets. On ZINC-12K we also reference CIN [Bod+21a] directly from literature. Most baselines results are taken from [Zha+22c]. Finally, we compare to GINE [Hu+20a] on QM9.

Hyperparameter and model configurations. Due to limited time and resource, we highly restrict the hyperparameters and fix most of hyperparameters the same across all models and baselines to ensure a fair comparison. This means the performance of $(k,c)(\leq)$-SetGNN* reported in the paper is not the best performance given that we didn't tune much hyperparameters. Nevertheless the performance still reflects the theory and designs proposed in the paper, and we postpone studying the SOTA performance of $(k,c)(\leq)$-SetGNN* to future work. To be clear, we fix batch size to 128, the hidden size to 128, the number of layers of Base GNN to 4, and the number of layers of $(k,c)(\leq)$-SetGNN* (the number of iterations of $(k,c)(\leq)$-SetWL) to be 2 (we will do ablation study over it later). This hyperparameters configuration is used for all datasets. We have run the baseline GINE over many

[3]We use the version of the dataset from PyG [FL19], and it contains 19 tasks.

TABLE 4.2: Simulation data performances. For $(k, c)(\leq)$-SETGNN*, (k, c) values that achieve reported performance in parenthesis. (ACC: accuracy, MA[S]E: mean abs.[sq.] error)

Method	EXP (ACC)	SR25 (ACC)	Counting Substructures (MAE)				Graph Properties ($\log_{10}$(MSE))		
			Triangle	Tailed Tri.	Star	4-Cycle	IsConnected	Diameter	Radius
GCN	50%	6.67%	0.4186	0.3248	0.1798	0.2822	-1.7057	-2.4705	-3.9316
GIN	50%	6.67%	0.3569	0.2373	0.0224	0.2185	-1.9239	-3.3079	-4.7584
PNA*	50%	6.67%	0.3532	0.2648	0.1278	0.2430	-1.9395	-3.4382	-4.9470
PPGN	**100%**	6.67%	0.0089	0.0096	0.0148	0.0090	-1.9804	-3.6147	-5.0878
GIN-AK$^+$	**100%**	6.67%	0.0123	0.0112	0.0150	0.0126	-2.7513	-3.9687	-5.1846
PNA*-AK$^+$	**100%**	6.67%	0.0118	0.0138	0.0166	0.0132	-2.6189	-3.9011	**-5.2026**
$(k, c)(\leq)$	**100%**	**100%**	**0.0073**	**0.0075**	**0.0134**	**0.0075**	**-5.4667**	**-4.0800**	-5.1603
	(≥3, ≥2)	(≥4, ≥1)	(3, 2)	(4, 1)	(3, 2)	(4, 1)	(4, 2)	(4, 1)	(2, 2)

datasets, and we tune the number of layers from [4,6] and keep other hyperparameters the same as $(k, c)(\leq)$-SETGNN*. For all other baselines, we took the performance reported in [Zha+22c].

For all datasets except QM9, we follow the same configuration used in [Zha+22c]. For QM9, we use the dataset from PyG and conduct regression over all 19 targets simultaneously. To balance the scale of each target, we preprocess the dataset by standardizing every target to a Gaussian distribution with mean 0 and standard derivation 1. The dataset is randomly split with ratio 80%/10%/10% to train/validation/test sets (with a fixed random state so that all runs and models use the same split). For every graph in QM9, it contains 3d positional coordinates for all nodes, and we use them to augment edge features by using the absolute difference between the coordinates of two nodes on an edge for all models. Notice that our goal is not to achieve SOTA performance on QM9 but mainly to verify our theory and effectiveness of designs.

We use Batch Normalization and ReLU activation in all models. We use Adam optimizer with learning rate 0.001 in all experiments for optimization. We repeat all experiments three times (for random initialization) to calculate mean and standard derivation. All experiments are conducted on V100 and RTX-A6000 GPUs.

4.5.2 Results

Theorem 4.3.2 shows that k-MULTISETWL is able to distinguish CFI(k) graphs. It also holds for $k(\leq)$-SETWL as the proof doesn't use repetitions. We implemented the construction of CFI(k) graphs for any k. Empirically we found that $(k, c)(\leq)$-SETGNN* *is able to distinguish CFI(k) for k = 3, 4, 5 (k>6 out of memory).* This empirically verifies its theoretical expressivity.

Table 4.2 shows the performance results on all the simulation datasets. The low expressive models such as GCN, GIN and PNA* underperform on EXP and SR25, while 3-WL equivalent PPGN excels only on EXP. Notably, $(k, c)(\leq)$-SETGNN* achieves 100% discriminating power with any k larger than 2 and 3, and any c larger than 1 and 0, resp. for EXP and SR25. $(k, c)(\leq)$-SETGNN* also outperforms the baselines on all substructure counting tasks, as well as on two out of three graph property prediction tasks (except Radius) with significant gap, for relatively small values of k and c.

In Table 4.3, we show the train and test MAEs on substructure counting tasks for individual values of k and c. As expected, performances improve for increasing k when c is fixed, and vice versa. It is notable that orders of magnitude improvements on test error occur moving from k=2 to 3 for the triangle tasks as well as the star task, while a similarly large magnitude drop is obtained at k=4 for the 4-cycle task, which is expected as triangle and 4-cycle have 3 and 4 nodes respectively. Similar observations hold for graph property tasks as well. (See Table 4.4.)

Table 4.3: Train and Test performances on substructure counting tasks by varying k and c. Notice the orders of magnitude drop in Test MAE between bolded entries per task.

		Counting Substructures (MAE)							
		Triangle		Tailed Tri.		Star		4-Cycle	
k	c	Train	Test	Train	Test	Train	Test	Train	Test
2	1	0.9941 ± 0.2623	**1.1409 ± 0.1224**	1.1506 ± 0.2542	**0.8695 ± 0.0781**	1.5348 ± 2.0697	**2.3454 ± 0.8198**	1.2159 ± 0.0292	0.8361 ± 0.1171
3	1	0.0311 ± 0.0025	**0.0088 ± 0.0001**	0.0303 ± 0.0108	**0.0085 ± 0.0018**	0.0559 ± 0.0019	**0.0151 ± 0.0006**	0.1351 ± 0.0058	**0.1893 ± 0.0030**
4	1	0.0321 ± 0.0008	0.0151 ± 0.0074	0.0307 ± 0.0085	0.0075 ± 0.0012	0.0687 ± 0.0104	0.0339 ± 0.0009	0.0349 ± 0.0007	**0.0075 ± 0.0002**
5	1	0.0302 ± 0.0070	0.0208 ± 0.0042	0.0553 ± 0.0009	0.0189 ± 0.0024	0.0565 ± 0.0078	0.0263 ± 0.0023	0.0377 ± 0.0057	0.0175 ± 0.0036
6	1	0.0344 ± 0.0024	0.0247 ± 0.0085	0.0357 ± 0.0017	0.0171 ± 0.0000	0.0560 ± 0.0000	0.0168 ± 0.0022	0.0356 ± 0.0014	0.0163 ± 0.0064
2	2	0.3452 ± 0.0329	0.4029 ± 0.0053	0.2723 ± 0.0157	0.2898 ± 0.0055	0.0466 ± 0.0025	0.0242 ± 0.0006	0.2369 ± 0.0123	0.2512 ± 0.0029
3	2	0.0234 ± 0.0030	0.0073 ± 0.0009	0.0296 ± 0.0074	0.0100 ± 0.0009	0.0640 ± 0.0003	0.0134 ± 0.0006	0.0484 ± 0.0135	0.0194 ± 0.0065
4	2	0.0587 ± 0.0356	0.0131 ± 0.0010	0.0438 ± 0.0140	0.0094 ± 0.0002	0.0488 ± 0.0008	0.0209 ± 0.0063	0.0464 ± 0.0037	0.0110 ± 0.0020

Table 4.4: Train and Test performances of $(k,c)(\leq)$-SetGNN* on regressing graph properties by varying k and c.

		Regressing Graph Properties ($\log_{10}$(MSE))					
		Is Connected		Diameter		Radius	
k	c	Train	Test	Train	Test	Train	Test
2	1	-4.2266 ± 0.1222	-2.9577 ± 0.1295	-4.0347 ± 0.0468	-3.6322 ± 0.0458	-4.4690 ± 0.0348	-4.9436 ± 0.0277
3	1	-4.2360 ± 0.1854	-3.4631 ± 0.6392	-4.0228 ± 0.1256	-3.7885 ± 0.0589	-4.4762 ± 0.1176	-5.0245 ± 0.0881
4	1	-4.7776 ± 0.0386	-4.9941 ± 0.0913	-4.1396 ± 0.0442	-4.0122 ± 0.0071	-4.2837 ± 0.5880	-4.1528 ± 0.9383
2	2	-4.6623 ± 0.3170	-4.7848 ± 0.3150	-4.0802 ± 0.1654	-3.8962 ± 0.0124	-4.5362 ± 0.2012	-5.1603 ± 0.0610
3	2	-4.2601 ± 0.3192	-4.4547 ± 1.1715	-4.3235 ± 0.3050	-3.9905 ± 0.0799	-4.6766 ± 0.1797	-4.9836 ± 0.0658
4	2	-4.8489 ± 0.1354	-5.4667 ± 0.2125	-4.5033 ± 0.1610	-3.9495 ± 0.3202	-4.4130 ± 0.2686	-4.1432 ± 0.4405

In addition to simulation datasets, we evaluate our $(k,c)(\leq)$-SetGNN* on real-world data; Table 4.5 shows our performance on ZINC-12K. Our method achieves a new state-of-the-art performance, with a mean absolute error (MAE) of 0.0750, using k=5 and c=2.

In Table 4.6 we show the test and validation MAE along with training loss for varying k and c for ZINC-12K. For a fixed c, validation MAE and training loss both follow a first decaying and later increasing trend with increasing k, potentially owing to the difficulty in fitting with too many sets (i.e. supernodes) and edges in the super-graph.

Similar results are shown for QM9 in Table 4.7. For comparison we also show GINE performances using both 4 and 6 layers, both of which are significantly lower than $(k,c)(\leq)$-SetGNN*.

Table 4.5: SetGNN* achieves SOTA on ZINC-12K.

Method	MAE
GatedGCN	0.363 ± 0.009
GCN	0.321 ± 0.009
PNA	0.188 ± 0.004
DGN	0.168 ± 0.003
GIN	0.163 ± 0.004
GINE	0.157 ± 0.004
HIMP	0.151 ± 0.006
PNA*	0.140 ± 0.006
GSN	0.115 ± 0.012
PF-GNN	0.122 ± 0.010
GIN-AK$^+$	0.080 ± 0.001
CIN	0.079 ± 0.006
$(k,c)(\leq)$	**0.0750 ± 0.0027**

4.5.3 Computational requirements

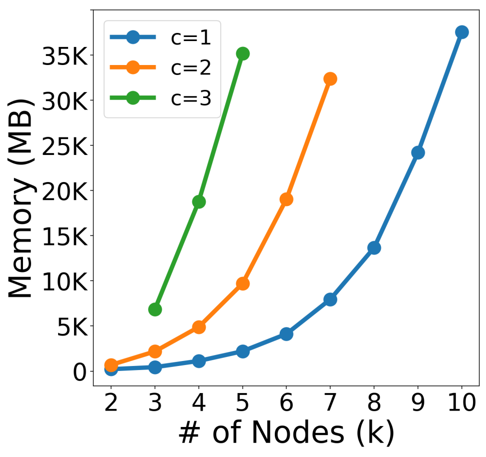
We next investigate how increasing k and c change the computational footprint of $(k,c)(\leq)$-SetGNN* in practice. Fig. 4.2 shows that increasing these parameters expectedly increases both memory consumption (in MB in (a)) as well as runtime (in seconds per epoch in (b)). Notably, since larger k and c increases our model's expressivity, we observe that suitable choices for k and c allow us to practically realize these increasingly expressive models on commodity hardware. With conservative values of c (e.g. c=1), we are able to consider passing messages between sets of k (e.g. 10) nodes far larger than k-WL-style higher order models can achieve ($\leq$3).

TABLE 4.6: $(k, c)(\leq)$-SETGNN* performances on ZINC-12K by varying (k,c). **Test MAE** at lowest Val. MAE, and lowest Test MAE.

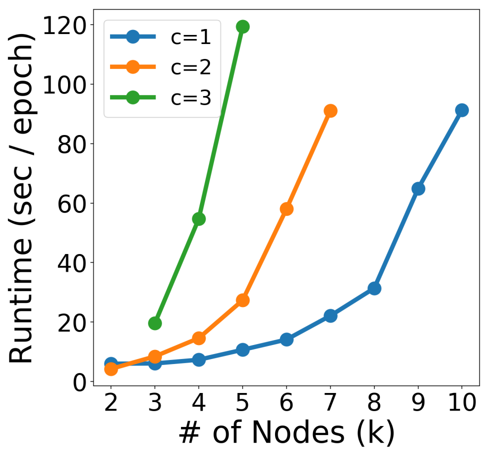
k	c	Train loss	Val. MAE	Test MAE
2	1	0.1381 ± 0.0240	0.2429 ± 0.0071	0.2345 ± 0.0131
3	1	0.1172 ± 0.0063	0.2298 ± 0.0060	0.2252 ± 0.0030
4	1	0.0693 ± 0.0111	0.1645 ± 0.0052	0.1636 ± 0.0052
5	1	0.0643 ± 0.0019	0.1593 ± 0.0051	0.1447 ± 0.0013
6	1	0.0519 ± 0.0064	0.0994 ± 0.0093	0.0843 ± 0.0048
7	1	0.0543 ± 0.0048	0.0965 ± 0.0061	0.0747 ± 0.0022
8	1	0.0564 ± 0.0152	0.0961 ± 0.0043	0.0732 ± 0.0037
9	1	0.0817 ± 0.0274	0.0909 ± 0.0094	0.0824 ± 0.0056
10	1	0.0894 ± 0.0266	0.1060 ± 0.0157	0.0950 ± 0.0102
2	2	0.1783 ± 0.0602	0.2913 ± 0.0102	0.2948 ± 0.0210
3	2	0.0640 ± 0.0072	0.1668 ± 0.0078	0.1391 ± 0.0102
4	2	0.0499 ± 0.0043	0.1029 ± 0.0033	0.0836 ± 0.0010
5	2	0.0483 ± 0.0017	0.0899 ± 0.0056	**0.0750 ± 0.0027**
6	2	0.0530 ± 0.0064	0.0927 ± 0.0050	0.0737 ± 0.0006
7	2	0.0547 ± 0.0036	0.0984 ± 0.0047	0.0784 ± 0.0043
3	3	0.0798 ± 0.0062	0.1881 ± 0.0076	0.1722 ± 0.0086
4	3	0.0565 ± 0.0059	0.1121 ± 0.0066	0.0869 ± 0.0026
5	3	0.0671 ± 0.0156	0.1091 ± 0.0097	0.0920 ± 0.0054

TABLE 4.7: $(k, c)(\leq)$-SETGNN* performances on QM9 by varying (k,c). **Test MAE** at lowest Val. MAE, and lowest Test MAE. All variances are $\leq$0.002 and thus omitted.

k	c	Train loss	Val. MAE	Test MAE
2	1	0.0376 ± 0.0005	0.0387 ± 0.0007	0.0389 ± 0.0008
3	1	0.0308 ± 0.0010	0.0386 ± 0.0017	0.0379 ± 0.0010
4	1	0.0338 ± 0.0003	0.0371 ± 0.0005	0.0370 ± 0.0006
5	1	0.0299 ± 0.0017	0.0343 ± 0.0008	0.0341 ± 0.0009
6	1	0.0226 ± 0.0004	0.0296 ± 0.0007	0.0293 ± 0.0007
7	1	0.0208 ± 0.0005	0.0289 ± 0.0007	0.0269 ± 0.0003
2	2	0.0367 ± 0.0007	0.0398 ± 0.0004	0.0398 ± 0.0004
3	2	0.0282 ± 0.0013	0.0358 ± 0.0009	0.0356 ± 0.0007
4	2	0.0219 ± 0.0004	0.0280 ± 0.0008	0.0278 ± 0.0008
5	2	0.0175 ± 0.0003	0.0267± 0.0005	**0.0251± 0.0006**
3	3	0.0391 ± 0.0107	0.0428 ± 0.0057	0.0425 ± 0.0052
4	3	0.0219 ± 0.0011	0.0301 ± 0.0010	0.0286 ± 0.0004
GINE (L=4)		0.0507 ± 0.0014	0.0478 ± 0.0003	0.0479 ± 0.0004
GINE (L=6)		0.0440 ± 0.0009	0.0440 ± 0.0009	0.0451 ± 0.0009



(A) Memory usage



(B) Training time

FIGURE 4.2: $(k, c)(\leq)$-SETGNN*'s footprint scales practically with both k and c in memory (a) and running time (b) – results on ZINC-12K.

4.6 Conclusion

Our work is motivated by the impracticality of higher-order GNN models based on the k-WL hierarchy, which make it challenging to study how much expressiveness real-world tasks truly necessitate. To this end, we proposed $(k, c)(\leq)$-SetWL, a more practical and progressively-expressive hierarchy with theoretical connections to k-WL and drastically lowered complexity. We also designed and implemented a practical model $(k, c)(\leq)$-SetGNN(*), expressiveness of which is gradually increased by larger k and c. Our model achieves strong performance, including several new best results on graph-level tasks like ZINC-12K and expressiveness-relevant tasks like substructure counting, while being practically trainable on commodity hardware.

Part III

Generative Model on Graphs

Chapter 5

Improving and Unifying Discrete Denoising Diffusion

Chapter is based on Lingxiao Zhao, Xueying Ding, Lijun Yu, and Leman Akoglu. "Improving and Unifying Discrete&Continuous-time Discrete Denoising Diffusion". In: *arXiv preprint arXiv:2402.03701* (2024). This chapter builds the foundation for the next chapter.

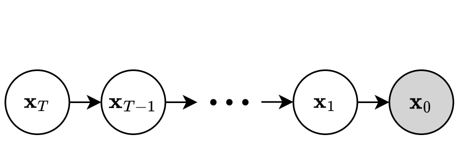
5.1 Introduction

Deep generative models have taken the world by storm, capturing complex data distributions and producing realistic data, from human-like text [Bro+20; Li+22; Ope23] and natural looking images [DN21; Ram+22; Zha+23b] to novel compounds like molecules and drugs [KC18; LPL21] and video synthesis [Ho+22]. Denoising diffusion models [HJA20], a powerful class of generative models, are trained through a forward diffusion process that gradually adds noise to the training samples, and a backward process that denoises these diffusion trajectories. New data are then generated by sampling from the noise distribution and employing the trained model for recursive denoising.

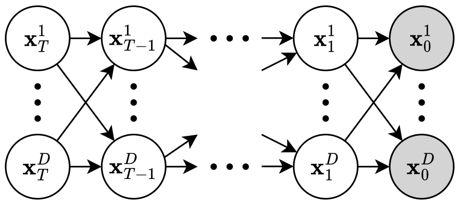
Discrete diffusion for categorical data has two modeling paradigms: discrete-time and continuous-time. The former discretizes time such that backward denoising is learned only at pre-specified time points. This limits generation, which can "jump back" through these fixed points only. In contrast, continuous-time diffusion allows a path through any point in range, and often yields higher sample quality.

Current literature on discrete-time discrete diffusion is relatively established, while only recently Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22] introduced the first continuous-time discrete diffusion framework. While groundbreaking, their their loss requires multiple evaluations at each time step during training. Moreover, the exact sampling through their learned backward process is extremely tedious for multi-dimensional variables. Due to mathematically complicated and computationally demanding formulations, Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22] propose nontrivial approximations for tractability with unknown errors.

In this paper, we present a series of *mathematical simplifications of the variational lower bound (VLB) loss* while keeping exactness, which enable more accurate and easy-to-optimize training for discrete-time and continuous-time discrete diffusion. In addition, we establish a *simpler reformulation of the backward denoising* probability that enables exact and accelerated sampling for both discrete-time and continuous-time discrete diffusion. Importantly, our simplified reformulations lead us



(a) Latent graphical model for diffusion



(b) Generalize (a) to multi-element object

FIGURE 5.1: Graphical model view

to an *elegant unification of the two modeling paradigms*; in particular, demonstrating that they share the same forward and backward procedures. The unification is not only mathematically elegant but also practically instrumental where the same source code can be used by both models up to a single alteration in the loss function during training. Further, our simplified analytical formulations allow both forward and now also backward probabilities to accommodate any noise distribution. This flexibility is particularly attractive for multi-element objects where each element can exhibit a different noise distribution. We summarize our main contributions as follows.

- **Loss Simplifications:** We derive simplified but exact VLB calculations for both discrete&continuous-time discrete diffusion—enabling more accurate and easy-to-optimize training that leads to SOTA performance.
- **Mathematical Unification:** Through a simplified reformulation of backward denoising, this is the *first work to unify discrete-&continuous-time discrete diffusion*—enabling flexibility and speed-up in generation as well as training with various noise distributions.
- **Extensive Evaluation:** We propose a Unified and Simplified Discrete Denoising Diffusion model called USD3 that outperforms both discrete-&continuous-time SOTA models on established datasets.

5.2 Discrete-time Discrete Diffusion

Notation: Let $\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x}_0)$ be the random variable of observed data with underlying distribution $p_{\text{data}}(\mathbf{x}_0)$. Let $\mathbf{x}_t \sim q(\mathbf{x}_t)$ be the latent variable at time t of a single-element object, like a pixel of an image or a node/edge of a graph, with maximum time T. Let $\mathbf{x}_{t|s} \sim q(\mathbf{x}_t|\mathbf{x}_s)$ be the conditional random variable. We model the *forward diffusion* process independently for each element of the object, while the *backward denoising* process is modeled jointly for all elements of the object. For simplicity and clarity of presentation, we first assume that the object only has 1 element and extend to multi-element object later. Let $\mathbf{x}_0^{1:D}$ denote the object with D elements, and $\mathbf{x}_t^i$ be the i-th element of latent object at time t. We assume all random variables take categorical values from $\{1, 2, ..., K\}$. Let $\boldsymbol{e}_k \in \{0, 1\}^K$ be the one-hot encoding of category k. For a random variable $\mathbf{x}$, we use $\boldsymbol{x}$ denoting its one-hot encoded sample where $\boldsymbol{x} \in \{\boldsymbol{e}_1, ..., \boldsymbol{e}_K\}$. Also, we interchangeably use $q(\mathbf{x}_t|\mathbf{x}_s)$, $q(\mathbf{x}_t = \boldsymbol{x}_t|\mathbf{x}_s = \boldsymbol{x}_s)$, and $q_{t|s}(\boldsymbol{x}_t|\boldsymbol{x}_s)$ when no ambiguity is introduced. Let $\langle\cdot,\cdot\rangle$ denote inner product. All vectors are column-wise vectors.

5.2.1 Graphical Model View of Diffusion Models

Diffusion models can be represented by latent variable graphical models (see Fig. 5.1). We can write the joint probability as $p_\theta(\mathbf{x}_{0:T}) := p_\theta(\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_T) = p_\theta(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ using the Markov condition. Parameters θ are learned by maximizing the loglikehood of the observed variable $\mathbf{x}_0$: $\log p_\theta(\mathbf{x}_0) = \log \int p_\theta(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T}$. However the marginalization is intractable, and instead the following variational lower bound (VLB) is used.

$$\log p_\theta(\mathbf{x}_0) = \log \int q(\mathbf{x}_{1:T}|\mathbf{x}_0) \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} d\mathbf{x}_{1:T} \tag{5.1}$$

$$\geq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \big[\log p_\theta(\mathbf{x}_{0:T}) \big] - \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \big[\log q(\mathbf{x}_{1:T}) \big]; . \tag{5.2}$$

The above inequality holds for any conditional probability $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ and finding the best $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ to tighten the bound is the inference problem in graphical models (i.e. E step in EM algorithm). Exact inference is intractable, thus $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ in diffusion models is fixed or chosen specifically to simplify the learning objective. To simplify Eq. (5.1) further, it is important to assume $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ is *decomposable*. The typical assumption is that $q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$ [HJA20], which we also adopt in this paper. Others that have been explored include $q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)$ [SME21].

Assuming $q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$, Eq. (5.1) can be simplified as

$$\underbrace{\mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)} \big[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1) \big]}_{-\mathcal{L}_1(\theta)} - \underbrace{D_{\text{KL}}\big(q(\mathbf{x}_T|\mathbf{x}_0)||p_\theta(\mathbf{x}_T)\big)}_{\mathcal{L}_{\text{prior}}}$$
$$- \sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} \big[D_{\text{KL}}\big(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\big) \big]}_{\mathcal{L}_t(\theta)} , \tag{5.3}$$

where $\mathcal{L}_{\text{prior}} \approx 0$, since $p_\theta(\mathbf{x}_T) \approx q(\mathbf{x}_T|\mathbf{x}_0)$ is designed as a fixed noise distribution that is easy to sample from. (See Appx. of [Zha+24b] for derivation.) To compute Eq. (5.3), we need to formalize distributions (*i*) $q(\mathbf{x}_t|\mathbf{x}_0)$ and (*ii*) $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$, as well as (*iii*) the parameterization of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$. We specify these respectively in §5.2.2, §5.2.3, and §5.2.4.

After reviewing the forward process for discrete diffusion (§5.2.2), here we contribute a series of analytical simplifications for various components (§5.2.3, §5.2.4) of the VLB, providing exact closed-form formulation in §5.2.5, as well as an approximated loss for easier optimization in §5.2.6. Further, we present fast backward sampling in §5.2.7, and the extension to multi-element case can be find in [Zha+24b].

5.2.2 the Forward Diffusion Process

We assume each discrete random variable $\mathbf{x}_t$ has a categorical distribution, i.e. $\mathbf{x}_t \sim \text{Cat}(\mathbf{x}_t; \boldsymbol{p})$ with $\boldsymbol{p} \in [0,1]^K$ and $\mathbf{1}^\top \boldsymbol{p} = 1$. One can verify that $p(\mathbf{x}_t = \boldsymbol{x}_t) = \boldsymbol{x}_t^\top \boldsymbol{p}$, or simply $p(\mathbf{x}_t) = \boldsymbol{x}_t^\top \boldsymbol{p}$. As shown in [Hoo+21; Aus+21], the forward process with discrete variables $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ can be represented as a transition matrix $Q_t \in [0,1]^{K \times K}$ such that $[Q_t]_{ij} = q(\mathbf{x}_t = \boldsymbol{e}_j|\mathbf{x}_{t-1} = \boldsymbol{e}_i)$. Then, we can write the distribution explicitly as

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \text{Cat}(\mathbf{x}_t; Q_t^\top \boldsymbol{x}_{t-1}) . \tag{5.4}$$

Given transition matrices $Q_1, ..., Q_T$, we can get the t-step marginal distribution conditioning on s-step $(t > s)$ as

$$q(\mathbf{x}_t|\mathbf{x}_s) = \text{Cat}(\mathbf{x}_t; \overline{Q}_{t|s}^\top \boldsymbol{x}_s), \text{with } \overline{Q}_{t|s} = Q_{s+1} ... Q_t . \tag{5.5}$$

The s-step posterior distribution conditioning on $\mathbf{x}_0$ and t-step can be derived as

$$q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0) = \frac{q(\mathbf{x}_t|\mathbf{x}_s) q(\mathbf{x}_s|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} = \text{Cat}(\mathbf{x}_s; \frac{\overline{Q}_{t|s} \boldsymbol{x}_t \odot \overline{Q}_s^\top \boldsymbol{x}_0}{\boldsymbol{x}_t^\top \overline{Q}_t^\top \boldsymbol{x}_0}). \ \forall t > s \tag{5.6}$$

The above formulations are valid (see derivation in Appx. of [Zha+24b]) for *any* transition matrices $Q_1, ..., Q_T$, which however should be chosen such that every row of $\overline{Q}_t = \overline{Q}_{t|0}$ converge to the same known stationary distribution when t becomes large (i.e. at T). Let the known stationary distribution be $\mathbf{m}_0 \sim \mathrm{Cat}(\mathbf{m}_0; \boldsymbol{m})$. Then, the constraint can be stated as

$$\lim_{t \to T} \overline{Q}_t = \mathbf{1}\boldsymbol{m}^\top . \tag{5.7}$$

In addition, this paper focuses on *nominal data* where categories are unordered and only equality comparison is defined. Hence, no ordering prior *except checking equality* should be used to define each transition matrix Q_t[1]. To achieve the desired convergence on nominal data while keeping the flexibility of choosing any categorical stationary distribution $\mathbf{m}_0 \sim \mathrm{Cat}(\mathbf{m}_0; \boldsymbol{m})$, we define Q_t as

$$Q_t = \alpha_t I + (1 - \alpha_t)\mathbf{1}\boldsymbol{m}^\top , \tag{5.8}$$

where $\alpha_t \in [0, 1]$. This results in the accumulated transition matrix $\overline{Q}_{t|s}$ being equal to

$$\overline{Q}_{t|s} = \overline{\alpha}_{t|s} I + (1 - \overline{\alpha}_{t|s})\mathbf{1}\boldsymbol{m}^\top \ \forall t > s , \tag{5.9}$$

where $\overline{\alpha}_{t|s} = \prod_{i=s+1}^{t} \alpha_i$. Note that $\overline{\alpha}_t = \overline{\alpha}_{t|0} = \overline{\alpha}_{t|s}\overline{\alpha}_s$. We achieve Eq. (5.7) by picking α_t such that $\lim_{t \to T} \overline{\alpha}_t = 0$.

5.2.3 Form of $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$

The formulation in Eq. (5.8) can be used to simplify $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$. We provide a general formulation of $q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0)$ for any s, t with $0 < s < t \leq T$, which will be useful for unifying with continuous-time diffusion. One can recover $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ by setting s as $t-1$.

Proposition 5.2.1. *For both discrete- and continuous-time discrete diffusion, we can write the conditional distribution as*

$$q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0) = \begin{cases} Cat\Big(\mathbf{x}_s;\ (1 - \lambda_{t|s}) \cdot \boldsymbol{x}_t + \lambda_{t|s} \cdot \boldsymbol{m}\Big) & \text{when } \boldsymbol{x}_t = \boldsymbol{x}_0 \\ Cat\Big(\mathbf{x}_s;\ (1 - \mu_{t|s}) \cdot \boldsymbol{x}_0 + \mu_{t|s}\overline{\alpha}_{t|s} \cdot \boldsymbol{x}_t & \\ \qquad\qquad + \mu_{t|s}(1 - \overline{\alpha}_{t|s}) \cdot \boldsymbol{m}\Big) & \text{when } \boldsymbol{x}_t \neq \boldsymbol{x}_0 \end{cases} \tag{5.10}$$

where $\lambda_{t|s}$ and $\mu_{t|s}$ are defined as

$$\lambda_{t|s} = \frac{(1 - \overline{\alpha}_s)(1 - \overline{\alpha}_{t|s})\langle \boldsymbol{m}, \boldsymbol{x}_t \rangle}{\overline{\alpha}_t + (1 - \overline{\alpha}_t)\langle \boldsymbol{m}, \boldsymbol{x}_t \rangle},\ \ \mu_{t|s} = \frac{1 - \overline{\alpha}_s}{1 - \overline{\alpha}_t} . \tag{5.11}$$

We remark that the above formulation is a generalization of the result shown in [Zhe+23].

Proof. First, let us define $\overline{Q}_{t|s} = Q_{s+1}...Q_t$. Note that $\overline{Q}_{t|0} = \overline{Q}_t$ and $\overline{Q}_{t|t-1} = Q_t$. Accordingly, we can derive the following two equalities.

$$q(\mathbf{x}_t|\mathbf{x}_s) = \mathrm{Cat}(\mathbf{x}_t; \overline{Q}_{t|s}^\top \mathbf{x}_s) \tag{5.12}$$

[1]Mathematically, this means $\forall k, j,\ q(\mathbf{x}_t|\mathbf{x}_{t-1} = \boldsymbol{e}_j, \mathbf{x}_t \neq \{\boldsymbol{e}_j, \boldsymbol{e}_k\}) = q(\mathbf{x}_t|\mathbf{x}_{t-1} = \boldsymbol{e}_k, \mathbf{x}_t \neq \{\boldsymbol{e}_j, \boldsymbol{e}_k\})$.

$$
\begin{aligned}
q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0) &= \frac{q(\mathbf{x}_t|\mathbf{x}_s)q(\mathbf{x}_s|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} = \frac{\mathrm{Cat}(\mathbf{x}_t; \overline{Q}_{t|s}^\top \mathbf{x}_s)\mathrm{Cat}(\mathbf{x}_s; \overline{Q}_s^\top \mathbf{x}_0)}{\mathrm{Cat}(\mathbf{x}_t; \overline{Q}_t^\top \mathbf{x}_0)} \\
&= \frac{\mathbf{x}_s^\top \overline{Q}_{t|s}\mathbf{x}_t \cdot \mathbf{x}_s^\top \overline{Q}_s^\top \mathbf{x}_0}{\mathbf{x}_t^\top \overline{Q}_t^\top \mathbf{x}_0} = \mathbf{x}_s^\top \frac{\overline{Q}_{t|s}\mathbf{x}_t \odot \overline{Q}_s^\top \mathbf{x}_0}{\mathbf{x}_t^\top \overline{Q}_t^\top \mathbf{x}_0} = \mathrm{Cat}(\mathbf{x}_s; \frac{\overline{Q}_{t|s}\mathbf{x}_t \odot \overline{Q}_s^\top \mathbf{x}_0}{\mathbf{x}_t^\top \overline{Q}_t^\top \mathbf{x}_0})
\end{aligned} \tag{5.13}
$$

Using the formulation of Q_t in Eq. (5.8), $\overline{Q}_{t|s}$ can be written as

$$
\overline{Q}_{t|s} = \overline{\alpha}_{t|s} I + (1 - \overline{\alpha}_{t|s})\mathbf{1}\boldsymbol{m}^\top \tag{5.14}
$$

where $\overline{\alpha}_{t|s} = \prod_{i=s+1}^{t} \alpha_i$. Note that $\overline{\alpha}_t = \overline{\alpha}_{t|s}\overline{\alpha}_s$.

Next, we can simplify Eq. (5.6) using the above formulations as

$$
\begin{aligned}
\frac{\overline{Q}_{t|s}\mathbf{x}_t \odot \overline{Q}_s^\top \mathbf{x}_0}{\mathbf{x}_t^\top \overline{Q}_t^\top \mathbf{x}_0} &= \frac{\big(\overline{\alpha}_{t|s}\mathbf{x}_t + (1 - \overline{\alpha}_{t|s})\langle \boldsymbol{m}, \mathbf{x}_t\rangle \mathbf{1}\big) \odot \big(\overline{\alpha}_s \mathbf{x}_0 + (1 - \overline{\alpha}_s)\boldsymbol{m}\big)}{\overline{\alpha}_t\langle \mathbf{x}_t, \mathbf{x}_0\rangle + (1 - \overline{\alpha}_t)\langle \boldsymbol{m}, \mathbf{x}_t\rangle} \\
&= \frac{\overline{\alpha}_t \mathbf{x}_t \odot \mathbf{x}_0 + (\overline{\alpha}_s - \overline{\alpha}_t)\langle \boldsymbol{m}, \mathbf{x}_t\rangle \mathbf{x}_0 + (\overline{\alpha}_{t|s} - \overline{\alpha}_t)\mathbf{x}_t \odot \boldsymbol{m} + (1 - \overline{\alpha}_s)(1 - \overline{\alpha}_{t|s})\langle \boldsymbol{m}, \mathbf{x}_t\rangle \boldsymbol{m}}{\overline{\alpha}_t\langle \mathbf{x}_t, \mathbf{x}_0\rangle + (1 - \overline{\alpha}_t)\langle \boldsymbol{m}, \mathbf{x}_t\rangle}
\end{aligned} \tag{5.15}
$$

We can simplify the above equation further by considering two cases: (1) $\mathbf{x}_t = \mathbf{x}_0$ and (2) $\mathbf{x}_t \neq \mathbf{x}_0$.

(Case 1) When $\mathbf{x}_t = \mathbf{x}_0$, using the fact that both $\mathbf{x}_t$ and $\mathbf{x}_0$ are one-hot encoded, we observe

$$
\begin{aligned}
\text{Eq. (5.15)} &= \frac{\overline{\alpha}_t \mathbf{x}_t + (\overline{\alpha}_s - \overline{\alpha}_t)\langle \boldsymbol{m}, \mathbf{x}_t\rangle \mathbf{x}_t + (\overline{\alpha}_{t|s} - \overline{\alpha}_t)\langle \boldsymbol{m}, \mathbf{x}_t\rangle \mathbf{x}_t + (1 - \overline{\alpha}_s)(1 - \overline{\alpha}_{t|s})\langle \boldsymbol{m}, \mathbf{x}_t\rangle \boldsymbol{m}}{\overline{\alpha}_t + (1 - \overline{\alpha}_t)\langle \boldsymbol{m}, \mathbf{x}_t\rangle} \\
&= \frac{\overline{\alpha}_t + (\overline{\alpha}_{t|s} + \overline{\alpha}_s - 2\overline{\alpha}_t)\langle \boldsymbol{m}, \mathbf{x}_t\rangle}{\overline{\alpha}_t + (1 - \overline{\alpha}_t)\langle \boldsymbol{m}, \mathbf{x}_t\rangle}\mathbf{x}_t + \frac{(1 - \overline{\alpha}_s)(1 - \overline{\alpha}_{t|s})\langle \boldsymbol{m}, \mathbf{x}_t\rangle}{\overline{\alpha}_t + (1 - \overline{\alpha}_t)\langle \boldsymbol{m}, \mathbf{x}_t\rangle}\boldsymbol{m} \\
&= (1 - \lambda_{t|s}) \cdot \mathbf{x}_t + \lambda_{t|s} \cdot \boldsymbol{m}\ ,
\end{aligned} \tag{5.16}
$$

where $\lambda_{t|s}$ is defined as

$$
\lambda_{t|s} = \frac{(1 - \overline{\alpha}_s)(1 - \overline{\alpha}_{t|s})\langle \boldsymbol{m}, \mathbf{x}_t\rangle}{\overline{\alpha}_t + (1 - \overline{\alpha}_t)\langle \boldsymbol{m}, \mathbf{x}_t\rangle}\ . \tag{5.17}
$$

(Case 2) When $\mathbf{x}_t \neq \mathbf{x}_0$, we can similarly derive

$$
\begin{aligned}
\text{Eq. (5.15)} &= \frac{(\overline{\alpha}_s - \overline{\alpha}_t)\langle \boldsymbol{m}, \mathbf{x}_t\rangle \mathbf{x}_0 + (\overline{\alpha}_{t|s} - \overline{\alpha}_t)\langle \boldsymbol{m}, \mathbf{x}_t\rangle \mathbf{x}_t + (1 - \overline{\alpha}_s)(1 - \overline{\alpha}_{t|s})\langle \boldsymbol{m}, \mathbf{x}_t\rangle \boldsymbol{m}}{(1 - \overline{\alpha}_t)\langle \boldsymbol{m}, \mathbf{x}_t\rangle} \\
&= \frac{\overline{\alpha}_s - \overline{\alpha}_t}{1 - \overline{\alpha}_t}\mathbf{x}_0 + \frac{1 - \overline{\alpha}_s}{1 - \overline{\alpha}_t}\overline{\alpha}_{t|s}\mathbf{x}_t + \frac{1 - \overline{\alpha}_s}{1 - \overline{\alpha}_t}(1 - \overline{\alpha}_{t|s})\boldsymbol{m} \\
&= (1 - \mu_{t|s}) \cdot \mathbf{x}_0 + \mu_{t|s}\overline{\alpha}_{t|s} \cdot \mathbf{x}_t + \mu_{t|s}(1 - \overline{\alpha}_{t|s}) \cdot \boldsymbol{m}\ ,
\end{aligned} \tag{5.18}
$$

where $\mu_{t|s}$ is defined as

$$
\mu_{t|s} = \frac{1 - \overline{\alpha}_s}{1 - \overline{\alpha}_t} \tag{5.19}
$$

Combining the results from both cases together, we can write $q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0)$ in the following form.

$$
q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0) = \begin{cases} \mathrm{Cat}\Big(\mathbf{x}_s;\ (1 - \lambda_{t|s}) \cdot \mathbf{x}_t + \lambda_{t|s} \cdot \boldsymbol{m}\Big) & \text{when } \mathbf{x}_t = \mathbf{x}_0 \\ \mathrm{Cat}\Big(\mathbf{x}_s;\ (1 - \mu_{t|s}) \cdot \mathbf{x}_0 + \mu_{t|s}\overline{\alpha}_{t|s} \cdot \mathbf{x}_t + \mu_{t|s}(1 - \overline{\alpha}_{t|s}) \cdot \boldsymbol{m}\Big) & \text{when } \mathbf{x}_t \neq \mathbf{x}_0 \end{cases} \tag{5.20}
$$

□

5.2.4 Parameterization of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$

The literature has explored three different parameterizations of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$: (1) parameterizing $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ directly; (2) parameterizing $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ with f_t^θ such that $p_\theta(\mathbf{x}_0|\mathbf{x}_t) = \text{Cat}(\mathbf{x}_0; f_t^\theta(\mathbf{x}_t))$ and letting $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = q(\mathbf{x}_{t-1}|\mathbf{x}_t, f_t^\theta(\mathbf{x}_t))$; and (3) parameterizing $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ with f_t^θ and then marginalizing $q(\mathbf{x}_{t-1}, \mathbf{x}_0|\mathbf{x}_t)$ such that $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \sum_{\mathbf{x}_0} q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)p_\theta(\mathbf{x}_0|\mathbf{x}_t)$.

Method (1) does not reuse any known distribution from the forward process, and hence is less effective in practice. Method (2) has been widely used for continuous diffusion models as in [HJA20] and [SME21], and some discrete diffusion models like in [Hoo+21] and [Zhe+23]. It avoids marginalization and works efficiently and effectively for continuous diffusion. However for discrete diffusion, as shown in Eq. (5.10), sample $\mathbf{x}_0$ determines which categorical distribution should be used, which cannot be determined without the true $\mathbf{x}_0$. Some heuristics have been proposed in [Zhe+23], however those can have a large gap to the true $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$, leading to an inaccurate sampling process.

Method (3) has been proposed in [Aus+21] by directly marginalizing out $\mathbf{x}_0$, which introduces additional computational cost in both loss function computation and sampling process as the formulation of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ has not been simplified to a closed-form distribution. In this paper, as one of our key contributions, we show that method (3) parameterization can be simplified to a clean formulation of categorical distribution. This not only simplifies and accelerates sampling greatly, but also leads to a clean formulation of the negative VLB loss. As before, we work with a more general distribution $p_\theta(\mathbf{x}_s|\mathbf{x}_t)$, $0 < s < t \leq T$ for any s, t.

Proposition 5.2.2. *The parameterization of $p_\theta(\mathbf{x}_s|\mathbf{x}_t)$ can be simplified for any $0 < s < t \leq T$ as*

$$p_\theta(\mathbf{x}_s|\mathbf{x}_t) = Cat\Big(\mathbf{x}_s; (1-\mu_{t|s}) \cdot f_t^\theta(\boldsymbol{x}_t) + (\mu_{t|s}\overline{\alpha}_{t|s} + \gamma_{t|s}^\theta) \cdot \boldsymbol{x}_t + (\mu_{t|s}(1-\overline{\alpha}_{t|s}) - \gamma_{t|s}^\theta) \cdot \boldsymbol{m}\Big) \tag{5.21}$$

where $\gamma_{t|s}^\theta$ is affected by $f_t^\theta(\mathbf{x}_t)$ and is defined as

$$\gamma_{t|s}^\theta = (\mu_{t|s} - \lambda_{t|s} - \mu_{t|s}\overline{\alpha}_{t|s})\langle f_t^\theta(\boldsymbol{x}_t), \boldsymbol{x}_t\rangle\ . \tag{5.22}$$

Notice that $f_t^\theta(\boldsymbol{x}_t)$ is a parameterized neural network with softmax normalization at the last layer such that $\mathbf{1}^T f_t^\theta(\boldsymbol{x}_t) = 1$. As we show next, the above formulation simplifies the negative VLB loss computation greatly (§5.2.5), further motivates an approximated loss that is much easier to optimize (§5.2.6), and accelerates the sampling process through reparameterization (§5.2.7).

Proof. We first provide the formulation as follows.

$$p_\theta(\mathbf{x}_s|\mathbf{x}_t) = \sum_{\mathbf{x}_0} q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0)p_\theta(\mathbf{x}_0|\mathbf{x}_t) \tag{5.23}$$

Using $q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0)$ in Eq. (5.10) and $p_\theta(\mathbf{x}_0|\mathbf{x}_t) = \text{Cat}(\mathbf{x}_0; f_t^\theta(\mathbf{x}_t))$, we can expand it as

$$
\begin{aligned}
p_\theta(\mathbf{x}_s|\mathbf{x}_t) = & q(\mathbf{x}_s|\mathbf{x}_t,\mathbf{x}_t)p_\theta(\mathbf{x}_t|\mathbf{x}_t) + \sum_{\mathbf{x}\neq\mathbf{x}_t} q(\mathbf{x}_s|\mathbf{x}_t,\mathbf{x})p_\theta(\mathbf{x}|\mathbf{x}_t) \\
=& \mathbf{x}_s^\top\Big((1-\lambda_{t|s})\mathbf{x}_t + \lambda_{t|s}\boldsymbol{m}\Big)\mathbf{x}_t^\top f_t^\theta(\mathbf{x}_t) \\
& + \sum_{\mathbf{x}\neq\mathbf{x}_t}\mathbf{x}_s^T\Big((1-\mu_{t|s})\mathbf{x} + \mu_{t|s}\overline{\alpha}_{t|s}\mathbf{x}_t + \mu_{t|s}(1-\overline{\alpha}_{t|s})\boldsymbol{m}\Big)\mathbf{x}^\top f_t^\theta(\mathbf{x}_t) \\
=& \mathbf{x}_s^\top\Big[\big((1-\lambda_{t|s})\mathbf{x}_t + \lambda_{t|s}\boldsymbol{m}\big)\mathbf{x}_t^\top f_t^\theta(\mathbf{x}_t) + (1-\mu_{t|s})\big(\sum_{\mathbf{x}\neq\mathbf{x}_t}\mathbf{x}\mathbf{x}^\top\big)f_t^\theta(\mathbf{x}_t) \\
& + (\mu_{t|s}\overline{\alpha}_{t|s}\mathbf{x}_t + \mu_{t|s}(1-\overline{\alpha}_{t|s})\boldsymbol{m})\big(\sum_{\mathbf{x}\neq\mathbf{x}_t}\mathbf{x}\big)^\top f_t^\theta(\mathbf{x}_t)\Big] \\
=& \mathbf{x}_s^\top\Big[\big((1-\lambda_{t|s})\mathbf{x}_t + \lambda_{t|s}\boldsymbol{m}\big)\mathbf{x}_t^\top f_t^\theta(\mathbf{x}_t) + (1-\mu_{t|s})(I-\mathbf{x}_t\mathbf{x}_t^\top)f_t^\theta(\mathbf{x}_t) \\
& + (\mu_{t|s}\overline{\alpha}_{t|s}\mathbf{x}_t + \mu_{t|s}(1-\overline{\alpha}_{t|s})\boldsymbol{m})(\mathbf{1}-\mathbf{x}_t)^\top f_t^\theta(\mathbf{x}_t)\Big] \\
=& \mathbf{x}_s^\top\Big[(1-\lambda_{t|s})\mathbf{x}_t^\top f_t^\theta(\mathbf{x}_t)\mathbf{x}_t + \lambda_{t|s}\mathbf{x}_t^\top f_t^\theta(\mathbf{x}_t)\boldsymbol{m} + (1-\mu_{t|s})(f_t^\theta(\mathbf{x}_t) - \mathbf{x}_t^\top f_t^\theta(\mathbf{x}_t)\mathbf{x}_t) \\
& + (\mu_{t|s}\overline{\alpha}_{t|s}\mathbf{x}_t + \mu_{t|s}(1-\overline{\alpha}_{t|s})\boldsymbol{m})(1-\mathbf{x}_t^T f_t^\theta(\mathbf{x}_t))\Big] \\
=& \mathbf{x}_s^\top\Big[(1-\mu_{t|s})\cdot f_t^\theta(\mathbf{x}_t) + \Big((\mu_{t|s}-\lambda_{t|s}-\mu_{t|s}\overline{\alpha}_{t|s})\mathbf{x}_t^\top f_t^\theta(\mathbf{x}_t) + \mu_{t|s}\overline{\alpha}_{t|s}\Big)\cdot\mathbf{x}_t \\
& + \Big(-(\mu_{t|s}-\lambda_{t|s}-\mu_{t|s}\overline{\alpha}_{t|s})\mathbf{x}_t^\top f_t^\theta(\mathbf{x}_t) + \mu_{t|s}(1-\overline{\alpha}_{t|s})\Big)\cdot\boldsymbol{m}\Big] \\
= & \ \mathrm{Cat}\Big(\mathbf{x}_s;\ (1-\mu_{t|s})\cdot f_t^\theta(\mathbf{x}_t) + (\mu_{t|s}\overline{\alpha}_{t|s}+\gamma_{t|s}^\theta)\cdot\mathbf{x}_t + (\mu_{t|s}(1-\overline{\alpha}_{t|s})-\gamma_{t|s}^\theta)\cdot\boldsymbol{m}\Big)
\end{aligned} \tag{5.24}
$$

□

5.2.5 Loss Function Derivation

With $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$ in Eq. (5.10) and $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ in Eq. (5.21), the $\mathcal{L}_t(\theta)$ term in Eq. (5.3) can be written as

$$
\begin{aligned}
&\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\Big[\delta_{\mathbf{x}_t,\mathbf{x}_0}D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1}|\mathbf{x}_t=\mathbf{x}_0)\|p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\big) \\
&+(1-\delta_{\mathbf{x}_t,\mathbf{x}_0})D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1}|\mathbf{x}_t\neq\mathbf{x}_0)\|p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\big)\Big],
\end{aligned} \tag{5.25}
$$

where $\delta_{\mathbf{x}_t,\mathbf{x}_0}$ denotes the Kronecker delta of $\mathbf{x}_t$ and $\mathbf{x}_0$. $q(\mathbf{x}_{t-1}|\mathbf{x}_t=\mathbf{x}_0)$ and $q(\mathbf{x}_{t-1}|\mathbf{x}_t\neq\mathbf{x}_0)$ represent the first and second categorical distribution in Eq. (5.10), respectively.

Apart from negative VLB, another commonly employed auxiliary loss is the cross-entropy (CE) loss between $q(\mathbf{x}_t|\mathbf{x}_0)$ and $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$, which measures the reconstruction quality.

$$
\mathcal{L}_t^{CE}(\theta) := \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}[-\log p_\theta(\mathbf{x}_0|\mathbf{x}_t)] \tag{5.26}
$$

Notice that both Eq. (5.26) and the negative VLB in Eq. (5.25) share the same global minima with $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ being the true posterior $q(\mathbf{x}_0|\mathbf{x}_t)$. However they have different optimization landscape; and thus under limited data and network capacity, which loss would be easier to minimize is unknown [TB07; DCO20].

5.2.6 Simplifying Loss Further for Easier Opt.

While Eq. (5.25) is the *exact* negative VLB loss, in practice we find it harder to minimize than $\mathcal{L}_t^{CE}$. In this section, we first derive a much simpler, approximated loss by observing a relation between $q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0)$ and $p_\theta(\mathbf{x}_t|\mathbf{x}_0)$. Recent successes in continuous diffusion models [HJA20; Kar+22] show that the coefficient of each loss term at different time steps should be revised to be invariant to noise scheduling for easier optimization. We show that coefficient simplification and $\mathcal{L}_t^{CE}$ are *both* valuable for optimizing a general negative VLB where only partial time steps are observed. We combine all designs to derive the final approximated loss, denoted as $\tilde{\mathcal{L}}_t$.

Proposition 5.2.3. *For any* $0 < s < t \leq T$*, with* $\mathbf{x}_0$ *known,*

$$\triangle \boldsymbol{p}_\theta(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0) := p_\theta(\mathbf{x}_s|\mathbf{x}_t) - q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0) = \tag{5.27}$$
$$(1 - \mu_{t|s})[f_t^\theta(\boldsymbol{x}_t) - \boldsymbol{x}_0 + \phi_{t|s}\langle f_t^\theta(\boldsymbol{x}_t) - \boldsymbol{x}_0, \boldsymbol{x}_t\rangle(\boldsymbol{x}_t - \boldsymbol{m})] \ ,$$

where

$$\phi_{t|s} = \frac{(1-\bar{\alpha}_s)\bar{\alpha}_{t|s}}{\bar{\alpha}_t + (1-\bar{\alpha}_t)\langle \mathbf{x}_t, \boldsymbol{m}\rangle} \ . \tag{5.28}$$

Proposition 5.2.3 shows that the distribution difference between $p_\theta(\mathbf{x}_s|\mathbf{x}_t)$ and $q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0)$ has a closed-form formulation. (See Appx. of [Zha+24b] for the proof.) With this formulation, we can apply the Taylor expansion (up to second order) to approximate the KL divergence directly

$$D_{\mathrm{KL}}\big(q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0)\|p_\theta(\mathbf{x}_s|\mathbf{x}_t)\big) \approx \sum_{\mathbf{x}_s} \frac{|\triangle \boldsymbol{p}_\theta(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0)|^2}{q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0)} \tag{5.29}$$

The above formulation along with Proposition 3 are valid for *any* $0 < s < t \leq T$, which is much general than the term used in Eq. (5.3) that only considers $s = t - 1$. We next show that minimizing divergence between q and p_θ at any s and t is also valid, as it is inside a general negative VLB with partial time steps. The initial version of the VLB is derived under the assumption that observations are made at every time step. Its backward denoising process is designed to advance by a single time step during each generation step for best generation quality. Let us consider a more general case where only partial time steps are observed in the forward process, then, minimizing its negative VLB can help improve generation quality with fewer steps. Assuming only $\mathbf{x}_s$ and $\mathbf{x}_t$ are observed, where $0 < s < t \leq T$, a derivation analogous to that of Eq. (5.3) can show that

$$\log p_\theta(\mathbf{x}_0) \geq -D_{\mathrm{KL}}[q(\mathbf{x}_t|\mathbf{x}_0)\|p_\theta(\mathbf{x}_t)] + \mathbb{E}_{q(\mathbf{x}_s|\mathbf{x}_0)}[\log p_\theta(\mathbf{x}_0|\mathbf{x}_s)] - D_{\mathrm{KL}}[q(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0)\|p_\theta(\mathbf{x}_s|\mathbf{x}_t)], \tag{5.30}$$

where the first divergence term between the prior and posterior quantifies the quality of the backward denoising process from T to t. The second term represents the CE loss, $\mathcal{L}_s^{\mathrm{CE}}$, which influences the generation quality from time s to time 0. The final term is given by Eq. (5.29), and contributes to the generation process from t to s.

This generalized formulation of the VLB highlights the significance of the CE loss and the alignment between $q(\mathbf{x}_s|\mathbf{x}_s, \mathbf{x}_0)$ and $p_\theta(\mathbf{x}_s|\mathbf{x}_t)$ at any observed times s and t. While CE loss does not have a coefficient that depends on noise schedules and time, changing s and t or noise schedule (which determines $\bar{\alpha}_t$) during training will greatly impact the scale of the term in Eq. (5.29). By rendering the loss scaling term independent of time and the noise schedule, the minimization of this adjusted loss concurrently leads to the minimization of the original loss in Eq. (5.29) for any given s and t. Hence, by removing the sensitive scale $\frac{(1-\mu_{t|s})^2}{q(\mathbf{x}_s|\mathbf{x}_t,\mathbf{x}_0)}$ in Eq. (5.29), we reformulate the loss as

$$\mathcal{L}_t^2 := \|f_t^\theta(\boldsymbol{x}_t) - \boldsymbol{x}_0 + \phi_{t|s}\langle f_t^\theta(\boldsymbol{x}_t) - \boldsymbol{x}_0, \boldsymbol{x}_t\rangle(\boldsymbol{x}_t - \boldsymbol{m})\|_2^2, \tag{5.31}$$

where we can further clip $\phi_{t|s}$ to $\min(1, \phi_{t|s})$ for minimal scaling influence. It is important to note that while we have modified the coefficient to be invariant to the noise schedule and time s to effectively minimize Eq. (5.29) at any t and s, a similar approach to coefficient revision has been previously explored in [HJA20; Kar+22], primarily to facilitate an easier optimization by achieving a balance of terms in the loss function. Overall, we have the final approximated loss

$$\tilde{\mathcal{L}}_t(\theta) = \mathcal{L}_t^2(\theta) + \mathcal{L}_t^{CE}(\theta)\ . \tag{5.32}$$

We find that in practice this loss is much easier to optimize than the original exact negative VLB on harder tasks.

5.2.7 Reparameterization Form for Sampling

In practice, we need to sample $\mathbf{x}_{t|0} \sim q(\mathbf{x}_t|\mathbf{x}_0)$ for training and $\mathbf{x}_{s|t} \sim p_\theta(\mathbf{x}_s|\mathbf{x}_t)$ for generation (backward denoising). In what follows, we provide the reparameterization form for these to facilitate fast sampling. Given $q(\mathbf{x}_t|\mathbf{x}_0) = \mathrm{Cat}(\mathbf{x}_t; \overline{\alpha}_t\mathbf{x}_0 + (1-\overline{\alpha}_t)\boldsymbol{m})$ and $p_\theta(\mathbf{x}_s|\mathbf{x}_t)$ as in Eq. (5.21), we can rewrite the corresponding variables as

$$\mathbf{x}_{t|0} = \delta_{1,\mathbf{b}_t}\mathbf{x}_0 + (1-\delta_{1,\mathbf{b}_t})\mathbf{m}_0,\ \text{ where } \mathbf{b}_t \sim \mathrm{Bernoulli}(\overline{\alpha}_t) \tag{5.33}$$

$$\mathbf{x}_{s|t} = \delta_{1,\mathbf{b}_{s|t}}\tilde{\mathbf{x}}_{0|t}\ +\ \delta_{2,\mathbf{b}_{s|t}}\mathbf{x}_t\ +\ \delta_{3,\mathbf{b}_{s|t}}\mathbf{m}_0, \tag{5.34}$$

where $\tilde{\mathbf{x}}_{0|t} \sim \mathrm{Cat}(f_t^\theta(\mathbf{x}_t))$ and $\mathbf{b}_{s|t} \sim \mathrm{Cat}(\cdot; [1-\mu_{t|s}, \mu_{t|s}\overline{\alpha}_{t|s} + \gamma_{t|s}^\theta,\ \mu_{t|s}(1-\overline{\alpha}_{t|s}) - \gamma_{t|s}^\theta])$.

Eq. (5.33) and Eq. (5.34) essentially show that the sampling process can be divided into two steps: first, sample the branch indicator $\mathbf{b}_t$ (or $\mathbf{b}_{s|t}$), and then sample from the categorical distribution of that branch, i.e. $\mathbf{x}_t$, $\mathbf{m}_0$, or $\tilde{\mathbf{x}}_{0|t}$. Moreover, the three terms in Eq. (5.34) highlight that the denoising step of generating $\mathbf{x}_s$ from $\mathbf{x}_t$ essentially draws samples via three levers: (1) use the predicted sample $\mathbf{x}_0$ from the trained network $f_t^\theta(\mathbf{x}_t)$ directly, (2) keep it unchanged as $\mathbf{x}_t$, or (3) roll it back to noise $\mathbf{m}_0$, offering an intuitive understanding.

5.3 Continuous-time Discrete Diffusion

Despite being simple, discrete-time diffusion limits the generation process as we can only "jump back" through fixed time points. Recent works generalize continuous-state diffusion models to continuous-time [Son+21].. This generalization enables great flexibility in backward generation as one can "jump back" through any time in $[0, T]$ to the target distribution, often with improved sample quality.

Nevertheless, generalizing discrete-state diffusion model from discrete-time to continuous-time is nontrivial, as the score-matching based technique [Son+21] in continuous-state models requires the score function $\nabla_\mathbf{x} \log p_t(\mathbf{x})$ to be available. This function, however, is evidently non-existent for discrete distributions. Recently, Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22] presented the first continuous-time diffusion model for discrete data. It formulates the forward process through a Continuous Time Markov Chain (CTMC) and aims at learning a reverse CTMC that matches the marginal distribution with the forward CTMC at any time t. While being theoretically solid, the formulation in [Cam+22] has two problems: **(1)** the negative VLB loss of matching the forward and backward CTMCs is analytically complicated and hard to implement; and **(2)** the exact sampling through the learned backward CTMC is unrealistic. Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22] propose approximate solutions, which however, trade off computational tractability with unknown errors and sample quality. SDDM [Sun+23] takes a different approach with ratio matching [Hyv07; Lyu09], which is the generalization of score matching to discrete data. However, it needs a specific network architecture and is not applicable to other models.

In this paper, we build on the CTMC formulation in Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22], and show that the loss can be analytically simplified for nominal data. This simplification also inspires an improved MCMC corrector with closed-form formulation. For problem **(2)**, we argue that the difficulty arises from using the learned transition rate matrix for the backward CTMC, where the sampling probability $p_\theta(\mathbf{x}_s|\mathbf{x}_t)$ is hard to compute using the transition rate matrix. Instead, capitalizing on the realization that this reverse transition rate matrix is computed based on the learned $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$, we propose to compute $p_\theta(\mathbf{x}_s|\mathbf{x}_t)$ through $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ without using the transition rate matrix. This avoids the approximation error and greatly simplifies the generation process.

Remarkably, with the new formulation of generation, we show that the continuous-time and discrete-time diffusion models can be unified together, with exactly the same forward diffusion and now also backward generation process. Moreover, we demonstrate that this unification offers mutual benefits: the continuous-time diffusion can leverage the swift and precise sampling formulation derived from the discrete-time case (as detailed in §5.2.7), while the discrete-time diffusion can utilize the MCMC corrector from the continuous-time scenario.

5.3.1 Background: Continuous-Time Markov Chain

CTMC generalizes Markov chain from discrete- to continuous-time via the Markov property: $\mathbf{x}_{t_1} \perp\!\!\!\perp \mathbf{x}_{t_3} \mid \mathbf{x}_{t_2}, \forall t_1 < t_2 < t_3$. Anderson [And12] provides an introduction to time-homogeneous CTMC. It can be derived from discrete-time Markov chain by increasing the number of time stamps N to infinite while keeping the total time T fixed. Specifically, we can define $\triangle t = \frac{T}{N}$, $t_i = i\triangle t$, and a discrete-time Markov chain characterized by transition probability $q(\mathbf{x}_{t_i}|\mathbf{x}_{t_{i-1}})$ and transition matrix Q_{t_i} with $[Q_{t_i}]_{jk} = q(\mathbf{x}_{t_i} = \boldsymbol{e}_k|\mathbf{x}_{t_{i-1}} = \boldsymbol{e}_j)$. By setting N to infinite, the transition probability $q(\mathbf{x}_{t_i}|\mathbf{x}_{t_{i-1}})$ converges to 0, hence is not suitable for describing CTMC. Instead, CTMC is fully characterized by its *transition rate* $r_t(\boldsymbol{y}|\boldsymbol{x})$, s.t.

$$r_t(\boldsymbol{y}|\boldsymbol{x}) = \lim_{\triangle t \to 0} \frac{q_{t|t-\triangle t}(\boldsymbol{y}|\boldsymbol{x}) - \delta_{\boldsymbol{x},\boldsymbol{y}}}{\triangle t} . \tag{5.35}$$

As the name suggests, $r_t(\boldsymbol{y}|\boldsymbol{x})$ measures the change rate of the transition probability of moving from state $\boldsymbol{x}$ to state $\boldsymbol{y}$ at time t in the direction of the process. The corresponding *transition rate matrix* R_t with $[R_t]_{ij} = r_t(\boldsymbol{e}_j|\boldsymbol{e}_i)$ fully determines the underlying stochastic process. A CTMC's transition probabilities satisfy the Kolmogorov equations [Kol31], which have unique solution. As Rindos, Woolet, Viniotis, and Trivedi [Rin+95] stated, when R_{t_1} and R_{t_2} commute (i.e. $R_{t_1}R_{t_2} = R_{t_2}R_{t_1}$) for any t_1, t_2, the transition probability matrix can be written as

$$\overline{Q}_{t|s} = \exp\left(\int_s^t R_a da\right) , \tag{5.36}$$

where $\exp(M) := \sum_{k=0}^{\infty} \frac{M^k}{k!}$. The commutative property of R_t can be achieved by choosing $R_t = \beta(t)R_b$ where $R_b \in \mathbb{R}^{K\times K}$ is a time-independent base rate matrix.

5.3.2 Forward and Backward CTMCs

Forward CTMC. Two properties are needed for modeling the forward process of adding noise with a CTMC: P1) the process can converge to an easy-to-sample stationary distribution at final time T; and P2) the conditional marginal distribution

$q(\mathbf{x}_t|\mathbf{x}_0)$ can be obtained analytically for efficient training. As given in Eq. (5.36), P2) can be achieved by choosing commutative transition rate matrices with $R_t = \beta(t)R_b$.

We next show that property P1), i.e. $\lim_{t\to T} \overline{Q}_{t|0} = \overline{Q}_{T|0} = \mathbf{1}\boldsymbol{m}^\top$ for some stationary distribution $\boldsymbol{m}$, can be achieved by choosing $R_b = \mathbf{1}\boldsymbol{m}^\top - I$, which is a valid transition rate matrix with the property $(-R_b)^2 = (-R_b)$.

Proof.

$$\begin{aligned}\overline{Q}_{t|s} = \exp(\overline{\beta}_{t|s}R_b) = I + \sum_{k=1}^{\infty}\frac{(-\overline{\beta}_{t|s})^k(-R_b)^k}{k!} = I - (\sum_{k=1}^{\infty}\frac{(-\overline{\beta}_{t|s})^k}{k!})R_b \\ = I - (e^{-\overline{\beta}_{t|s}} - 1)R_b = e^{-\overline{\beta}_{t|s}}I + (1 - e^{-\overline{\beta}_{t|s}})\mathbf{1}\boldsymbol{m}^\top \end{aligned} \tag{5.37}$$

□

Then we have

$$\overline{Q}_{t|s} = \exp(\overline{\beta}_{t|s}R_b) = e^{-\overline{\beta}_{t|s}}I + (1 - e^{-\overline{\beta}_{t|s}})\mathbf{1}\boldsymbol{m}^\top. \tag{5.38}$$

★ **Unified forward process.** Eq. (5.38) will have exactly the same formulation as the transition matrix of the discrete-time case in Eq. (5.9), if we set $\overline{\alpha}_{t|s} = \exp(-\overline{\beta}_{t|s}) = \exp(-\int_s^t \beta(a)da)$. Thus, *this formulation unifies the forward processes of adding noise for both discrete- and continuous-time discrete diffusion.* With $\lim_{t\to T}\overline{\alpha}_{t|0} = 0$, or equivalently $\lim_{t\to T}\int_0^T \beta(a)da = \infty$, we achieve the goal $\overline{Q}_{T|0} = \mathbf{1}\boldsymbol{m}^\top$. We use $\overline{\alpha}_{t|s}$ directly in the following sections, i.e. Eq. (5.9). To summarize, for the forward CTMC

$$\begin{aligned} R_t &= \beta(t)(\mathbf{1}\boldsymbol{m}^\top - I)\text{ , and} \\ \overline{Q}_{t|s} &= \overline{\alpha}_{t|s}I + (1 - \overline{\alpha}_{t|s})\mathbf{1}\boldsymbol{m}^\top \ . \end{aligned} \tag{5.39}$$

We can further get vector-form forward rate

$$\begin{aligned} r_t(\boldsymbol{x}|\cdot) &= R_t\boldsymbol{x} = \beta(t)\big(\langle \boldsymbol{x}, \boldsymbol{m}\rangle\mathbf{1} - \boldsymbol{x}\big), \\ r_t(\cdot|\boldsymbol{x}) &= R_t^\top\boldsymbol{x} = \beta(t)\big(\boldsymbol{m} - \boldsymbol{x}\big) \ . \end{aligned} \tag{5.40}$$

Backward CTMC. To generate samples from the target distribution, we have to reverse the process of the forward CTMC. Let $\widehat{r}_t$ be the transition rate of the backward CTMC with corresponding matrix $\widehat{R}_t$. When the forward and backward CTMCs are matched exactly, theoretically the forward and backward CTMCs have the following relationship (see Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22]'s Proposition 1):

$$\widehat{r}_t(\boldsymbol{x}|\boldsymbol{y}) = r_t(\boldsymbol{y}|\boldsymbol{x})\frac{q_t(\boldsymbol{x})}{q_t(\boldsymbol{y})}, \quad \forall \boldsymbol{x} \neq \boldsymbol{y} \ . \tag{5.41}$$

However, the marginal distributions $q_t(\boldsymbol{x})$ and $q_t(\boldsymbol{y})$ are intractable analytically, hence we cannot derive the backward CTMC directly from Eq. (5.41). Instead, Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22] parameterize the transition rate $\widehat{r}_t^\theta$, by observing that $\frac{q_t(\boldsymbol{x})}{q_t(\boldsymbol{y})} = \sum_{\boldsymbol{x}_0}\frac{q_{t|0}(\boldsymbol{x}|\boldsymbol{x}_0)}{q_{t|0}(\boldsymbol{y}|\boldsymbol{x}_0)}q_{0|t}(\boldsymbol{x}_0|\boldsymbol{y})$[2], as follows

$$\widehat{r}_t^\theta(\boldsymbol{x}|\boldsymbol{y}) = r_t(\boldsymbol{y}|\boldsymbol{x})\sum_{\boldsymbol{x}_0}\frac{q_{t|0}(\boldsymbol{x}|\boldsymbol{x}_0)}{q_{t|0}(\boldsymbol{y}|\boldsymbol{x}_0)}p_{0|t}^\theta(\boldsymbol{x}_0|\boldsymbol{y}). \tag{5.42}$$

Then, $\widehat{r}_t^\theta$ is obtained by learning the parameters θ to minimize the continuous-time negative VLB introduced next.

[2]As $q_t(\boldsymbol{x}) = \sum_{\boldsymbol{x}_0} q_{t|0}(\boldsymbol{x}|\boldsymbol{x}_0)q_{\text{data}}(\boldsymbol{x}_0)$ and $q_t(\boldsymbol{x}) = \frac{q_{t|0}(\boldsymbol{x}|\tilde{\boldsymbol{x}}_0)q_{\text{data}}(\tilde{\boldsymbol{x}}_0)}{q_{0|t}(\tilde{\boldsymbol{x}}_0|\boldsymbol{x})}$.

Negative VLB. Similar to the discrete-time case, the backward CTMC can be learned by maximizing the VLB for data log-likelihood. Computing VLB for CTMC is nontrivial, and fortunately Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22] has derived (see their Proposition 2) that the negative VLB can be formulated as

$$T\ \mathbb{E}_{\substack{t\sim \text{Uni}(0,T)\\ \boldsymbol{x}\sim q(\mathbf{x}_t|\mathbf{x}_0)}}\Big[\sum_{\boldsymbol{z}\neq\boldsymbol{x}}\widehat{r}^{\theta}_t(\boldsymbol{z}|\boldsymbol{x}) - \sum_{\boldsymbol{z}\neq\boldsymbol{x}} r_t(\boldsymbol{z}|\boldsymbol{x})\log \widehat{r}^{\theta}_t(\boldsymbol{x}|\boldsymbol{z})\Big]\,. \tag{5.43}$$

However, Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22]'s original design did not simplify the negative VLB with the parameterization of $\widehat{r}^{\theta}_t$ in Eq. (5.42), making the implementation nontrivial and inefficient. In this section, we show that their formulation can be greatly simplified to a closed-form evaluation.

Before the simplification of Eq. (5.43), we introduce $g^{\theta}_t(\boldsymbol{x}|\boldsymbol{y})$ such that $\widehat{r}^{\theta}_t(\boldsymbol{x}|\boldsymbol{y}) = r_t(\boldsymbol{y}|\boldsymbol{x})g^{\theta}_t(\boldsymbol{x}|\boldsymbol{y})$, with

$$g^{\theta}_t(\boldsymbol{x}|\boldsymbol{y}) := \sum_{\boldsymbol{x}_0}\frac{q_{t|0}(\boldsymbol{x}|\boldsymbol{x}_0)}{q_{t|0}(\boldsymbol{y}|\boldsymbol{x}_0)}p^{\theta}_{0|t}(\boldsymbol{x}_0|\boldsymbol{y}) \approx \frac{q_t(\boldsymbol{x})}{q_t(\boldsymbol{y})}\,, \tag{5.44}$$

which is the estimator of the marginal probability ratio.

Proposition 5.3.1. *The vector form parameterization of $g^{\theta}_t(\boldsymbol{x}|\boldsymbol{y})$ can be simplified analytically as:*

$$g^{\theta}_t(\cdot|\boldsymbol{y}) = \Big[(1-\frac{\overline{\alpha}_{t|0}\langle f^{\theta}_t(\boldsymbol{y}),\boldsymbol{y}\rangle}{\overline{\alpha}_{t|0}+(1-\overline{\alpha}_{t|0})\langle \boldsymbol{y},\boldsymbol{m}\rangle})\boldsymbol{m} + \frac{\overline{\alpha}_{t|0}}{1-\overline{\alpha}_{t|0}}f^{\theta}_t(\boldsymbol{y})\Big]\odot\frac{\mathbf{1}-\boldsymbol{y}}{\langle \boldsymbol{y},\boldsymbol{m}\rangle}+\boldsymbol{y} \tag{5.45}$$

The proof and its extension to multi-element case $g^{\theta,d}_t(\cdot|\boldsymbol{y}^{1:D})$ can be found in Appendix of [Zha+24b].

5.3.3 Simplification of Continuous-time Negative VLB

As the derivation is much harder in multi-element case, we work on it directly and single-element can be induced as a special case. Given $\boldsymbol{x}^{1:D}$, let $\boldsymbol{x}^{\setminus d}$ represent $\boldsymbol{x}^{1:D\setminus d}$, i.e. the object without d-th element. Before diving into the loss, we first need to generalize the definition of transition rate of forward and backward CTMC to multi-element case. We present the result below.

$$r^{1:D}_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D}) = \sum_{d=1}^{D} r^d_t(\boldsymbol{z}^d|\boldsymbol{x}^d)\delta_{\boldsymbol{x}^{\setminus d},\boldsymbol{z}^{\setminus d}}$$

$$\widehat{r}^{\theta,1:D}_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D}) = \sum_{d=1}^{D} r^d_t(\boldsymbol{x}^d|\boldsymbol{z}^d)g^{\theta,d}_t(\boldsymbol{z}^d|\boldsymbol{x}^{1:D})\cdot\delta_{\boldsymbol{x}^{\setminus d},\boldsymbol{z}^{\setminus d}}$$

where $\delta_{\boldsymbol{x},\boldsymbol{y}}$ is the Kronecker delta, $r^{1:D}_t$ is the forward transition rate and $\widehat{r}^{\theta,1:D}_t$ is the backward transition rate parameterization. As we assume the forward processes are independent for different elements, r^d_t represents the transition rate of the forward CTMC process at the d-th element.

Proof. In this section, we show how to extend transition rates, and and the ratio g^{θ}_t, into multi-element case. We let $\boldsymbol{x}^{\setminus d}$ represent $\boldsymbol{x}^{1:D\setminus d}$, i.e. the object without d-th element, for simplicity.

Forward Transition Rates: First, the transition rates for forward sampling has a specific decomposition formulation in multi-element case as proven by [Cam+22], thus, we summarize the result as follows. The key assumption for CTMC is that at a single time, only one dimension can change.

$$r_t^{1:D}(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D}) = \sum_{d=1}^{D} r_t^d(\boldsymbol{y}^d|\boldsymbol{x}^d)\delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}} \tag{5.46}$$

where $\delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}}$ is the Kronecker delta and it is 1 if and only if $\boldsymbol{x}^{\backslash d} = \boldsymbol{y}^{\backslash d}$. As we also assume that all dimension processes are indepedent, r_t^d denotes the transition rate of the CTMC process at d-th element/dimension.

Backward Transition Rates: Now let us work on $\hat{r}_t^{1:D}(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D})$. Notice that as the backward process is also a CTMC, it also satisfies that only one dimension can change at a time. We summarize two equivalent formulations as follows.

$$\hat{r}_t^{1:D}(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D}) = \sum_{d=1}^{D} r_t^d(\boldsymbol{x}^d|\boldsymbol{y}^d)\delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}} \sum_{\boldsymbol{x}_0^d} \frac{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)} q_{0|t}(\boldsymbol{x}_0^d|\boldsymbol{x}^{1:D}) \tag{5.47}$$

$$\hat{r}_t^{1:D}(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D}) = \sum_{d=1}^{D} \frac{r_t^d(\boldsymbol{x}^d|\boldsymbol{y}^d)\delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}}}{\sum_{\boldsymbol{x}_0^d} \frac{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)} q_{0|t}(\boldsymbol{x}_0^d|\boldsymbol{y}^{1:D})} \tag{5.48}$$

Notice that these two formulations should be equivalent. In practice, we use the first formulation to parameterize the reverse transition rate in learning.

Proof.

$$\frac{q_t(\boldsymbol{y}^{1:D})}{q_t(\boldsymbol{x}^{1:D})} = \sum_{\boldsymbol{x}_0^{1:D}} \frac{q_{t|0}(\boldsymbol{y}^{1:D}|\boldsymbol{x}_0^{1:D})}{q_{t|0}(\boldsymbol{x}^{1:D}|\boldsymbol{x}_0^{1:D})} q_{0|t}(\boldsymbol{x}_0^{1:D}|\boldsymbol{x}^{1:D}) = \sum_{\boldsymbol{x}_0^{1:D}} q_{0|t}(\boldsymbol{x}_0^{1:D}|\boldsymbol{x}^{1:D}) \prod_{d=1}^{D} \frac{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)} \tag{5.49}$$

$$\sum_{\boldsymbol{x}_0^{\backslash d}} q_{0|t}(\boldsymbol{x}_0^{1:D}|\boldsymbol{x}^{1:D}) = \sum_{\boldsymbol{x}_0^{\backslash d}} q_{0|t}(\boldsymbol{x}_0^d|\boldsymbol{x}^{1:D}) q_{0|t}(\boldsymbol{x}_0^{\backslash d}|\boldsymbol{x}^{1:D}, \boldsymbol{x}_0^d) = q_{0|t}(\boldsymbol{x}_0^d|\boldsymbol{x}^{1:D}) \tag{5.50}$$

(Case 1)

$$\begin{aligned}
\hat{r}_t^{1:D}(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D}) &= r_t^{1:D}(\boldsymbol{x}^{1:D}|\boldsymbol{y}^{1:D}) \frac{q_t(\boldsymbol{y}^{1:D})}{q_t(\boldsymbol{x}^{1:D})} \\
&= \Big(\sum_{d=1}^{D} r_t^d(\boldsymbol{x}^d|\boldsymbol{y}^d)\delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}}\Big)\Big(\sum_{\boldsymbol{x}_0^{1:D}} q_{0|t}(\boldsymbol{x}_0^{1:D}|\boldsymbol{x}^{1:D}) \prod_{d=1}^{D} \frac{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)}\Big) \\
&= \sum_{d=1}^{D} \sum_{\boldsymbol{x}_0^{1:D}} r_t^d(\boldsymbol{x}^d|\boldsymbol{y}^d)\delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}} q_{0|t}(\boldsymbol{x}_0^{1:D}|\boldsymbol{x}^{1:D}) \frac{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)} \\
&= \sum_{d=1}^{D} r_t^d(\boldsymbol{x}^d|\boldsymbol{y}^d)\delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}} \sum_{\boldsymbol{x}_0^d} \frac{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)} \sum_{\boldsymbol{x}_0^{\backslash d}} q_{0|t}(\boldsymbol{x}_0^{1:D}|\boldsymbol{x}^{1:D}) \\
&= \sum_{d=1}^{D} r_t^d(\boldsymbol{x}^d|\boldsymbol{y}^d)\delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}} \sum_{\boldsymbol{x}_0^d} \frac{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)} q_{0|t}(\boldsymbol{x}_0^d|\boldsymbol{x}^{1:D})
\end{aligned} \tag{5.51}$$

(Case 2)

$$
\begin{aligned}
\hat{r}_t^{1:D}(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D}) &= r_t^{1:D}(\boldsymbol{x}^{1:D}|\boldsymbol{y}^{1:D}) / \frac{q_t(\boldsymbol{x}^{1:D})}{q_t(\boldsymbol{y}^{1:D})} \\
&= \sum_{d=1}^{D} \frac{r_t^d(\boldsymbol{x}^d|\boldsymbol{y}^d)\delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}}}{\sum_{\boldsymbol{x}_0^{1:D}} q_{0|t}(\boldsymbol{x}_0^{1:D}|\boldsymbol{y}^{1:D}) \prod_{i=1}^{D} \frac{q_{t|0}(\boldsymbol{x}^i|\boldsymbol{x}_0^i)}{q_{t|0}(\boldsymbol{y}^i|\boldsymbol{x}_0^i)}} \\
&= \sum_{d=1}^{D} \frac{r_t^d(\boldsymbol{x}^d|\boldsymbol{y}^d)\delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}}}{\sum_{\boldsymbol{x}_0^{1:D}} q_{0|t}(\boldsymbol{x}_0^{1:D}|\boldsymbol{y}^{1:D}) \frac{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)}} \\
&= \sum_{d=1}^{D} \frac{r_t^d(\boldsymbol{x}^d|\boldsymbol{y}^d)\delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}}}{\sum_{\boldsymbol{x}_0^d} \frac{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)} q_{0|t}(\boldsymbol{x}_0^d|\boldsymbol{y}^{1:D})}
\end{aligned} \tag{5.52}
$$

□

Ratio: We now define an estimator $g_t^{\theta,d}(\boldsymbol{x}^d|\boldsymbol{y}^{1:D})$ as follows.

$$
g_t^{\theta,d}(\boldsymbol{x}^d|\boldsymbol{y}^{1:D}) := \sum_{\boldsymbol{x}_0^d} \frac{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)} p_{0|t}^{\theta}(\boldsymbol{x}_0^d|\boldsymbol{y}^{1:D}) \approx \sum_{\boldsymbol{x}_0^d} \frac{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)} q_{0|t}(\boldsymbol{x}_0^d|\boldsymbol{y}^{1:D}) = \frac{q_t(\boldsymbol{x}^d|\boldsymbol{y}^{\backslash d})}{q_t(\boldsymbol{y}^d|\boldsymbol{y}^{\backslash d})} \tag{5.53}
$$

$$
g_t^{\theta}(\boldsymbol{x}^{1:D}|\boldsymbol{y}^{1:D}) := \prod_{d=1}^{D} g_t^{\theta,d}(\boldsymbol{x}^d|\boldsymbol{y}^{1:D}) = \sum_{\boldsymbol{x}_0^{1:D}} \frac{q_{t|0}(\boldsymbol{x}^{1:D}|\boldsymbol{x}_0^{1:D})}{q_{t|0}(\boldsymbol{y}^{1:D}|\boldsymbol{x}_0^{1:D})} p_{0|t}^{\theta}(\boldsymbol{x}_0^{1:D}|\boldsymbol{y}^{1:D}) \approx \frac{q_t(\boldsymbol{x}^{1:D})}{q_t(\boldsymbol{y}^{1:D})} \tag{5.54}
$$

We can extend the vector formulation Eq. (5.45) in Proposition 4 to $g_t^{\theta,d}(\boldsymbol{x}^d|\boldsymbol{y}^{1:D})$:

$$
g_t^{\theta,d}(\cdot|\boldsymbol{x}^{1:D}) = \frac{1}{\langle \boldsymbol{x}^d, \boldsymbol{m}^d \rangle} \Big[\big(1 - \frac{\overline{\alpha}_{t|0} \langle f_t^{\theta,d}(\boldsymbol{x}^{1:D}), \boldsymbol{x}^d \rangle}{\overline{\alpha}_{t|0} + (1 - \overline{\alpha}_{t|0}) \langle \boldsymbol{x}^d, \boldsymbol{m}^d \rangle} \big) \boldsymbol{m}^d + \frac{\overline{\alpha}_{t|0}}{1 - \overline{\alpha}_{t|0}} f_t^{\theta,d}(\boldsymbol{x}^{1:D}) \Big] \odot (\mathbf{1} - \boldsymbol{x}^d) + \boldsymbol{x}^d \tag{5.55}
$$

Then, we can derive two approximators for transition rate $\hat{r}_t^{1:D}(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D})$ as follows.

$$
[\hat{r}_t^{\theta,1:D}]^1(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D}) := \sum_{d=1}^{D} r_t^d(\boldsymbol{x}^d|\boldsymbol{y}^d) g_t^{\theta,d}(\boldsymbol{y}^d|\boldsymbol{x}^{1:D}) \cdot \delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}} \approx \text{Eq. (5.47)} \tag{5.56}
$$

$$
[\hat{r}_t^{\theta,1:D}]^2(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D}) := \sum_{d=1}^{D} \frac{r_t^d(\boldsymbol{x}^d|\boldsymbol{y}^d)}{g_t^{\theta,d}(\boldsymbol{x}^d|\boldsymbol{y}^{1:D})} \cdot \delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}} \approx \text{Eq. (5.48)} \tag{5.57}
$$

□

Proposition 5.3.2. *The negative VLB in Eq. (5.43) in multi-element case can be simplified as*

$$
T \mathop{\mathbb{E}}_{\substack{t \sim Uni(0,T) \\ \boldsymbol{x}^{1:D} \sim q_{t|0}(\boldsymbol{x}_0^{1:D}) \\ \boldsymbol{z}^{1:D} \sim S_t(\boldsymbol{x}^{1:D})}} \Big[\sum_{d=1}^{D} r_t^d(\boldsymbol{x}^d|\cdot)^\top g_t^{\theta,d}(\cdot|\boldsymbol{x}^{1:D}) - \frac{1}{\mathcal{M}_{S_t}(\boldsymbol{z}^{1:D}|\boldsymbol{x}_0^{1:D})} \sum_{d=1}^{D} \frac{\mathbf{1}^\top [q_{t|0}(\cdot|\boldsymbol{x}_0^d) \odot r_t^d(\boldsymbol{z}^d|\cdot) \odot \log g_t^{\theta,d}(\cdot|\boldsymbol{z}^{1:D})]}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)} \Big] \tag{5.58}
$$

where $S_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D})$ is any unnormalized distribution (see Eq. (5.67)) of sampling the auxiliary variable $\boldsymbol{z}^{1:D}$ from $\boldsymbol{x}^{1:D}$. The auxiliary variable is introduced to avoid multiple passes of the model for computing the second term of Eq. (5.43). M_{S_t} is a normalization scalar (see Eq. (5.81)) that only depends on $S_t, \boldsymbol{z}^{1:D}$, and $\boldsymbol{x}_0^{1:D}$.

Proof. As forward process is defined in Eq. (5.39), in multi-element case we can easily get

$$r_t^d(\boldsymbol{x}^d|\cdot) = R_t^d \boldsymbol{x}^d = \beta(t)\big(\langle \boldsymbol{x}^d, \boldsymbol{m}^d\rangle \mathbf{1} - \boldsymbol{x}^d\big) \tag{5.59}$$

$$r_t^d(\cdot|\boldsymbol{x}^d) = (R_t^d)^\top \boldsymbol{x}^d = \beta(t)\big(\boldsymbol{m}^d - \boldsymbol{x}^d\big) \tag{5.60}$$

The $r_t^d(\boldsymbol{x}|\cdot)$ and $r_t^d(\cdot|\boldsymbol{x})$ are essentially the $\boldsymbol{x}$-th column and row of the transition rate matrix R_t^d.

In Mult-element case, the negative VLB loss in Eq. (5.43) can be written as

$$T\ \mathbb{E}_{\substack{t\sim \text{Uni}(0,T)\\ \boldsymbol{x}^{1:D}\sim q_{t|0}}}\Big[\sum_{\boldsymbol{z}^{1:D}\neq \boldsymbol{x}^{1:D}} \hat{r}_t^{\theta,1:D}(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D}) - \sum_{\boldsymbol{z}^{1:D}\neq \boldsymbol{x}^{1:D}} r_t^{1:D}(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D}) \log \hat{r}_t^{\theta,1:D}(\boldsymbol{x}^{1:D}|\boldsymbol{z}^{1:D})\Big] \tag{5.61}$$

As there are two terms, let's work on each term separately.

Term 1

Based on the formulation of $r_t^{1:D}$ and $\hat{r}_t^{\theta,1:D}$, we can rewrite the first term as

$$\begin{aligned}
\text{Term1} &= T\ \mathbb{E}_{t,\boldsymbol{x}^{1:D}} \sum_{\boldsymbol{z}^{1:D}\neq \boldsymbol{x}^{1:D}} \hat{r}_t^{\theta,1:D}(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D}) && (5.62)\\
&= T\ \mathbb{E}_{t,\boldsymbol{x}^{1:D}} \sum_{\boldsymbol{z}^{1:D}\neq \boldsymbol{x}^{1:D}} \sum_{d=1}^{D} r_t^d(\boldsymbol{x}^d|\boldsymbol{z}^d) g_t^{\theta,d}(\boldsymbol{z}^d|\boldsymbol{x}^{1:D}) \cdot \delta_{\boldsymbol{x}^{\setminus d},\boldsymbol{z}^{\setminus d}} && (5.63)\\
&= T\ \mathbb{E}_{t,\boldsymbol{x}^{1:D}} \Big[\sum_{d=1}^{D} \sum_{\boldsymbol{z}^d\neq \boldsymbol{x}^d} r_t^d(\boldsymbol{x}^d|\boldsymbol{z}^d) g_t^{\theta,d}(\boldsymbol{z}^d|\boldsymbol{x}^{1:D})\Big] && (5.64)\\
&= T\ \mathbb{E}_{t,\boldsymbol{x}^{1:D}} \Big[\sum_{d=1}^{D} r_t^d(\boldsymbol{x}^d|\cdot)^\top g_t^{\theta,d}(\cdot|\boldsymbol{x}^{1:D})\Big] + \text{const.} && (5.65)\\
&= T\ \mathbb{E}_{\substack{t\sim \text{Uni.}(0,T)\\ \boldsymbol{x}^{1:D}\sim q_{t|0}}} \beta(t) \sum_{d=1}^{D} \langle \boldsymbol{x}^d, \boldsymbol{m}^d\rangle \Big[\mathbf{1}^\top g_t^{\theta,d}(\cdot|\boldsymbol{x}^{1:D})\Big] + \text{const.} && (5.66)
\end{aligned}$$

Term 2

As the evaluation of $\hat{r}_t^\theta(\cdot|\boldsymbol{x})$ for any $\boldsymbol{x}$ requires a single forward pass of the parameterized network $p_{0|t}^\theta(\cdot|\boldsymbol{x})$, the second term within the expectation of Eq. (5.43) requires multiple passes of the model. This complexity is even greatly amplified in cases with multi-element objects. Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22] avoids the multiple passes by changing the expectation variable through importance sampling. We take a similar approach to simplify the second term. Differently, Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22] uses a specific sampling distribution (same as the forward transition rate) to introduce the auxiliary variable for changing the expectation variable, we generalize it to use a general sampling process S_t defined below.

Let $\boldsymbol{y}^{1:D}$ be the new variable upon which the exchanged expectation is based, and assume that $\boldsymbol{y}^{1:D}$ is sampled from an unnormalized joint distribution $S_t(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D})$. We restrict $S_t(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D})$ to be a unnormalized probability that is nonzero if and only if $\boldsymbol{y}^{1:D}$ and $\boldsymbol{x}^{1:D}$ are different at a single element. Formally, we can write the unnormalized distribution as

$$S_t(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D}) = (1 - \delta_{\boldsymbol{y}^{1:D},\boldsymbol{x}^{1:D}}) \sum_{d=1}^{D} S_t^d(\boldsymbol{y}^d|\boldsymbol{x}^d)\delta_{\boldsymbol{y}^{\backslash d},\boldsymbol{x}^{\backslash d}}$$

$$\text{with normalizer } \mathcal{S}_t(\boldsymbol{x}^{1:D}) = \sum_{\boldsymbol{y}^{1:D}} S_t(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D}) = \sum_{d=1}^{D} \sum_{\boldsymbol{y}^d \neq \boldsymbol{x}^d} S_t^d(\boldsymbol{y}^d|\boldsymbol{x}^d) \tag{5.67}$$

where $S_t^d(\boldsymbol{y}^d|\boldsymbol{x}^d)$ is any unnormalized probability at dimension d.

Now for the second term, we have

$$T\ \mathbb{E}_{t,\boldsymbol{x}^{1:D}} \sum_{\boldsymbol{z}^{1:D} \neq \boldsymbol{x}^{1:D}} r_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D}) \log \hat{r}_t^{\theta}(\boldsymbol{x}^{1:D}|\boldsymbol{z}^{1:D}) \tag{5.68}$$

$$= T\ \mathbb{E}_{t,\boldsymbol{x}^{1:D}} \sum_{\boldsymbol{z}^{1:D} \neq \boldsymbol{x}^{1:D}} \frac{S_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D})}{\mathcal{S}(\boldsymbol{x}^{1:D})} \cdot \frac{\mathcal{S}_t(\boldsymbol{x}^{1:D})}{S_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D})} \cdot r_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D}) \log \hat{r}_t^{\theta}(\boldsymbol{x}^{1:D}|\boldsymbol{z}^{1:D}) \tag{5.69}$$

$$= T\ \mathbb{E}_{\substack{t,\boldsymbol{x}^{1:D} \\ \boldsymbol{z}^{1:D} \sim S_t}} \left[\frac{\mathcal{S}_t(\boldsymbol{x}^{1:D})}{S_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D})} \cdot r_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D}) \log \hat{r}_t^{\theta}(\boldsymbol{x}^{1:D}|\boldsymbol{z}^{1:D}) \right] \qquad (\text{As } S_t \text{ samples } \boldsymbol{z}^{1:D} \neq \boldsymbol{x}^{1:D}\ ) \tag{5.70}$$

$$= T\ \mathbb{E}_{t,\boldsymbol{z}^{1:D} \sim S_t} \sum_{\boldsymbol{x}^{1:D}} \left[q_{S_t}(\boldsymbol{x}^{1:D}|\boldsymbol{x}_0^{1:D}, \boldsymbol{z}^{1:D}) \cdot \frac{\mathcal{S}_t(\boldsymbol{x}^{1:D})}{S_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D})} \cdot r_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D}) \log \hat{r}_t^{\theta}(\boldsymbol{x}^{1:D}|\boldsymbol{z}^{1:D}) \right] \tag{5.71}$$

where $q_{S_t}(\boldsymbol{x}^{1:D}|\boldsymbol{x}_0^{1:D}, \boldsymbol{z}^{1:D})$ is the conditional posterior distribution such that (for clearity we replace $\boldsymbol{x}^{1:D}, \boldsymbol{z}^{1:D}$ with $\boldsymbol{x}_t^{1:D}, \boldsymbol{z}_t^{1:D}$ respectively, as they are variables at time t)

$$q_{S_t}(\boldsymbol{x}_t^{1:D}|\boldsymbol{x}_0^{1:D}, \boldsymbol{z}_t^{1:D}) = \frac{q_{S_t}(\boldsymbol{x}_t^{1:D}, \boldsymbol{z}_t^{1:D}|\boldsymbol{x}_0^{1:D})}{\sum_{\boldsymbol{y}_t^{1:D}} q_{S_t}(\boldsymbol{y}_t^{1:D}, \boldsymbol{z}_t^{1:D}|\boldsymbol{x}_0^{1:D})} \tag{5.72}$$

$$= \frac{q_{t|0}(\boldsymbol{x}_t^{1:D}|\boldsymbol{x}_0^{1:D}) \cdot S_t(\boldsymbol{z}_t^{1:D}|\boldsymbol{x}_t^{1:D})/\mathcal{S}_t(\boldsymbol{x}_t^{1:D})}{\sum_{\boldsymbol{y}_y^{1:D}} q_{t|0}(\boldsymbol{x}^{1:D}|\boldsymbol{x}_0^{1:D}) \cdot S_t(\boldsymbol{z}_t^{1:D}|\boldsymbol{y}_t^{1:D})/\mathcal{S}_t(\boldsymbol{y}_t^{1:D})} \tag{5.73}$$

$$= \frac{(1 - \delta_{\boldsymbol{z}_t,\boldsymbol{x}_t}) \sum_{d=1}^{D} \delta_{\boldsymbol{z}_t^{\backslash d},\boldsymbol{x}_t^{\backslash d}} S_t^d(\boldsymbol{z}_t^d|\boldsymbol{x}_t^d)/\mathcal{S}_t(\boldsymbol{x}_t^{1:D}) \cdot q_{t|0}(\boldsymbol{x}_t^d \circ \boldsymbol{z}_t^{\backslash d}|\boldsymbol{x}_0^{1:D})}{\sum_{\boldsymbol{y}_t^{1:D}} \left[(1 - \delta_{\boldsymbol{z}_t,\boldsymbol{y}_t}) \sum_{d=1}^{D} \delta_{\boldsymbol{z}_t^{\backslash d},\boldsymbol{y}_t^{\backslash d}} S_t^d(\boldsymbol{z}_t^d|\boldsymbol{y}_t^d)/\mathcal{S}_t(\boldsymbol{y}_t^{1:D}) \cdot q_{t|0}(\boldsymbol{y}_t^d \circ \boldsymbol{z}_t^{\backslash d}|\boldsymbol{x}_0^{1:D}) \right]} \tag{5.74}$$

$$= \frac{(1 - \delta_{\boldsymbol{z}_t,\boldsymbol{x}_t}) \sum_{d=1}^{D} \delta_{\boldsymbol{z}_t^{\backslash d},\boldsymbol{x}_t^{\backslash d}} \frac{S_t^d(\boldsymbol{z}_t^d|\boldsymbol{x}_t^d)}{\mathcal{S}_t(\boldsymbol{x}_t^{1:D})} \cdot \frac{q_{t|0}(\boldsymbol{x}_t^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{z}_t^d|\boldsymbol{x}_0^d)}}{\sum_{\boldsymbol{y}_t^{1:D}} \left[(1 - \delta_{\boldsymbol{z}_t,\boldsymbol{y}_t}) \sum_{d=1}^{D} \delta_{\boldsymbol{z}_t^{\backslash d},\boldsymbol{y}_t^{\backslash d}} \frac{S_t^d(\boldsymbol{z}_t^d|\boldsymbol{y}_t^d)}{\mathcal{S}_t(\boldsymbol{y}_t^{1:D})} \cdot \frac{q_{t|0}(\boldsymbol{y}_t^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{z}_t^d|\boldsymbol{x}_0^d)} \right]} \tag{5.75}$$

$$= \frac{(1 - \delta_{\boldsymbol{z}_t,\boldsymbol{x}_t}) \sum_{d=1}^{D} \delta_{\boldsymbol{z}_t^{\backslash d},\boldsymbol{x}_t^{\backslash d}} \frac{S_t^d(\boldsymbol{z}_t^d|\boldsymbol{x}_t^d)}{\mathcal{S}_t(\boldsymbol{x}_t^{1:D})} \cdot \frac{q_{t|0}(\boldsymbol{x}_t^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{z}_t^d|\boldsymbol{x}_0^d)}}{\sum_{d=1}^{D} \sum_{\boldsymbol{y}_t^d \neq \boldsymbol{z}^d} \frac{S_t^d(\boldsymbol{z}_t^d|\boldsymbol{y}_t^d)}{\mathcal{S}_t(\boldsymbol{y}_t^d \circ \boldsymbol{z}_t^{\backslash d})} \cdot \frac{q_{t|0}(\boldsymbol{y}_t^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{z}_t^d|\boldsymbol{x}_0^d)}} \tag{5.76}$$

Now taking the formulation of q_{S_t} back into Term 2, we further simplify Term 2 as

$$T\ \mathbb{E}_{t,\boldsymbol{z}^{1:D}\sim S_t}\frac{\sum_{\boldsymbol{x}^{1:D}}\Big[(1-\delta_{\boldsymbol{z},\boldsymbol{x}})\sum_{d=1}^{D}\delta_{\boldsymbol{z}^{\backslash d},\boldsymbol{x}^{\backslash d}}\frac{S_t^d(\boldsymbol{z}^d|\boldsymbol{x}^d)}{S_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D})}\cdot\frac{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)}\cdot r_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D})\log\hat{r}_t^\theta(\boldsymbol{x}^{1:D}|\boldsymbol{z}^{1:D})\Big]}{\sum_{d=1}^{D}\sum_{\boldsymbol{y}^d\neq\boldsymbol{z}^d}\frac{S_t^d(\boldsymbol{z}^d|\boldsymbol{y}^d)}{\mathcal{S}_t(\boldsymbol{y}^d\circ\boldsymbol{z}^{\backslash d})}\cdot\frac{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)}} \tag{5.77}$$

$$=\mathbb{E}_{t,\boldsymbol{z}^{1:D}\sim S_t}\frac{\sum_{d=1}^{D}\sum_{\boldsymbol{x}^d\neq\boldsymbol{z}^d}\frac{S_t^d(\boldsymbol{z}^d|\boldsymbol{x}^d)}{S_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^d\circ\boldsymbol{z}^{\backslash d})}\cdot\frac{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)}\cdot r_t(\boldsymbol{z}^{1:D}|\boldsymbol{x}^d\circ\boldsymbol{z}^{\backslash d})\log\hat{r}_t^\theta(\boldsymbol{x}^d\circ\boldsymbol{z}^{\backslash d}|\boldsymbol{z}^{1:D})}{\sum_{d=1}^{D}\sum_{\boldsymbol{y}^d\neq\boldsymbol{z}^d}\frac{S_t^d(\boldsymbol{z}^d|\boldsymbol{y}^d)}{\mathcal{S}_t(\boldsymbol{y}^d\circ\boldsymbol{z}^{\backslash d})}\cdot\frac{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)}} \tag{5.78}$$

$$=\mathbb{E}_{t,\boldsymbol{z}^{1:D}\sim S_t}\frac{\sum_{d=1}^{D}\sum_{\boldsymbol{x}^d\neq\boldsymbol{z}^d}\frac{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)}\cdot r_t^d(\boldsymbol{z}^d|\boldsymbol{x}^d)\cdot\log r_t^d(\boldsymbol{z}^d|\boldsymbol{x}^d)g_t^{\theta,d}(\boldsymbol{x}^d|\boldsymbol{z}^{1:D})}{\sum_{d=1}^{D}\sum_{\boldsymbol{y}^d\neq\boldsymbol{z}^d}\frac{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)}\cdot\frac{S_t^d(\boldsymbol{z}^d|\boldsymbol{y}^d)}{\mathcal{S}_t(\boldsymbol{y}^d\circ\boldsymbol{z}^{\backslash d})}} \tag{5.79}$$

$$=\mathbb{E}_{t,\boldsymbol{z}^{1:D}\sim S_t}\frac{\sum_{d=1}^{D}\sum_{\boldsymbol{x}^d\neq\boldsymbol{z}^d}\frac{q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)}\cdot r_t^d(\boldsymbol{z}^d|\boldsymbol{x}^d)\cdot\log g_t^{\theta,d}(\boldsymbol{x}^d|\boldsymbol{z}^{1:D})}{\sum_{d=1}^{D}\sum_{\boldsymbol{y}^d\neq\boldsymbol{z}^d}\frac{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)}\cdot\frac{S_t^d(\boldsymbol{z}^d|\boldsymbol{y}^d)}{\mathcal{S}_t(\boldsymbol{y}^d\circ\boldsymbol{z}^{\backslash d})}}+\text{const.} \tag{5.80}$$

The above equation further shows that the sampling distribution S_t for adding exchanging variable $\boldsymbol{z}^{1:D}$ only affect a weighting term of the loss computation. Let us define the scalar weighting term as

$$\mathcal{M}_{S_t}(\boldsymbol{z}^{1:D}|\boldsymbol{x}_0^{1:D}):=\sum_{d=1}^{D}\sum_{\boldsymbol{y}^d\neq\boldsymbol{z}^d}\frac{q_{t|0}(\boldsymbol{y}^d|\boldsymbol{x}_0^d)}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)}\cdot\frac{S_t^d(\boldsymbol{z}^d|\boldsymbol{y}^d)}{\mathcal{S}_t(\boldsymbol{y}^d\circ\boldsymbol{z}^{\backslash d})} \tag{5.81}$$

With this definition, the Term 2 can be further rewrited as

$$\mathbb{E}_{t,\boldsymbol{z}^{1:D}\sim S_t}\Big[\frac{1}{\mathcal{M}_{S_t}(\boldsymbol{z}^{1:D}|\boldsymbol{x}_0^{1:D})}\sum_{d=1}^{D}\frac{1}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)}\sum_{\boldsymbol{x}^d\neq\boldsymbol{z}^d}q_{t|0}(\boldsymbol{x}^d|\boldsymbol{x}_0^d)\cdot r_t^d(\boldsymbol{z}^d|\boldsymbol{x}^d)\cdot\log g_t^{\theta,d}(\boldsymbol{x}^d|\boldsymbol{z}^{1:D})\Big] \tag{5.82}$$

$$=\mathbb{E}_{t,\boldsymbol{z}^{1:D}\sim S_t}\Big[\frac{1}{\mathcal{M}_{S_t}(\boldsymbol{z}^{1:D}|\boldsymbol{x}_0^{1:D})}\sum_{d=1}^{D}\frac{\mathbf{1}^\top[q_{t|0}(\cdot|\boldsymbol{x}_0^d)\odot r_t^d(\boldsymbol{z}^d|\cdot)\odot\log g_t^{\theta,d}(\cdot|\boldsymbol{z}^{1:D})]}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)}\Big]+\text{const.} \tag{5.83}$$

$$=\mathbb{E}_{t,\boldsymbol{z}^{1:D}\sim S_t}\Big[\frac{\beta(t)}{\mathcal{M}_{S_t}(\boldsymbol{z}^{1:D}|\boldsymbol{x}_0^{1:D})}\sum_{d=1}^{D}\frac{\langle\boldsymbol{z}^d,\boldsymbol{m}^d\rangle}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)}\mathbf{1}^\top[q_{t|0}(\cdot|\boldsymbol{x}_0^d)\odot\log g_t^{\theta,d}(\cdot|\boldsymbol{z}^{1:D})]+\text{const.}\Big] \tag{5.84}$$

All Terms

Combine term 1 and term 2 together, we can write the negative VLB loss as

$$T\ \mathbb{E}_{\substack{t\sim\text{Uni}(0,T)\\ \boldsymbol{x}^{1:D}\sim q_{t|0}}}\Big[\sum_{\boldsymbol{z}^{1:D}\neq\boldsymbol{x}^{1:D}}\hat{r}_t^{\theta,1:D}(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D})-\sum_{\boldsymbol{z}^{1:D}\neq\boldsymbol{x}^{1:D}}r_t^{1:D}(\boldsymbol{z}^{1:D}|\boldsymbol{x}^{1:D})\log\hat{r}_t^{\theta,1:D}(\boldsymbol{x}^{1:D}|\boldsymbol{z}^{1:D})\Big]$$

$$=T\ \mathbb{E}_{\substack{t\sim\text{Uni}(0,T)\\ \boldsymbol{x}^{1:D}\sim q_{t|0}(\boldsymbol{x}_0^{1:D})\\ \boldsymbol{z}^{1:D}\sim S_t(\boldsymbol{x}^{1:D})}}\Big[\sum_{d=1}^{D}r_t^d(\boldsymbol{x}^d|\cdot)^\top g_t^{\theta,d}(\cdot|\boldsymbol{x}^{1:D})-$$

$$\frac{1}{\mathcal{M}_{S_t}(\boldsymbol{z}^{1:D}|\boldsymbol{x}_0^{1:D})}\sum_{d=1}^{D}\frac{\mathbf{1}^\top[q_{t|0}(\cdot|\boldsymbol{x}_0^d)\odot r_t^d(\boldsymbol{z}^d|\cdot)\odot\log g_t^{\theta,d}(\cdot|\boldsymbol{z}^{1:D})]}{q_{t|0}(\boldsymbol{z}^d|\boldsymbol{x}_0^d)}\Big]+\text{const.} \tag{5.85}$$

□

This formulation simplifies and generalizes the result in Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22], such that any S_t can be used for introducing the auxiliary expectation variable $\boldsymbol{z}_{1:D}$. Importantly, Eq. (5.58) shows that changing S_t only affects a scalar weighting term. Computing the loss requires two passes of the model for $\boldsymbol{z}^{1:D}$ and $\boldsymbol{x}^{1:D}$ separately. Choosing S_t carefully such that $\boldsymbol{z}^{1:D}$ and $\boldsymbol{x}^{1:D}$ have the same distribution can avoid the second pass of the model, which we leave to future work.

5.3.4 Backward Sampling & Unification

Algorithm 2 USD3 Unified Training: Red: discrete-time step, and blue: continuous-time step.

```
1: Input: A stationary distribution $\boldsymbol{m}$, samples $\sim p_{\text{data}}$, weight factor $\lambda$, max time $T$
2: repeat
3:     Draw $\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x}_0)$
4:     Draw $t \sim \textbf{Uniform}(0, ..., T)$ or $t \sim \textbf{Uniform}(0, 1)$
5:     Compute $\overline{\alpha}_t$ from noise scheduler
6:     Draw $\mathbf{m}_0^{1:D} \sim \text{Cat}(\mathbf{m}_0^{1:D}; \boldsymbol{m})$
7:     $\mathbf{x}_{t|0}^{1:D} = \delta^{1:D}_{1,\mathbf{b}_t^{1:D}} \odot \mathbf{x}_0^{1:D} + (1 - \delta^{1:D}_{1,\mathbf{b}_t^{1:D}}) \odot \mathbf{m}_0^{1:D}$  where $\mathbf{b}_t \sim \text{Bernoulli}(\overline{\alpha}_t)$
8:     Compute $f_t^\theta(\boldsymbol{x}_t^{1:D})$ as parameterization of $p_\theta(\mathbf{x}_0^{1:D}|\mathbf{x}_t^{1:D})$
9:     Take gradient descent step on $\nabla_\theta\{\mathcal{L}_t(\theta) + \lambda\mathcal{L}_t^{CE}(\theta)\}$ (from Eq. (5.25) + Eq. (5.26))
10:    or $\nabla_\theta\{\mathcal{L}_t^{CTMC}(\theta) + \lambda\mathcal{L}_t^{CE}(\theta)\}$ (from Eq. (5.43) + Eq. (5.26))
11: until convergence
```

Algorithm 3 USD3 Unified Sampling

```
1: Input: A stationary distribution $\boldsymbol{m}$, $\{t_i\}_{i=0}^n$ s.t. $0 = t_0 < t_1 < ... < t_n = T$, learned $f_{t_i}^\theta$.
2: MCMC Input: use_MCMC, step size $\triangle n$, total steps $N$.
3: Set $\mathbf{x}_n^{1:D} \sim \text{Cat}(\mathbf{x}_n^{1:D}; \boldsymbol{m})$
4: for $i \in \{n, ..., 0\}$ do
5:     Compute $f_{t_i}^\theta(\boldsymbol{x}_i^{1:D})$ and $p_\theta(\mathbf{x}_{i-1}^{1:D}|\mathbf{x}_i^{1:D})$ from Eq. (5.21)
6:     Draw $\mathbf{x}_{i-1}^{1:D} \sim \text{Cat}(\mathbf{x}_{i-1}^{1:D}; p_\theta(\mathbf{x}_{i-1}^{1:D}|\mathbf{x}_i^{1:D}))$
7:     If use_MCMC:
8:         $\mathbf{x}_{i-1}^{1:D} \leftarrow \text{MCMC\_Corrector}(t_i, \mathbf{x}_{i-1}^{1:D}, \triangle n, f_{t_i}^\theta, N)$ (Algo. 4)
9: return $\mathbf{x}_0$
```

Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22] proposes to use the learned transition rate $\widehat{r}_t^\theta$ of the backward CTMC to sample reversely for the target distribution $p_{\text{data}}(\mathbf{x}_0)$. However, different from the forward CTMC where all elements are sampled independently, the backward CTMC processes for all elements are coupled together and do not have the closed-form transition probability derived from the learned transition rate. Direct and exact sampling from a CTMC uses the algorithm by Gillespie [Gil77], which is extremely inefficient for multi-element objects. Instead of exact sampling, Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22] proposes to use tau-leaping [Gil01] that approximately samples all transitions occurring from time t to $t-\tau$, assuming $R_t^{1:D}$ fixed during the time period. However, it 1) introduces unknown errors that needs additional correction process; 2) can only accept at most 1 alteration for each element from t to $t-\tau$ on categorical data, which requires small enough τ, hence potentially too many backward steps.

We propose to avoid the costly sampling from using the transition rate of the backward CTMC. We first observe that the estimated transition rate $\widehat{r}_t^\theta$ is essentially derived from the estimated $p_{0|t}^\theta(\mathbf{x}_0|\mathbf{x}_t)$. Hence using $\widehat{r}_t^\theta$ is equivalent to using $p_{0|t}^\theta(\mathbf{x}_0|\mathbf{x}_t)$ in an indirect way. We realize that the transition probability $q_{s|t}(\mathbf{x}_s|\mathbf{x}_t)$ can be computed easily from $p_{0|t}^\theta(\mathbf{x}_0|\mathbf{x}_t)$ directly, as shown in Eq. (5.21). With the help of Eq. (5.21), we can sample $\mathbf{x}_0$ or equivalently $\mathbf{x}_{0|T}$ through sampling $\mathbf{x}_{t_{n-1}|t_n}, ..., \mathbf{x}_{t_1|t_2}, \mathbf{x}_{t_0|t_1}$ sequentially, with any $\{t_i\}_{i=0}^n$ that satisfies $0{=}t_0 < t_1 < ... < t_n{=}T$. Although Eq. (5.21) is derived for discrete-time case, it applies directly in continuous-time case, thanks to the unification of the forward process for discrete- and continuous-time diffusion (§5.3.2). Hence, discrete-time and continuous-time diffusion also have the same **unified backward generation process**.

★ **Unified Discrete Diffusion.** All in all, through a series of mathematical simplifications, we have shown that both discrete&continuous-time discrete diffusion share **(1)** the same forward diffusion process; **(2)** the same parameterization for learning $p_{0|t}^\theta$; and **(3)** the same backward denoising process. In light of our reformulations, we propose USD3, a novel Unified and Simplified Discrete Denoising Diffusion model. Notably, USD3 utilizes the same source code for both discrete- and continuous-time, up to a single alteration in the loss function during training, and a shared sample generation process (resp. Algo. 2 and 3).

5.3.5 Shared MCMC Derivation

Campbell, Benton, De Bortoli, Rainforth, Deligiannidis, and Doucet [Cam+22] show that the MCMC for discrete data can be done by a predictor step to simulate $\widehat{r}_t^\theta$ and a corrector step using $r_t + \widehat{r}_t^\theta$. We extend this result with improved derivation and show that, thanks to the parameterized forms in Eq. (5.46), *both* discrete- and continuous-time discrete diffusion can leverage the same transition probability calculation (see Eq. (5.91)), leading to a shared MCMC scheme.

The MCMC Sampling Corrector

[Son+21] introduced a predictor-corrector step to further improve the quality of generated data based on score-based Markov Chain Monte Carlo (MCMC) for continuous-time diffusion over continuous distribution. [Cam+22] showed that there is a similar MCMC based corrector that can be used for CTMC to improve reverse sampling at any time t. Although we use different reverse sampling than [Cam+22], the similar corrector step can also be developed to improve the quality of reverse sampling introduced in §5.3.4. In this section, we derive the corrector formally and simplify it based the multi-element formulations summarized in Eq. (5.91).

Formally, at any time t, [Cam+22] proved that a *time-homogeneous* CTMC with transition rate being $c_t := r_t + \hat{r}_t$ has its stationary distribution being $q_t(\mathbf{x}_t^{1:D})$. To avoid ambiguity, we use $n \in [0, +\infty)$ as the time variable for that CTMC with stationary distribution $q_t(\mathbf{x}_t^{1:D})$. Then for any sample $\boldsymbol{z}_t^{1:D}$ generated from reverse sampling process at time t, we can push it closer to the target marginal distribution $q_t(\mathbf{x}_t^{1:D})$ by sampling from the corrector CTMC with initial value being $\boldsymbol{z}_t^{1:D}$, named as $\boldsymbol{z}_{t,n=0}^{1:D}$. Let

N be the maximum time allocated in the corrector CTMC, then after the corrector step $\boldsymbol{z}_{t,n=N}^{1:D}$ is used to replace the original $\boldsymbol{z}_t^{1:D}$.

We now introduce how to sampling from the CTMC. Let $\triangle n$ be the time incremental for each sampling step of the corrector CTMC. Solving the Kolmogorov forward equation of this time-homogeneous CTMC can derive the transition probability at any time n as

$$\forall n \text{ and } \triangle n,\ p_{n+\triangle n|n}(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D}) = \exp(\triangle n \cdot C_t^{1:D})[\boldsymbol{x}^{1:D}, \boldsymbol{y}^{1:D}] \tag{5.86}$$

Where C_t is the transition rate matrix of the corrector CTMC, and $\exp(\triangle n \cdot C_t^{1:D})$ is the transition probability matrix at time n. Notice that this matrix exponential does not have analytical formulation. Instead, we propose to control $\triangle n$ to be small enough such that

$$\exp(\triangle n \cdot C_t^{1:D}) \approx I + \triangle n \cdot C_t^{1:D} \tag{5.87}$$

Then taking it back we can obtain

$$\begin{aligned} p_{n+\triangle n|n}(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D}) &\approx \delta_{\boldsymbol{x}^{1:D},\boldsymbol{y}^{1:D}} + \triangle n \cdot c_t^{1:D}(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D}) \\ &= \delta_{\boldsymbol{x}^{1:D},\boldsymbol{y}^{1:D}} + \triangle n \sum_{d=1}^{D} \Big[r_t^d(\boldsymbol{y}^d|\boldsymbol{x}^d) + r_t^d(\boldsymbol{x}^d|\boldsymbol{y}^d) \cdot g_t^d(\boldsymbol{y}^d|\boldsymbol{x}^{1:D}) \Big] \delta_{\boldsymbol{x}^{\backslash d},\boldsymbol{y}^{\backslash d}} \end{aligned} \tag{5.88}$$

Instead of sampling all elements jointly, we propose to sample each element of the object independently from their individual marginal distribution, which can be analytically formulated as

$$\begin{aligned} p_{n+\triangle n|n}(\boldsymbol{y}^d|\boldsymbol{x}^{1:D}) &= \sum_{\boldsymbol{y}^{\backslash d}} p_{n+\triangle n|n}(\boldsymbol{y}^{1:D}|\boldsymbol{x}^{1:D}) \\ &= \triangle n \Big[r_t^d(\boldsymbol{y}^d|\boldsymbol{x}^d) + r_t^d(\boldsymbol{x}^d|\boldsymbol{y}^d) \cdot g_t^d(\boldsymbol{y}^d|\boldsymbol{x}^{1:D}) \Big] \text{ if } \boldsymbol{y}^d \neq \boldsymbol{x}^d \\ &= \triangle n (\boldsymbol{y}^d)^\top \Big[r_t^d(\cdot|\boldsymbol{x}^d) + r_t^d(\boldsymbol{x}^d|\cdot) \odot g_t^d(\cdot|\boldsymbol{x}^{1:D}) \Big] \text{ if } \boldsymbol{y}^d \neq \boldsymbol{x}^d \\ &= \triangle n \beta(t) (\boldsymbol{y}^d)^\top \Big[\boldsymbol{m}^d - \boldsymbol{x}^d + \big(\langle \boldsymbol{x}^d, \boldsymbol{m}^d \rangle \mathbf{1} - \boldsymbol{x}^d \big) \odot g_t^d(\cdot|\boldsymbol{x}^{1:D}) \Big] \text{ if } \boldsymbol{y}^d \neq \boldsymbol{x}^d \\ &= \triangle n \beta(t) (\boldsymbol{y}^d)^\top \Big[\boldsymbol{m}^d + \langle \boldsymbol{x}^d, \boldsymbol{m}^d \rangle g_t^d(\cdot|\boldsymbol{x}^{1:D}) \Big] \text{ if } \boldsymbol{y}^d \neq \boldsymbol{x}^d \\ &\approx \triangle n \beta(t) (\boldsymbol{y}^d)^\top \Big[\boldsymbol{m}^d + \langle \boldsymbol{x}^d, \boldsymbol{m}^d \rangle g_t^{\theta,d}(\cdot|\boldsymbol{x}^{1:D}) \Big] \text{ if } \boldsymbol{y}^d \neq \boldsymbol{x}^d \end{aligned} \tag{5.89}$$

Now we define the notation $\overline{p_{\triangle n}(\cdot|\boldsymbol{x}^{1:D})}$ to derive the distributional form of $p_{n+\triangle n|n}(\boldsymbol{y}^d|\boldsymbol{x}^{1:D})$.

$$\overline{p_{\triangle n}^{\theta,d}(\cdot|\boldsymbol{x}^{1:D})} := \triangle n \beta(t) \Big[\big(2 - \frac{\overline{\alpha}_{t|0} \langle f_t^{\theta,d}(\boldsymbol{x}^{1:D}), \boldsymbol{x}^d \rangle}{\overline{\alpha}_{t|0} + (1 - \overline{\alpha}_{t|0}) \langle \boldsymbol{x}^d, \boldsymbol{m}^d \rangle} \big) \boldsymbol{m}^d + \frac{\overline{\alpha}_{t|0}}{1 - \overline{\alpha}_{t|0}} f_t^{\theta,d}(\boldsymbol{x}^{1:D}) \Big] \odot (\mathbf{1} - \boldsymbol{x}^d) \tag{5.90}$$

With the above notation, the sampling probability can be further simplified as

$$p_{n+\triangle n|n}(\boldsymbol{y}^d|\boldsymbol{x}^{1:D}) = \operatorname{Cat}\Big(\boldsymbol{y}^d; \overline{p_{\triangle n}^{\theta,d}(\cdot|\boldsymbol{x}^{1:D})} + \big(1 - \mathbf{1}^\top \overline{p_{\triangle n}^{\theta,d}(\cdot|\boldsymbol{x}^{1:D})} \big) \boldsymbol{x}^d \Big) \tag{5.91}$$

Notice that $\triangle n$ should be set small enough such that $\mathbf{1}^\top \overline{p_{\triangle n}^{\theta,d}(\cdot|\boldsymbol{x}^{1:D})} \leq 1$. This condition can be used to derive $\triangle n$ dynamically. In practice, we can also easily clip the scale of $\overline{p_{\triangle n}^{\theta,d}(\cdot|\boldsymbol{x}^{1:D})}$ to 1 when $\mathbf{1}^\top \overline{p_{\triangle n}^{\theta,d}(\cdot|\boldsymbol{x}^{1:D})} > 1$ to prevent illness condition. Intuitively, $1 - \mathbf{1}^\top \overline{p_{\triangle n}^{\theta,d}(\cdot|\boldsymbol{x}^{1:D})}$ defines the keeping rate of the d-th element during correction step, and it should be larger with increasing t and n during the reverse sampling and correction period.

Algorithm 4 The MCMC correcting algorithm at time t

```
**Input:** The sample at time $t$ from reverse sampling, $\boldsymbol{z}_t^{1:D}$; step size $\triangle n$; learned $f_t^\theta$; total steps $N$.
**Initialize** $\boldsymbol{x}_{t,0}^{1:D} \leftarrow \boldsymbol{z}_t^{1:D}$
**for** $i$ from 1 to $N$ **do**
    Compute $p_{\triangle n}^{\theta,d}(\cdot|\boldsymbol{x}_{t,i-1}^{1:D})$ from Eq. (5.90);
    $\forall d$, Draw $\boldsymbol{x}_{t,i}^d \sim \mathrm{Cat}\Big(\cdot\,; \overline{p_{\triangle n}^{\theta,d}(\cdot|\boldsymbol{x}_{t,i-1}^{1:D})} + \big(1 - \mathbf{1}^\top \overline{p_{\triangle n}^{\theta,d}(\cdot|\boldsymbol{x}_{t,i-1}^{1:D})}\big)\boldsymbol{x}_{t,i-1}^d\Big)$
**end for**
**Output:** the improved (corrected) sample $\boldsymbol{x}_{t,N}^{1:D}$
```

5.4 Experiments

Discrete diffusion as a field is in its infancy with no established guidelines on model training. All prior work optimize a combined VLB and cross-entropy (CE) loss with a fixed weight, without deeper investigation. In this work we not only improve VLB loss mathematically, but also empirically explore an extensive testbed of training regimes for discrete diffusion—including various loss combinations, both discrete- and continuous-time training, as well as varying model sizes—toward a deeper understanding of which yield tractable optimization and higher generation quality. USD3 is a hybrid of our simplified exact VLB and CE, and USD3* refers to its approximation with further simplifications (§5.2.6 and §5.3.3). USD3-CE and USD3-VLB are variants, resp. with CE or VLB loss *only*.

5.4.1 Datasets and Metrics

Lakh Piano Dataset Details

The Lakh pianoroll dataset contains 6, 000 training and 973 evaluating piano sequences, with each music sequence spanning a length of 256 in total. Each music note in the sequence can take on a value of the 128 music notes plus 1 additional class meaning an empty note. The music note orderings are scrambled into the same random order as described in [Cam+22] such that the ordinal structure of the music notes are destroyed.

For evaluation, the first 32 notes of the 967 evaluation sequences are given to the model, while the model is asked to generate the resting 224 notes. Upon an analysis of the training and evaluation music sequences, we find that a total of 124 evaluation samples can be found with at least one matching training samples that have the same 32 dimensions. We separate out these samples and call the set PIANO-P. Among PIANO-P, 20 samples can be found to contain the same first 32 notes with 2 samples from the training sequences, three of them contain the same first 32 notes with 4 training samples each, one of them shares the same with 6 training samples, and one shares the same with 8 training samples. If the same 32 notes appear both in the training samples and as quests for the model to provide the inferences, the model is likely to directly memorizing the remaining 224 notes from the training set, and "parroting" music sequences according to the training samples. The rest 843 evaluation samples that do not have matching training samples are constituent of PIANO.

Pre-training VQGAN

To generate images in a categorical discrete latent space, we follow the implementation of VQGAN [ERO20] in MaskGIT [Cha+22]. Specifically, we use the same VQGAN setting as mentioned in [Sun+23]. VQGAN is a variant of Vector Quantized Variational Autoencoder (VQ-VAE) [OVK18]. In our setup for CIFAR10, a VQGAN encodes an image of shape $H \times W \times 3$ to $(\frac{H}{4} \times \frac{W}{4})$ tokens with vocabulary size of 512. For the encoder, we use three convolutional blocks, with filter sizes of 64,128 and 256, and an average pooling between each blocks. For each block, it consists of two residual blocks. After an image is encoded, the output is mapped to a token index with a codebook of 512×256. For VQGAN loss objective, the additional GAN loss and perceptual loss are added with weight 0.1. To train a general VQGAN model that allows us to embed CIFAR10 images without overfitting, we apply data augmentation (random flipping and cropping) to the 64×64 version of ImageNet dataset [Den+09], and train for 90 epochs. The VQGAN is trained with Adam optimizer ($\beta_1 = 0$, $\beta_2 = 0.99$), and the learning rate is linearly warmup to the peak of $1e^{-4}$ and then drops with cosine decay.

After VQGAN is trained, we freeze the VQGAN and apply encoder to the CIFAR10 images to create 8×8 latent codes. The latent codes then flattened to a vector of size 64 as the input of the diffusion model. Before we evaluate our diffusion methods, we will feed the generated latent codes back to the VQGAN decoder to reconstruct a sample in the image spaces. We test the effectivenss of VQGAN by trying to reconstruct the CIFAR10 dataset. The reconstruction gives a FID of 7.68 and IS of 10.42, using the Inception V3 Model [3].

5.4.2 Baselines.

We compare USD3 and variants to three latest SOTA discrete diffusion models in the literature: D3PM [Aus+21] (discrete-time), and τ-LDR [Cam+22] and SDDM [Sun+23] (continuous-time).

5.4.3 Training Details.

Thanks to unification, we can evaluate both discrete- and continuous-time models with both cosine and constant noise schedulers at ease. In USD3, we combine VLB and CE with weight 0.001 for the latter, following prior literature. For USD3*, we combine VLB with CE weight 1.0. As architecture, we parameterize $f_t^\theta(\mathbf{x}_t)$ with a sequence transformer model. By varying its depths and widths, we study the impact of model size.

USD3 Lakh Pianoroll Training Details.

For the backbone sequence transformer structure, we adopt a similar transformer model as utilized by in τ-LDR-0 [Cam+22]. The sequence transformer is composed of several encoder blocks, where for each internal block, time is fed into the block through a FiLM layer and added to the input. The result is then fed into a multi-headed self-attention layer and fully connected layer. We use RELU for activation. At the output of self-attention layer and fully-connected layer, a dropout of 0.1 is applied. After obtaining the final embedding of each token, we feed that into a 1-block ResNet to obtain the predicted logits. In comparison with other baseline metrics, the transformer contains 6 encoder blocks, each containing 16 attention heads, input dimension of 1024

[3] https://github.com/openai/consistency_models/tree/main/evaluations

and MLP dimension of 4096. In ablation study, we also test our methods on a smaller architecture that contains 6 encoder blocks, each containing 8 attention heads, with input dimension of 128 and MLP dimension of 1024.

For training the pianoroll dataset, we use a batch size of 64, a learning rate of $5e^{-4}$, with a warmup of first 25 epochs. We adopt a constant learning rate decay scheduler, decaying the learning rate by half after every 500 epochs. The final result is given over 3000 epochs. We run our results with 2 A6000 GPUs. In discrete-time diffusion, we sample 1000 number of timesteps, fixing a cosine scheduler with $\alpha = 0.008$, In continuous-time diffusion, we sample time t between $[0, 1]$ and apply a constant scheduler with rate equals 0.007. We maintain an exponential moving average of parameters with decay factor 0.9999. We clip the gradient norm at a norm value of 1.0.

Baseline Training Details.

For **D3PM** and **τ-LDR-0**, for fair comparison, we use the same architecture, diffusion scheme and training scheme. For calculating the loss of both methods, we follow the previous literature and give 0.001 to CE loss, and 1 to the VLB loss.

For **SDDM**, since it adopts a different architecture, directly utilizing SDDM with our experiment configurations is not feasible. Instead, we use the experiment configurations as given in the paper: the backbone structure is a hollow transformer [Sun+23]. Each transformer block has 6 layers with embedding size of 256, 8 attention heads and hidden dimension of 2048. The batch size is 64 and number of training steps is 2 million. The weight decay is set to $1e^{-6}$. The learning rate is at constant $1e^{-3}$. For diffusion, SDDM adopts the constant noise scheduler with a uniform rate constant of 0.04.

USD3 **VQCIFAR10 Training Details.**

For image generation task, we parameterize $f_t^{\theta}(\mathbf{x}_t)$ with the same sequence transformer as mentioned before. The model is a 12 layer transformer, where each layer has 16 attention heads, input embedding of 768 and a hidden layer size of 3072 for the MLPs. We use ReLU for activation. At the output of each internal block, a dropout of 0.1 is applied. The time is input into the network through FiLM layers before the self attention block. We use a learning rate of $5e^{-4}$, with warmup of 50000 steps, and a cosine learning rate decay scheduler, to train 2 million steps. In discrete-time diffusion, all the USD3 and its variants use the same cosine noise scheduler with $\alpha = 0.008$. For continuous-time diffusion, USD3-CE and USD3* apply the cosine noise scheduler with $\alpha = 0.008$. We find that USD3 is extremely hard to optimize due to the scale differences of coefficient $\beta(t)$, so we provide USD3* which clips the $\beta(t) = \max(1, \beta(t))$. USD3 utilizes a constant noise scheduler with 0.007, to match the scheduler in τ-LDR-0 and SDDM. We maintain an exponential moving average of parameters with decay factor 0.9999. We clip the gradient norm at a norm value of 1.0.

Baseline Training Details.

For **τ-LDR-0** and **D3PM**, we again use the same transformer architecture and same training scheme. We apply a hybrid loss with cross entropy as a directly supervision, added with 0.001 CE loss. The τ-LDR-0 uses a constant rate noise scheduler of 0.007, while D3PM uses cosine scheduler with $\alpha = 0.008$. We train all models in parallel on 2 A6000 GPUs.

For **SDDM**, the model uses a masked modeling, where backbone neural network is BERT-based, with 12 layers of transformers. Each layer has 12 attention heads, embedding size of 768 and hidden layer size of 3072 for MLPs. After obtaining the final embedding of each token, the output is fed that into a 2-block ResNet to acquire the predicted logits. For diffusion, SDDM uses a constant uniform rate of 0.007 as the noise scheduler in the forward process. In [Sun+23], the number of training step is set to $700,000$, where the learning rate is warmed up to $1e^{-4}$ during the first 3% steps, and then decays to 0 in a linear schedule. We extended the training to $2m$ steps, but did not observe improved performance.

5.4.4 Music Generation

We evaluate monophonic music generation on Piano, the cleaned Lakh pianoroll dataset [Raf16; Don+17], containing $6,000$ training and 973 evaluation (or test) sequences of 256 notes each. Here we perform conditional generation; given the first 32 notes, the models are required to generate the rest 224 notes.

Interestingly, we find that some (124) evaluation sequences share the *same* first 32 notes as one or more training sequences. Such repetition gives us a unique opportunity to investigate what-we-call "parroting"—the phenomenon that models are simply replaying the exact training data during generation. To that end we create Piano-P, a subset consisting only of these 124 evaluation sequences paired with their first-32-note-matching training sequences.

Music Generation Eval Metrics

In evaluation of the conditional music generation task, we apply the following metrics to measure generation quality:

- 1-gram Hellinger Distance ($\downarrow$) and 1-gram Proportion of Outliers ($\downarrow$): for these two metrics, we following the same evaluation as described in [Cam+22] and [Che23].

- $\{2,3\}$-gram Hellinger Distance ($\downarrow$) and $\{2,3\}$-gram Proportion of Outliers ($\downarrow$): similar to n-gram models, we first convert the music sequences into tuples of neighboring nodes. Then for Hellinger Distance, we compute the distance of the empirical probabilities of conditional generated samples to the ground truth samples. The empirical probabilities are constructed based on the histograms of the neighboring tuples, with bins being all possible $\{2,3\}$-gram nodes. Similarly, for the Proportion of Outliers, we count the fractions of newly appeared tuples that are not seen in the training samples. With these metrics, we are able to capture the sequence information instead of just measuring the single node distributions.

- Diverse Edit Distance ($\uparrow$): which accounts for the creativity/novelty of generated samples across multiple generation runs. For the conditionally generated music samples given the same first 32 notes, we calculate the edit distance, which is the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one music sequence into the other, between each two of the generation samples. The mean and standard deviation are obtained for all edit distances between pairs. The higher the diverse edit distance, the further apart the music sequences are and more creativity are enforced in the generation process. However, there is also a trade-off between diverse edit distance and other accuracy measurements when the model processes large uncertainty about the underlying distribution.

TABLE 5.1: Conditional music gen. quality (3 samples avg.) on PIANO w.r.t. n-gram Hellinger, n-gram Prop. of Out., and Diverse Edit Dist. Also, Train-to-Test Ratio for 3-gram Prop. of Out. on PIANO-P quantifies "parroting". **First** & **Second** shown in color.

	Method	n-gram Hellinger(↓)			n-gram Prop. of Out.(↓)			Div. Edit Dist. (↑)	3g-Prop. Ratio(↑)
		1gram	2gram	3gram	1gram	2gram	3gram		
Discrete-time	D3PM	0.398	0.530	0.591	0.120	0.253	0.379	**0.295**	2.221
	USD3-CE	0.375	0.483	0.574	**0.107**	0.209	0.303	0.047	2.888
	USD3-VLB	0.379	**0.464**	**0.542**	0.117	**0.184**	**0.273**	**0.082**	2.863
	USD3	0.377	0.469	0.552	**0.107**	**0.186**	0.286	0.064	**3.083**
	USD3*	0.375	0.470	0.555	0.110	0.191	**0.283**	0.066	2.959
Continuous-time	SDDM	0.375	0.485	0.577	0.110	0.205	0.340	0.060	2.901
	τ-LDR-0	0.379	0.481	0.571	0.114	0.207	0.320	0.050	2.965
	USD3-CE	**0.373**	0.483	0.577	0.115	0.221	0.346	0.043	2.637
	USD3-VLB	0.376	0.470	0.552	0.111	0.191	0.291	0.066	2.805
	USD3	**0.371**	0.479	0.575	0.111	0.207	0.322	0.051	**3.082**
	USD3*	0.375	**0.465**	**0.548**	0.114	0.190	0.285	0.078	2.867

- Train-to-Test Ratios (↑) for $\{1, 2, 3\}$-gram Hellinger as well as Proportion of Outliers, which compare the weighted distance of a generated sample to its evaluation (test) vs. training sequence that share the same first 32 notes. We evaluate the ratios only for PIANO-P. Denote the evaluation ground truth set in PIANO-P as *tr* , the corresponding set of training samples as *ts*, and conditionally generated samples as *gs*. For the selected distance metrics *dist()* (from n-gram Hellinger or n-gram Proportion of Outliers), we calculate the ratio as: $\frac{1}{dist(tr,gs)+dist(ts,gs)} * \frac{dist(tr,gs)}{dist(ts,gs)}$. Such ratio measures quantify the extent of "parroting" in PIANO-P. The larger the ratio, the more equally distant the generated examples is from its training and evaluating set, and the less "parroting" occurs by simply memorizing all training sequences. We apply an additional coefficient to the ratio such that it will also penalize the models that provide unrealistic examples that do not conform both training and evaluation distributions.

Results.

Table 5.1 shows that USD3 and its variants perform better than the baseline methods across metrics. An exception is D3PM's Diverse Edit Distance, which trades-off high novelty with low generation quality. USD3-CE, while performing well w.r.t. 1-gram metrics, does not compete with USD3-VLB and USD3 w.r.t. {2,3}-gram metrics, which indicates that *CE only* loss captures the least sequential information. Models with combined losses, USD3 and USD3*, achieve higher Train-to-Test ratio than pure CE or VLB based ones, suggesting that the combination of two losses can alleviate overfitting, i.e. "parroting" the training data. While continuous-time baselines outperform the discrete-time baseline D3PM, we find discrete-time USD3 models to perform better than continuous-time counterparts except w.r.t. 1-gram Hellinger. This may be due to task complexity not warranting the harder optimization with the latter models that require denoising any timestep.

We also report evaluation results for different model sizes in Table 5.2. This ablation study shows that CE loss is also preferred in combination with VLB, especially for smaller network structures, since VLB is harder to optimize alone.

TABLE 5.2: Metrics comparing different loss combinations and different model sizes for Lakh Pianoroll. For each of n-gram Hellinger Distance (ng.-Hellinger) and Proportion of Outliers (ng.-Prop. Outlier) metrics, we show mean ± std with respect to 3 generated samples. We use USD3-VLB to denote an additional variant of our model that only uses the exact VLB loss in training. "Small" refers to the backbone transformer model that has 6 Layers, 8 Attention Heads, Input Dimension of 128 and MLP dimension of 1024. The top two are highlighted by **First**, **Second**.

	Method	1g.-Hellinger(↓)	2g.-Hellinger(↓)	3g.-Hellinger(↓)	1g.-Prop.Outlier(↓)	2g.-Prop.Outlier(↓)	3g.-Prop.Outlier(↓)	Edit Distance(↑)
Discrete-time	USD3-CE-Small	0.3984±0.0006	0.4902 ±0.0004	0.5785±0.0004	0.1158±0.0002	0.1899±0.0006	0.3142±0.0005	0.1301±0.0613
	USD3-Small	0.4011±0.0014	0.4902±0.0009	0.5707±0.0008	0.1215± 0.0007	**0.1866±0.0006**	0.3006±0.0013	0.1292±0.0623
	USD3-VLB-Small	0.4115±0.0001	0.4954± 0.0001	0.5738±0.0005	0.1203±0.0006	0.1958±0.0009	0.3036±0.0010	0.2137±0.0843
	USD3-CE	0.3754±0.0007	0.4835±0.0009	0.5741±0.0008	**0.1079±0.0002**	0.2099±0.0005	0.3034±0.0007	0.0472±0.0465
	USD3	0.3770±0.0011	**0.4693±0.0015**	**0.5525±0.0015**	**0.1077±0.0006**	**0.1861±0.0011**	**0.2861±0.0013**	0.0648±0.0459
	USD3-VLB	0.3790±0.0009	**0.4640±0.0010**	0.5427±0.0009	0.1174±0.0007	0.1845±0.0010	**0.2734±0.0011**	0.0828±0.0567
Continuous-time	USD3-CE-Small	0.4208±0.0170	0.5196±0.0078	0.6072±0.0005	0.1355±0.0147	0.2276±0.0007	0.3431±0.0181	0.2358±0.0619
	USD3-Small	0.4239±0.0012	0.5083±0.0011	0.5852±0.0011	0.1403±0.0007	0.2070±0.0008	0.2943±0.0016	**0.2468±0.0907**
	USD3-VLB-Small	0.4435±0.0012	0.5295±0.0011	0.6070±0.0009	0.1562±0.0014	0.2269±0.0013	0.3168±0.0013	**0.2809±0.0866**
	USD3-CE	**0.3734±0.0002**	0.4837±0.0003	0.5776±0.0003	0.1158±0.0004	0.2218±0.0003	0.3461±0.0005	0.0434±0.0357
	USD3	**0.3712±0.0005**	0.4794±0.0010	0.5752±0.0012	0.1112±0.0002	0.2074±0.0009	0.3225±0.0016	0.0516±0.0348
	USD3-VLB	0.3769±0.0007	0.4702±0.0010	**0.5527±0.0012**	0.1118±0.0009	0.1915±0.0010	0.2912±0.0014	0.0661±0.0478

TABLE 5.3: Image gen. quality w.r.t. Inception Score (IS) and the Frechet Inception Dist. (FID) over 50,000 samples unconditionally generated and decoded by VQGAN, as compared against original CIFAR10 training images. **First** & **Second** shown in color.

	Method	IS (↑)	FID (↓)		Method	IS (↑)	FID (↓)
Discrete-time	D3PM	8.13	18.08	Continuous-time	SDDM	8.72	14.17
					τ-LDR	8.37	17.61
	USD3-CE	9.02	12.64		USD3-CE	**9.23**	**11.97**
	USD3*	**9.27**	**12.07**		USD3*	8.59	15.87
	USD3	8.85	13.25		USD3	8.78	13.63
VQGAN Recons. (upper limit)						10.42	7.68

5.4.5 Image Generation

CIFAR10 contains 50,000 training images in continuous values, which we convert to vectors from 64×512-dimensional quantization hash-code space with a pre-trained VQGAN [ERO20]. This conversion allows us to (1) evaluate our methods on *nominal* data by breaking the neighboring orders in the image space, and (2) select a closed-form stationary distribution $\boldsymbol{m}$.

Metrics.

After feeding the generated samples through the VQGAN decoder to obtain representations from the discretized space, we measure the Inception Score (IS) (↑) and Frechet Inception Distance (FID) (↓) against the original CIFAR10 dataset. Note that our training set, which are discretized images from VQGAN, achieves IS=10.42 and FID=7.68, which are optimistic limits for generation.

Results.

Table 5.3 shows that our proposed approaches outperform existing baselines. Discrete-time D3PM falls short for the harder image generation task, whereas our simplified discrete-time loss boosts quality significantly. In continuous-time, τ-LDR with a similar loss to USD3 falls short due to its complicated generation process, whereas SDDM is most competitive among the baselines, although requires substantial compute resources while being limited to a specialized model architecture. USD3-CE achieves

Table 5.4: The GPU-memory, running time and number of network parameters in all methods.USD3 is easier to train and incurs the least GPU memory in both discrete- and continuous- time diffusions.

	Method	Num. Parameters	Memory	Runtime		Method	Num. Parameters	Memory	Runtime
Discrete-time	D3PM	∼ 102,700,000	15669MiB	93 hrs	Continuous-time	SDDM	∼ 12,350,000	85528MiB	96 hrs
	USD3-CE	∼ 102,700,000	9735MiB	83 hrs		τ-LDR-0	∼ 102,700,000	15669 MiB	129 hrs
	USD3*	∼ 102,700,000	9735MiB	82 hrs		USD3-CE	∼ 102,700,000	15703MiB	93 hrs
	USD3	∼ 102,700,000	9735MiB	90 hrs		USD3*	∼ 102,700,000	15703MiB	91 hrs
						USD3	∼ 102,700,000	15703MiB	101 hrs

competitive IS and FID scores, which validates our finding (recall Eq. (5.30)) that CE loss is essential for diffusion loss minimization.

We provide memory and runtimes of models in Table 5.4, example images generated by USD3 in Fig. 5.2.

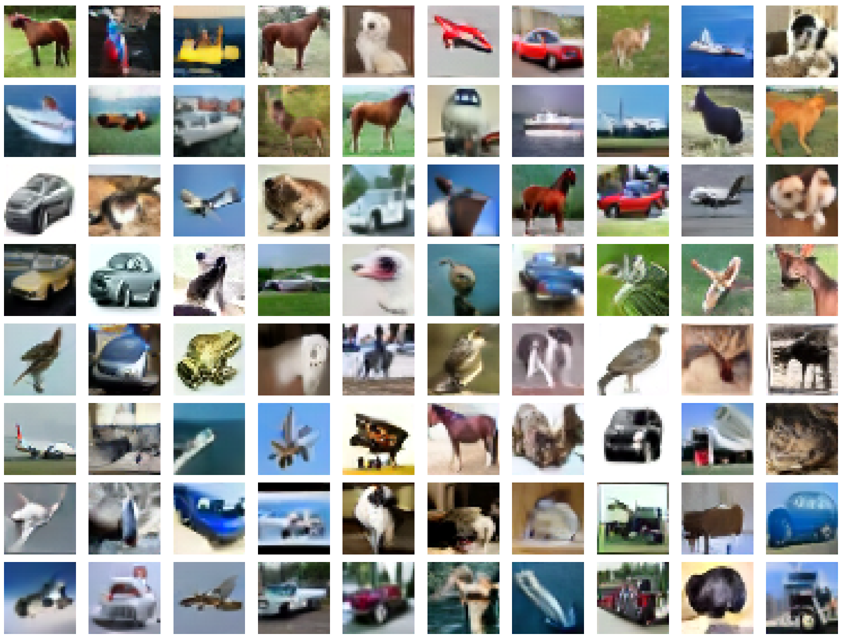


Figure 5.2: Example image samples generated by USD3* as trained on VQCIFAR10. Most images are easy to recognize as being from one of the 10 classes in CIFAR10.

Ablation results using MCMC sampling

To demonstrate the effectiveness of MCMC corrector steps, we take the top performing methods in discrete and continuous time diffusion models (for discrete time, USD3*, and for continuous time USD3-CE) and show improved quality metrics over generated images after MCMC corrector is applied. Due to extensive time required for MCMC corrector step, we could only conduct evaluation over $5,000$ images, and thus the results are not comparable to the main VQCIFAR10 result in Table 5.3. We set the number of generation steps to be 100 and use MCMC corrector on the last 10, 20 generation timesteps, respectively.

From the results shown in Table 5.5 and Table 5.6, we can see that MCMC corrector can significantly improve the quality of the generated samples. Specifically, for discrete-time generation, IS is showing a significant improvement than the sampling process without the MCMC corrector. For continuous-time case, both IS and FID scores are improving from the baseline (without MCMC).

TABLE 5.5: Image gen. quality w.r.t. Inception Score (IS) and the Frechet Inception Dist. (FID) over 5,000 samples unconditionally generated by USD3* in discrete-time case. MCMC corrector is conducted for the last 10,20 timesteps over 100 sampling steps.

MCMC Configuration	IS (↑)	FID (↓)
Without MCMC	9.01	19.79
$\Delta n = 0.001, N = 2$, Start Steps:10	9.28	18.46
$\Delta n = 0.001, N = 2$, Start Steps:20	**9.59**	18.36
$\Delta n = 0.005, N = 2$, Start Steps:10	9.43	18.26
$\Delta n = 0.005, N = 2$, Start Steps:20	9.43	20.37
$\Delta n = 0.001, N = 5$, Start Steps:10	9.29	**18.02**
$\Delta n = 0.001, N = 5$, Start Steps:20	9.47	18.56
$\Delta n = 0.002, N = 5$, Start Steps:10	9.35	18.18
$\Delta n = 0.002, N = 5$, Start Steps:20	9.48	20.37

TABLE 5.6: Image gen. quality w.r.t. Inception Score (IS) and the Frechet Inception Dist. (FID) over 5,000 samples generated by USD3-CE in continuous-time case. MCMC corrector is conducted for the last 10,20 timesteps over 100 sampling steps.

MCMC Configuration	IS (↑)	FID (↓)
Without MCMC	8.98	19.19
$\Delta n = 0.001, N = 2$, Start Steps:10	9.12	17.62
$\Delta n = 0.001, N = 2$, Start Steps:20	**9.23**	17.35
$\Delta n = 0.005, N = 2$, Start Steps:10	9.01	17.71
$\Delta n = 0.005, N = 2$, Start Steps:20	**9.23**	17.58
$\Delta n = 0.001, N = 5$, Start Steps:10	9.03	**17.26**
$\Delta n = 0.001, N = 5$, Start Steps:20	9.00	17.42
$\Delta n = 0.002, N = 5$, Start Steps:10	9.16	17.98
$\Delta n = 0.002, N = 5$, Start Steps:20	8.97	17.83

5.5 Conclusion

This work introduced two fundamental contributions for both discrete-time and continuous-time diffusion for categorical data. First, we presented extensive *mathematical simplifications for the loss functions*, including exact closed-form derivations as well as novel easy-to-optimize approximations. Second, we established a *mathematical unification of the backward denoising processes* of discrete-time and continuous-time diffusion, enabling faster generation and flexible training with varying noise schedules. Equipped with these advances in both training (thanks to simpler loss computation) and generation (thanks to flexible sampling), our proposed approach USD3 for discrete diffusion achieved state-of-the-art performance on established datasets across a suite of generation quality metrics.

Chapter 6

Permutation-Invariant Autoregressive Diffusion on Graphs

This chapter is based on Lingxiao Zhao, Xueying Ding, and Leman Akoglu. "Pard: Permutation-Invariant Autoregressive Diffusion for Graph Generation". In: *arXiv preprint arXiv:2402.03687* (2024)

6.1 Introduction

Graphs provide a powerful abstraction for representing relational information in many domains, including social networks, biological and molecular structures, recommender systems, and networks of various infrastructures such as computers, roads, etc. Accordingly, generative models of graphs that learn the underlying graph distribution from data find applications in network science [Bon+20], drug discovery [LZL18; Ton+21], protein design [AH18; Tri+21], and various use-cases for Internet of Things [De+22]. Also, they serve as a prerequisite for building a generative foundation model [Bom+21] for graphs.

Despite significant progress in generative models for images and language, graph generation is uniquely challenged by its inherent combinatorial nature. Specifically: 1) Graphs are naturally high-dimensional and *discrete* with *varying sizes*, contrasting with the continuous space and fixed-size advancements that cannot be directly applied here; 2) Being permutation-invariant objects, graphs require modeling an *exchangeable probability* distribution, where permutations of nodes and edges do not alter the graph's probability; 3) The rich substructures in graphs necessitate an expressive model capable of capturing *higher-order* motifs and interactions. Several graph generative models have been proposed to address (part of) these challenges, based on various techniques like autoregression [You+18; Lia+19], VAEs [SK18], GANs [DCK18], flow-based methods [Shi+20], and denoising diffusion [Niu+20; Vig+23]. Among these, autoregressive models and diffusion models stand out with superior performance, thus significant popularity. However, current autoregressive models, while efficient, are sensitive to order with non-exchangeable probabilities; whereas diffusion models, though promising, are less efficient, requiring thousands of denoising steps and extra features to achieve high generation quality.

In this paper, we introduce PARD (leopard in Ancient Greek), the *first* Permutation-invariant AutoRegressive Diffusion model that combines the efficiency of autoregressive methods and the quality of diffusion models together, while retaining the property of exchangeable probability. Instead of generating an entire graph directly, we explore the direction of generating through *block-wise* graph enlargement. Graph enlargement offers a fine-grained control over graph generation, which can be particularly advantageous for real-world applications that require local revisions to generate graphs. Moreover, it essentially decomposes the joint distribution of the graph into a series of simpler conditional distributions, thereby leveraging the data efficiency characteristic of autoregressive modeling. We also argue that graphs, unlike sets, inherently exhibit a *unique partial order* among nodes, naturally facilitating the decomposition of the joint distribution. Thanks to this unique partial order, PARD's block-wise sequence is permutation-equivariant.

To model the conditional distribution of nodes and edges in a block, we first show that the corresponding graph transformation cannot be solved directly with *any* equivariant network no matter how powerful it is. However, through a diffusion process that injects noise, a permutation equivariant network can progressively denoise to realize targeted graph transformations. This approach is inspired by the annealing process where energy is initially heightened before achieving a stable state, akin to the process of tempering iron. Our analytical findings serve as the foundation for the design of our proposed PARD that combines autoregressive approach with local block-wise discrete denoising diffusion. Using a diffusion model with equivariant networks ensures that each block's conditional distribution is exchangeable. Coupled with the permutation-invariant sequence of blocks, this renders the entire process permutation invariant and the joint distribution exchangeable.

Within PARD, we further propose several architectural improvements. First, to achieve 2-FWL expressivity with improved memory efficiency, we propose a higher-order graph transformer that integrates the transformer framework with PPGN [Mar+19a], while utilizing a significantly reduced representation size for edges. Second, to ensure training efficiency without substantial overhead compared to the original diffusion model, we design a GPT-like causal mechanism to support parallel training of all blocks with shared representations. These extensions are generalizable and can lay the groundwork for a higher-order GPT.

PARD achieves new SOTA performance on many molecular and non-molecular datasets *without any extra features*, significantly outperforming DiGress [Vig+23]. Thanks to efficient architecture and parallel training, PARD scales to large datasets like MOSES [Pol+20] with 1.9M graphs. Finally, not only PARD can serve as a generative foundation model for graphs in the future, its autoregressive parallel mechanism can further be combined with language models for language-graph generative pretraining, planting seeds for high-potential future work.

6.2 Related Work

Autoregressive (AR) Models for Graph Generation. AR models create graphs step-by-step, adding nodes and edges sequentially. This method acknowledges graphs' discrete nature but faces a key challenge as there is no inherent order in graph generation. To address this, various strategies have been proposed to simplify orderings and approximate the marginalization over permutations; i.e. $p(G) = \sum_{\pi \in \mathcal{P}(G)} p(G, \pi)$. Li, Vinyals, Dyer, Pascanu, and Battaglia [Li+18] propose using random or deterministic empirical orderings. GraphRNN [You+18] aligns permutations with breadth-first-search (BFS) ordering, with a many-to-one mapping. GRAN [Lia+19] offers marginalization over a family of canonical node orderings, including node degree descending, DFS/BFS tree rooted at the largest degree node, and k-core ordering. GraphGEN

[GJR20] uses a single canonical node ordering, but does not guarantee the same canonical ordering during generation. Chen, Han, Hu, Ruiz, and Liu [Che+21] avoid defining ad-hoc orderings by modeling the conditional probability of orderings, $p(\pi|G)$, with a trainable AR model, estimating marginalized probabilities during training to enhance both the generative model and the ordering probability model.

Diffusion Models for Graph Generation. EDP-GNN [Niu+20] is the first work that adapts score matching [SE19] to graph generation, by viewing graphs as matrices with continuous values. GDSS [JLH22] generalizes EDP-GNN by adapting SDE-based diffusion [Son+21] and considers node and edge features. Yan, Liang, Song, Liao, and Wang [Yan+23] argues that learning exchangeable probability with equivariant networks is hard, hence proposes permutation-sensitive SwinGNN with continuous-state score matching. Previous works apply continuous-state diffusion to graph generation, ignoring the natural discreteness of graphs. DiGress [Vig+23] is the first to apply discrete-state diffusion [Aus+21; Hoo+21] to graph generation and achieves significant improvement. However, DiGress relies on many additional structural and domain-specific features. GraphArm [Kon+23] applies Autoregressive Diffusion Model (ADM) [Hoo+22a] to graph generation, where exactly one node and its adjacent edges decay to the absorbing states at each forward step based on a *random* node order. Similar to AR models, GraphArm is permutation sensitive.

We remark that although both are termed “autoregressive diffusion”, it is important to distinguish that PARD is not equivalent to ADM. The term “autoregressive diffusion” in our context refers to the integration of autoregressive methods with diffusion models. In contrast, ADM represents a specific type of discrete denoising diffusion where exactly one dimension decays to an absorbing state at a time in the forward diffusion process. See Fan et al. [Fan+23] for a comprehensive survey of recent diffusion models on graphs.

6.3 Autoregressive Denoising Diffusion

We first introduce setting and notations. We focus on graphs with categorical features. Let $G = (\mathcal{V}, \mathcal{E})$ be a *labeled* graph with maximum K_v and K_e distinct node and edge labels respectively. Let $\boldsymbol{v}^i \in \{0,1\}^{K_v}, \forall i \in |\mathcal{V}|$ be the one-hot encoding of node i's label. Let $\boldsymbol{e}^{i,j} \in \{0,1\}^{K_e}, \forall i,j \in |\mathcal{V}|$ be the one-hot encoding of the label for the edge between node i and j. We also represent “absence of edge” as a type of edge label, hence $|\mathcal{E}| = |\mathcal{V}| \times |\mathcal{V}|$. Let $\boldsymbol{V} \in \{0,1\}^{|\mathcal{V}|\times K_v}$ and $\boldsymbol{E} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|\times K_e}$ be the collection of one-hot encodings of all nodes and edges. To describe probability, let $\mathbf{x}$ be a random variable with its sampled value $\boldsymbol{x}$. In diffusion process, noises are injected from $t{=}0$ to $t{=}T$ with T being the maximum time step. Let $\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x}_0)$ be the random variable of observed data with underlying distribution $p_{\text{data}}(\mathbf{x}_0)$, $\mathbf{x}_t \sim q(\mathbf{x}_t)$ be the random variable at time t, and let $\mathbf{x}_{t|s} \sim q(\mathbf{x}_t|\mathbf{x}_s)$ be the conditional random variable. Also, we interchangeably use $q(\mathbf{x}_t|\mathbf{x}_s)$, $q(\mathbf{x}_t{=}\boldsymbol{x}_t|\mathbf{x}_s{=}\boldsymbol{x}_s)$, and $q_{t|s}(\boldsymbol{x}_t|\boldsymbol{x}_s)$ when there is no ambiguity. We model the *forward diffusion* process independently for each node and edge of the graph, while the *backward denoising* process is modeled jointly for all nodes and edges. All vectors are column-wise vectors. Let $\langle\cdot,\cdot\rangle$ denote inner product.

6.3.1 Discrete Denoising Diffusion on Graphs

Denoising Diffusion is first developed by Sohl-Dickstein, Weiss, Maheswaranathan, and Ganguli [SD+15] and later improved by Ho, Jain, and Abbeel [HJA20]. It is further generalized to discrete-state case by Hoogeboom, Nielsen, Jaini, Forré, and Welling [Hoo+21] and Austin, Johnson, Ho, Tarlow, and Berg [Aus+21]. Taking a graph G_0 as example, diffusion model defines a forward diffusion process to gradually inject noise to all nodes and edges independently until all reach a non-informative state G_T. Then, a denoising network is trained to reconstruct G_0 from the noisy sample G_t at each time step, by optimizing a Variational Lower Bound (VLB) for $\log p_\theta(\mathrm{G}_0)$. Specifically, the forward process is defined as a Markov chain with $q(\mathrm{G}_t|\mathrm{G}_{t-1}), \forall t \in [1, T]$, and the backward denoising process is parameterized with another Markov chain $p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t), \forall t \in [1, T]$. Note that while the forward process is independently applied to all elements, the backward process is coupled together with conditional independence assumption. Formally,

$$q(\mathrm{G}_t|\mathrm{G}_{t-1}) = \prod_i^{|\mathcal{V}|} q(\mathbf{v}_t^i|\mathbf{v}_{t-1}^i) \prod_{i,j}^{|\mathcal{V}|} q(\mathbf{e}_t^{i,j}|\mathbf{e}_{t-1}^{i,j}) \tag{6.1}$$

$$p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t) = \prod_i^{|\mathcal{V}|} p_\theta(\mathbf{v}_{t-1}^i|\mathrm{G}_t) \prod_{i,j}^{|\mathcal{V}|} p_\theta(\mathbf{e}_{t-1}^{i,j}|\mathrm{G}_t)\ . \tag{6.2}$$

The VLB lower bound can be written as

$$\begin{aligned} \log p_\theta(\mathrm{G}_0) &= \log \int q(\mathrm{G}_{1:T}|\mathrm{G}_0) \frac{p_\theta(\mathrm{G}_{0:T})}{q(\mathrm{G}_{1:T}|\mathrm{G}_0)} d\mathrm{G}_{1:T} \\ &\geq \mathbb{E}_{q(\mathrm{G}_1|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_0|\mathrm{G}_1)\big] - D_{\mathrm{KL}}\big(q(\mathrm{G}_T|\mathrm{G}_0)||p_\theta(\mathrm{G}_T)\big) \\ &- \sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(\mathrm{G}_t|\mathrm{G}_0)}\big[D_{\mathrm{KL}}\big(q(\mathrm{G}_{t-1}|\mathrm{G}_t, \mathrm{G}_0)||p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)\big]}_{\mathcal{L}_t(\theta)}\ , \end{aligned} \tag{6.3}$$

Proof.

$$\log \int q(\mathrm{G}_{1:T}|\mathrm{G}_0) \frac{p_\theta(\mathrm{G}_{0:T})}{q(\mathrm{G}_{1:T}|\mathrm{G}_0)} d\mathrm{G}_{1:T} \geq \mathbb{E}_{q(\mathrm{G}_{1:T}|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_{0:T}) - \log q(\mathrm{G}_{1:T}|\mathrm{G}_0)\big] \tag{6.4}$$

$$\begin{aligned} &= \mathbb{E}_{q(\mathrm{G}_{1:T}|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_{0:T}) + \sum_{t=1}^{T} \log \frac{p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)}{q(\mathrm{G}_t|\mathrm{G}_{t-1})}\big] \\ &= \mathbb{E}_{q(\mathrm{G}_{1:T}|\mathbf{x}_0)}\big[\log p_\theta(\mathrm{G}_T) + \log \frac{p_\theta(\mathrm{G}_0|\mathrm{G}_1)}{q(\mathrm{G}_1|\mathrm{G}_0)} + \sum_{t=2}^{T} \log \frac{p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)}{q(\mathrm{G}_t|\mathrm{G}_{t-1}, \mathrm{G}_0)}\big] \\ &= \mathbb{E}_{q(\mathrm{G}_{1:T}|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_T) + \log \frac{p_\theta(\mathrm{G}_0|\mathrm{G}_1)}{q(\mathrm{G}_1|\mathrm{G}_0)} + \sum_{t=2}^{T} \log \left(\frac{p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)}{q(\mathrm{G}_{t-1}|\mathrm{G}_t, \mathrm{G}_0)} \cdot \frac{q(\mathrm{G}_{t-1}|\mathrm{G}_0)}{q(\mathrm{G}_t|\mathrm{G}_0)} \right)\big] \\ &= \mathbb{E}_{q(\mathrm{G}_{1:T}|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_T) + \log \frac{p_\theta(\mathrm{G}_0|\mathrm{G}_1)}{q(\mathrm{G}_1|\mathrm{G}_0)} + \log \frac{q(\mathrm{G}_1|\mathrm{G}_0)}{q(\mathrm{G}_T|\mathrm{G}_0)} + \sum_{t=2}^{T} \log \frac{p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)}{q(\mathrm{G}_{t-1}|\mathrm{G}_t, \mathrm{G}_0)}\big] \\ &= \mathbb{E}_{q(\mathrm{G}_{1:T}|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_0|\mathrm{G}_1) + \log \frac{p_\theta(\mathrm{G}_T)}{q(\mathrm{G}_T|\mathrm{G}_0)} - \sum_{t=2}^{T} \log \frac{q(\mathrm{G}_{t-1}|\mathrm{G}_t, \mathrm{G}_0)}{p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)}\big] \\ &= \underbrace{\mathbb{E}_{q(\mathrm{G}_1|\mathrm{G}_0)}\big[\log p_\theta(\mathrm{G}_0|\mathrm{G}_1)\big]}_{-\mathcal{L}_1(\theta)} - \sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(\mathrm{G}_t|\mathrm{G}_0)}\big[D_{\mathrm{KL}}\big(q(\mathrm{G}_{t-1}|\mathrm{G}_t, \mathrm{G}_0)||p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)\big]}_{\mathcal{L}_t(\theta)} - \text{const.} \end{aligned} \tag{6.5}$$

Using Eq. (6.2), the first term can simplified as

$$\mathbb{E}_{q(\mathrm{G}_1|\mathrm{G}_0)}[\sum_i \log p_\theta(\mathbf{v}_0^i|\mathrm{G}_1) + \sum_{i,j} \log p_\theta(\mathbf{e}_0^{i,j}|\mathrm{G}_1)]\ , \tag{6.6}$$

and similarly, the t-th step loss $\mathcal{L}_t(\theta)$ is

$$\mathbb{E}_{q(\mathrm{G}_t|\mathrm{G}_0)}\big[\sum_i KL\big(q(\mathbf{v}^i_{t-1}|\mathbf{v}^i_t,\mathbf{v}^i_0)\ ||\ p_\theta(\mathbf{v}^i_{t-1}|\mathrm{G}_t\big)+ \sum_{i,j} KL\big(q(\mathbf{e}^{i,j}_{t-1}|\mathbf{e}^{i,j}_t,\mathbf{e}^{i,j}_0)\ ||\ p_\theta(\mathbf{e}^{i,j}_{t-1}|\mathrm{G}_t)\big] \tag{6.7}$$

□

where the second term is ≈ 0, since $p_\theta(\mathrm{G}_T) \approx q(\mathrm{G}_T|\mathrm{G}_0)$ is designed as a fixed noise distribution that is easy to sample from. To compute Eq. (6.3), we need to formalize the distributions (i) $q(\mathrm{G}_t|\mathrm{G}_0)$ and (ii) $q(\mathrm{G}_{t-1}|\mathrm{G}_t,\mathrm{G}_0)$, as well as (iii) the parameterization of $p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)$.

DiGress [Vig+23] applies D3PM's [Aus+21] to define these three terms. Since all elements in the forward process are independent as shown in Eq. (6.1), one can verify that the two terms $q(\mathrm{G}_t|\mathrm{G}_0)$ and $q(\mathrm{G}_{t-1}|\mathrm{G}_t,\mathrm{G}_0)$ are in the form of a product of independent distributions on each element. For simplicity, we introduce the formulation for a single element $\mathbf{x}$, with $\mathbf{x}$ being $\mathbf{v}^i$ or $\mathbf{e}^{i,j}$. We assume each discrete random variable $\mathbf{x}_t$ has a categorical distribution, i.e. $\mathbf{x}_t \sim \mathrm{Cat}(\mathbf{x}_t;\boldsymbol{p})$ with $\boldsymbol{p}\in[0,1]^K$ and $\mathbf{1}^\top\boldsymbol{p}=1$. One can verify that $p(\mathbf{x}_t=\boldsymbol{x}_t)=\boldsymbol{x}_t^\top\boldsymbol{p}$, or simply $p(\mathbf{x}_t)=\boldsymbol{x}_t^\top\boldsymbol{p}$. As shown in Hoogeboom, Nielsen, Jaini, Forré, and Welling [Hoo+21] and Austin, Johnson, Ho, Tarlow, and Berg [Aus+21], the forward process with discrete variables $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ can be represented as a transition matrix $Q_t\in[0,1]^{K\times K}$ such that $[Q_t]_{ij}=q(\mathbf{x}_t=\boldsymbol{e}_j|\mathbf{x}_{t-1}=\boldsymbol{e}_i)$. Then, we can write the distribution explicitly as

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathrm{Cat}(\mathbf{x}_t; Q_t^\top\boldsymbol{x}_{t-1})\ . \tag{6.8}$$

Given transition matrices $Q_1,...,Q_T$, we can get

$$(i)\ q(\mathbf{x}_t|\mathbf{x}_0)=\mathrm{Cat}(\mathbf{x}_t;\overline{Q}_t^\top\boldsymbol{x}_0), \text{with } \overline{Q}_t=Q_1...Q_t\ , \tag{6.9}$$

and the $(t-1)$-step posterior distribution as

$$(ii)\quad q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)=\mathrm{Cat}(\mathbf{x}_{t-1};\frac{Q_t\boldsymbol{x}_t\odot\overline{Q}_{t-1}^\top\boldsymbol{x}_0}{\boldsymbol{x}_t^\top\overline{Q}_t^\top\boldsymbol{x}_0})\ . \tag{6.10}$$

Proof. First, define $\overline{Q}_{t|s}=Q_{s+1}...Q_t$. Note that $\overline{Q}_{t|0}=\overline{Q}_t$ and $\overline{Q}_{t|t-1}=Q_t$. Accordingly, we can derive the following two equalities.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1})=\mathrm{Cat}(\mathbf{x}_t;\overline{Q}_t^\top\mathbf{x}_{t-1}) \tag{6.11}$$

$$\begin{aligned} q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) &= \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} = \frac{\mathrm{Cat}(\mathbf{x}_t;Q_t^\top\mathbf{x}_{t-1})\mathrm{Cat}(\mathbf{x}_{t-1};\overline{Q}_{t-1}^\top\mathbf{x}_0)}{\mathrm{Cat}(\mathbf{x}_t;\overline{Q}_t^\top\mathbf{x}_0)} \\ &= \frac{\mathbf{x}_{t-1}^\top Q_t\mathbf{x}_t\cdot\mathbf{x}_{t-1}^\top\overline{Q}_{t-1}^\top\mathbf{x}_0}{\mathbf{x}_t^\top\overline{Q}_t^\top\mathbf{x}_0} = \mathbf{x}_{t-1}^\top\frac{Q_t\mathbf{x}_t\odot\overline{Q}_{t-1}^\top\mathbf{x}_0}{\mathbf{x}_t^\top\overline{Q}_t^\top\mathbf{x}_0} = \mathrm{Cat}(\mathbf{x}_{t-1};\frac{Q_t\mathbf{x}_t\odot\overline{Q}_{t-1}^\top\mathbf{x}_0}{\mathbf{x}_t^\top\overline{Q}_t^\top\mathbf{x}_0}) \end{aligned} \tag{6.12}$$

□

We have the option to specify node- or edge-specific quantities, $Q_t^{v,i}$ and $Q_t^{e,i,j}$, respectively, or allow all nodes and edges to share a common Q_t^v and Q_t^e. Leveraging Eq. (6.9) and Eq. (6.10), we can precisely determine $q(\mathbf{v}^i_t|\mathbf{v}^i_0)$ and $q(\mathbf{v}^i_{t-1}|\mathbf{v}^i_t,\mathbf{v}^i_0)$ for every node, and a similar approach can be applied for the edges. To ensure simplicity and a non-informative $q(\mathrm{G}_T|\mathrm{G}_0)$, we choose

$$Q_t = \alpha_t I + (1-\alpha_t)\mathbf{1}\boldsymbol{m}^\top \tag{6.13}$$

for all nodes and edges, where $\alpha_t \in [0,1]$, and $\boldsymbol{m}$ is the probability of a uniform distribution ($\mathbf{1}/K_v$ for nodes and $\mathbf{1}/K_e$ for edges). Note that DiGress [Vig+23] chooses $\boldsymbol{m}$ as the marginal distribution of nodes and edges.

As $p(\mathbf{x}_{t-1}|\mathbf{x}_t) = \sum_{\mathbf{x}_0} q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)p(\mathbf{x}_0|\mathbf{x}_t)$, the parameterization of $p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)$ can use the relationship, with

$$(iii) \quad p_\theta(\mathbf{x}_{t-1}|\mathrm{G}_t) = \sum_{\mathbf{x}_0} q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)p_\theta(\mathbf{x}_0|\mathrm{G}_t) \tag{6.14}$$

where $\mathbf{x}$ can be any $\mathbf{v}^i$ or $\mathbf{e}^{i,j}$. With Eq. (6.14), we can parameterize $p_\theta(\mathbf{x}_0|\mathrm{G})$ directly with a neural network, and compute the negative VLB loss in Eq. (6.3) exactly, using Eq.s (6.9), (6.10) and (6.14). In addition, the cross entropy loss between $q(\mathbf{x}_t|\mathbf{x}_0)$ and $p_\theta(\mathbf{x}_0|\mathrm{G}_t)$ that quantifies reconstruction quality is often employed as an auxiliary loss as

$$\mathcal{L}_t^{CE}(\theta) = -\mathbb{E}_{q(\mathrm{G}_t|\mathrm{G}_0)}\Big[\sum_i \log p_\theta(\mathbf{v}_0^i|\mathrm{G}_t) + \sum_{i,j} \log p_\theta(\mathbf{e}_0^{i,j}|\mathrm{G}_t)\Big] . \tag{6.15}$$

In fact, DiGress solely uses $\mathcal{L}_t^{CE}(\theta)$ to train their diffusion model. In this paper, we adapt a hybrid loss [Aus+21], i.e. $\mathcal{L}_t(\theta) + \lambda\mathcal{L}_t^{CE}(\theta)$ with $\lambda = 0.1$, as we found it to help reduce overfitting. To generate a graph from $p_\theta(\mathrm{G}_0)$, a pure noise graph is first sampled from $p_\theta(\mathrm{G}_T)$ and gradually denoised using the learned $p_\theta(\mathrm{G}_{t-1}|\mathrm{G}_t)$ from step T to 0.

A significant advantage of diffusion models is their ability to achieve exchangeable probability in combination with permutation equivariant networks under certain conditions [Xu+22]. DiGress is the first work that applies discrete denoising diffusion to graph generation, and achieves significant improvement over previous continuous-state based diffusion. However, given the inherently high-dimensional nature of graphs and their complex internal dependencies, modeling the joint distribution of all nodes and edges directly presents significant challenges. DiGress requires thousands of denoising steps to accurately capture the original dependencies. Moreover, DiGress relies on many supplementary features, such as cycle counts and eigenvectors, to effectively break symmetries among structural equivalences to achieve high performance.

6.3.2 Autoregressive Graph Generation

Order is important for AR models. Unlike diffusion models that aim to capture the joint distribution directly, AR models decompose the joint probability into a product of simpler conditional probabilities based on an order. This makes AR models inherently suitable for ordinal data, where a natural order exists, such as in natural languages and images.

Order Sensitivity. Early works of graph generation contain many AR models like GraphRNN [You+18] and GRAN [Lia+19] based on non-deterministic heuristic node orders like BFS/DFS and k-core ordering. Despite being permutation sensitive, AR models achieve SOTA performance on small simulated structures like grid and lobster graphs. However, permutation invariance is necessary for estimating an accurate likelihood probability of a graph, and can benefit large-size datasets for better generalization.

Let π denote an ordering of nodes. To make AR order-insensitive, there are two directions: (1) Modeling the joint probability $p(\mathrm{G}, \pi)$ and then marginalizing π, (2) Finding a unique canonical order $\pi^*(G)$ for any graph G such that $p(\pi|\mathrm{G}) = 1$ if $\pi = \pi^*(G)$ and 0 otherwise. In direction (1), directly integrating out π is prohibitive as the number of permutations is factorial in the graph size. Several studies [Li+18; Che+21; Lia+19] have investigated the use of subsets of either random or canonical orderings. This approach aims to simplify the process, but it results in approximate integrals with indeterminate errors. Moreover, it escalates computational expense due to the need for data augmentation involving these subsets of orderings. In direction (2), identifying a universal canonical order for all graphs is referred to as graph canonicalization. This task is recognized to be at least as challenging as the Graph Isomorphism problem, which is classified as NP-intermediate. Goyal, Jain, and Ranu [GJR20] explores using minimum DFS code to construct canonical labels for a specific dataset with non-polynomial time complexity. However, the canonicalization is specific to each training dataset with the randomness derived from DFS. This results in the canonical order being $\pi(G|\text{TrainSet})$ instead of $\pi(G)$, which exhibits the generalization issue.

The Existence of Partial Order. While finding a unique order for all nodes of a graph is NP-intermediate, we argue that finding a unique *partial* order, where certain nodes and edges are with the same rank, is achievable. For example, a trivial partial order is simply all nodes and edges having the same rank. Nevertheless, a graph is not the same as a set (a set is just a graph with empty $\mathcal{E}$), where all elements are essentially unordered with equivalent rank. That is because a non-empty graph contains edges between nodes, and these edges give different structural properties to nodes. Notice that some nodes or edges have the same structural property as they are structurally equivalent. We can view each structural property as a color, and rank all unique colors within the graph to define the partial order over nodes, which we call a *structural partial order*. The structural partial order defines a sequence of *blocks* such that all nodes within a block have the same rank (i.e. color).

Let $\phi : \mathcal{V} \to [1, ..., |\mathcal{S}|]$ be the function that assigns rank to nodes based on their structural properties. When $\mathcal{S} \subseteq \mathcal{V}$, we use $G[\mathcal{S}]$ to denote the induced subgraph on the subset $\mathcal{S}$. There are many ways to assign rank to structural colors, however we would like the resulting partial order to satisfy certain constraints. Most importantly, we want

$$\forall r \in [1, ..., |\mathcal{S}|],\ G[\phi(\mathcal{V}) \leq r] \text{ is a connected graph.} \tag{6.16}$$

The connectivity requirement is to ensure a more accurate representation of real-world graph generation processes, where most real-world dynamic graphs are enlarged with newcoming nodes being connected at any time. Then, *one can sequentially remove all nodes with the lowest degree to maintain this connectivity and establish a partial order.* However, as degree only reflects information of the first-hop neighbor structure, many nodes share the same degree, leading to only a few distinct blocks, not significantly different from a trivial, single-block approach.

To ensure connectivity while reducing rank collision, we consider larger hops to define a weighted degree. Consider a maximum of K_h hops. For any node $\boldsymbol{v} \in \mathcal{V}$, the number of neighbors at each hop of $\boldsymbol{v}$ can be easily obtained as $[d_1(\boldsymbol{v}), ..., d_{K_h}(\boldsymbol{v})]$. We then define the weighted degree as

$$w_{K_h}(\boldsymbol{v}) = \sum_{k=1}^{K_h} d_k(\boldsymbol{v}) \times |\mathcal{V}|^{K_h - k} \tag{6.17}$$

Eq. (6.17) is fast to compute, and ensures that 1) nodes have the same rank if and only if they have the same number of neighbors up to K_h hops; and 2) lower-hop degrees are weighted higher such that nodes with smaller lower-hop degree have smaller w_{K_h}. With w_{K_h} defined, we give the structural partial order in Algo. 5. It is important to note that for any G, its structural partial order is unique, deterministic, and

Algorithm 5 Structural Partial Order ϕ

```
1: Input: Graph $G$, maximum hops $K_h$.
2: Init: $G_0 = G$, $i = 0$, $\phi$ with $\phi(\boldsymbol{v}) = 0\ \forall \boldsymbol{v}$.
3: while $G_i$ is not $\emptyset$ do
4:     Compute $w_{K_h}(\boldsymbol{v}), \forall \boldsymbol{v} \in \mathcal{V}(G_i)$, using Eq. (6.17).
5:     Find all nodes $\mathcal{L}$ with $w_{K_h} = \min_{\boldsymbol{v}\in\mathcal{V}(G)} w_{K_h}(\boldsymbol{v})$.
6:     Let $\phi(\boldsymbol{v}) = i\ \forall \boldsymbol{v} \in \mathcal{L}$.
7:     $G_{i+1} \leftarrow G_i[\mathcal{V}(G) \setminus \mathcal{L}]$; $i \leftarrow i + 1$.
8: end while
9: Output: $\phi \leftarrow i - \phi$
```

permutation equivariant. Formally, let $\boldsymbol{P}$ be any permutation operator, then

$$\phi(\boldsymbol{P} \star G) = \boldsymbol{P} \star \phi(G) \tag{6.18}$$

Autoregressive Blockwise Generation. The structural partial order ϕ of G in Algo. 5 with output in range $[1, K_B]$ divides the nodes $\mathcal{V}(G)$ into K_B blocks $[\mathcal{B}_1, ..., \mathcal{B}_{K_B}]$ in order. Let $\mathcal{B}_{1:i} := \cup_{j=1}^{i} \mathcal{B}_j$ be the union of the first i blocks. PARD decomposes the joint probability of a graph G into

$$p_\theta(G) = \prod_{i=1}^{K_B} p_\theta\Big(G[\mathcal{B}_{1:i}] \setminus G[\mathcal{B}_{1:i-1}] \;\Big|\; G[\mathcal{B}_{1:i-1}]\Big) \tag{6.19}$$

where $G[\mathcal{B}_{1:0}]$ is defined as the empty graph, and $G[\mathcal{B}_{1:i}] \setminus G[\mathcal{B}_{1:i-1}]$ denotes the set of nodes and edges that are present in $G[\mathcal{B}_{1:i}]$ but not in $G[\mathcal{B}_{1:i-1}]$. As each conditional probability only contains a subset of edges and nodes, and having access to all previous blocks, this conditional probability is significantly easier to model than the whole joint probability. Given the permutation equivariant property Eq. (6.18) of $\mathcal{B}_i$, it is easy to verify that $p_\theta(G)$ is exchangeable with permutation-invariant probability for any G if and only if all conditional probabilities are exchangeable.

6.3.3 Impossibility of Equivariant Graph Transformation

With Eq. (6.19), we need to parameterize the conditional probability $p_\theta\Big(G[\mathcal{B}_{1:i}] \setminus G[\mathcal{B}_{1:i-1}] \;\Big|\; G[\mathcal{B}_{1:i-1}]\Big)$ to be permutation-invariant. This can be achieved by letting the conditional probability be

$$p_\theta\Big(|\mathcal{B}_i| \;\Big|\; G[\mathcal{B}_{1:i-1}]\Big) \prod_{\substack{\mathbf{x}\in G[\mathcal{B}_{1:i}]\setminus \\ G[\mathcal{B}_{1:i-1}]}} p_\theta\Big(\mathbf{x}\Big|\; G[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]\Big) \tag{6.20}$$

where $\mathbf{x}$ is any node and edge in $G[\mathcal{B}_{1:i}] \setminus G[\mathcal{B}_{1:i-1}]$, $\emptyset$ denotes an empty graph, hence $G[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]$ depicts augmenting $G[\mathcal{B}_{1:i-1}]$ with empty (or virtual) nodes and edges to the same size as $G[\mathcal{B}_{1:i}]$. With the augmented graph, we can parameterize $p_\theta\big(\mathbf{x} \mid G[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]\big)$ for any node and edge $\mathbf{x}$ with a permutation equivariant network to achieve the required permutation invariance. For simplicity, let $G\big[\mathcal{B}_{1:i-1}, |\mathcal{B}_i|\big] := G[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]$.

The Flaw in Equivariant Modeling. Although the parameterization in Eq. (6.20) along with an equivariant network makes the conditional probability in Eq. (6.19) become permutation-invariant, we have found that the equivariant graph transformation $p_\theta(\mathbf{x} \mid G\big[\mathcal{B}_{1:i-1}, |\mathcal{B}_i|\big])$ cannot be achieved in general for *any* permutation equivariant network, no matter how powerful it is (!) The underlying cause is the symmetry of structural equivalence, which is also a problem in link prediction [SR20; Zha+21]. Formally, let $\boldsymbol{A}(G)$ be the adjacency matrix representation of G (ignoring labels) based on G's default node order, then an *automorphism* σ of G satisfies

$$\boldsymbol{A}(G) = \boldsymbol{A}(\sigma \star G) \tag{6.21}$$

where $\sigma \star G$ represents a reordering of nodes of G based on the mapping σ. Then the automorphism group $\mathrm{Aut}(G)$ is

$$\mathrm{Aut}(G) = \{\sigma \in \mathbb{P}_{|\mathcal{V}|} \mid \boldsymbol{A}(G) = \boldsymbol{A}(\sigma \star G)\} \tag{6.22}$$

where $\mathbb{P}_n$ denotes all permutation mappings for size n. That is, $\mathrm{Aut}(G)$ contains all automorphisms of G. For a node i of G, the *orbit* that contains node i is defined as

$$o(i) = \{\sigma(i) \mid \forall \sigma \in \mathrm{Aut}(G)\} \, . \tag{6.23}$$

In words, the orbit $o(i)$ contains all nodes that are *structurally equivalent* to node i in G. We say that two edges (i, j) and (u, v) are structurally equivalent if $\exists \sigma \in \mathrm{Aut}(G)$, such that $\sigma(i) = u$ and $\sigma(j) = v$.

Theorem 6.3.1. *Any structurally equivalent nodes and edges have the same representation for any equivariant network.*

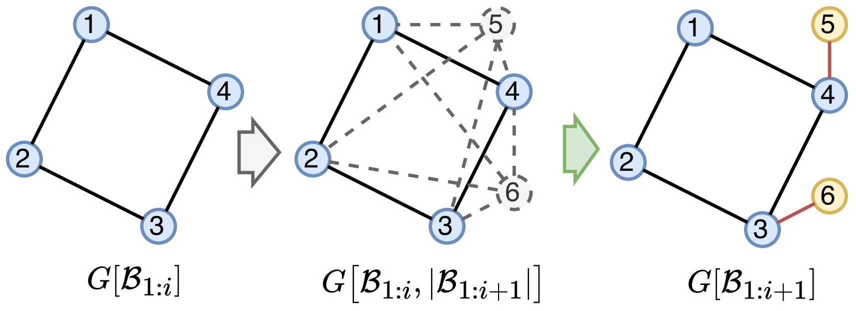


FIGURE 6.1: Example case where the equivariant graph transformation from $G[\mathcal{B}_{1:i}]$ to $G[\mathcal{B}_{1:i+1}]$ is impossible.

Proof. We prove this for node case. For structurally equivalent edges, the analysis is the same. Assume node i and node j are structually equivalent, then we can find an automorphism $\sigma \in \mathrm{Aut}(G)$ such that $\sigma(i) = j$. For any permutation $\boldsymbol{P} \in \mathbb{P}_{|G|}$ and an equivariant network f, we have

$$f(\boldsymbol{P} \star G) = \boldsymbol{P} \star f(G) \tag{6.24}$$

Replace $\boldsymbol{P}$ with σ, and using the fact that $\sigma \star G = G$. We can get

$$f(G) = f(\sigma \star G) = \sigma \star f(G) \tag{6.25}$$

Hence, we get $f(G)_i = f(G)_j$, that is two nodes i and j have the same representation. □

Theorem 6.3.1 indicates that no matter how powerful the equivariant network is, any structually equivalent elements have the same representation. Based on this theorem, we can easily show that there are many "bottleneck" cases where the targeted graph transformation cannot be achieved. Figure 6.1 shows a case where $G[\mathcal{B}_{1:}]$ is a 4-cycle, and the next target block contains two additional nodes, each with a single edge connecting to one of the nodes of $G[\mathcal{B}_{1:}]$. It is easy to see that nodes 1–4 are all structurally equivalent, and so are nodes $5, 6$ in the augmented case (middle). Hence, edges in $\{(5, i) | \forall i \in [1, 4]\}$ are structurally equivalent (also $\{(6, i) | \forall i \in [1, 4]\}$). Similarly, $\forall i \in [1, 4]$, edge $(5, i)$ and $(6, i)$ are structurally equivalent. Combining all cases, edges in $\{(j, i) | \forall i \in [1, 4], j \in \{5, 6\}\}$ are structurally equivalent. Theorem 6.3.1

states that all these edges would have the same prediction with any equivariant model, hence making the target $G[\mathcal{B}_{1:i+1}]$ not achievable.

The Magic of Annealing/Randomness. In Figure 6.1 we showed that a graph with many automorphisms cannot be transformed to a target graph with fewer automorphisms. We hypothesize that *a graph with lower "energy" is hard to be transformed to a graph with higher "energy" with equivariant networks.* There exist some definitions and discussion of graph energy [GLZ09; Bal04] based on symmetry and eigen-information to measure graph complexity, where graphs with more symmetries have lower energy. The theoretical characterization of the conditions for successful graph transformation is a valuable direction, which we leave for future work to investigate.

Based on the above hypothesis, to achieve a successful transformation of a graph into a target graph, it is necessary to increase its energy. Since graphs with fewer symmetries exhibit higher energy levels, our approach involves introducing random noise to both nodes and edges. Our approach of elevating the energy level, followed by its reduction to attain desired target properties, mirrors the annealing process.

Diffusion. This further motivates us to use denoising diffusion to model $p_\theta(\mathbf{x} \mid G\big[\mathcal{B}_{1:i-1}, |\mathcal{B}_i|\big])$: it naturally injects noise in the forward process, and its backward denoising process is the same as annealing. What is more, we can achieve the permutation-invariant property for $p_\theta\Big(G[\mathcal{B}_{1:i}]\backslash G[\mathcal{B}_{1:i-1}] \Bigm| G[\mathcal{B}_{1:i-1}]\Big)$, based on Proposition 1 in [Xu+22]. Finally, as we have analyzed, this yields $p_\theta(G)$ in Eq. (6.19) to be permutation-invariant.

6.3.4 PARD: Autoregressive Denoising Diffusion

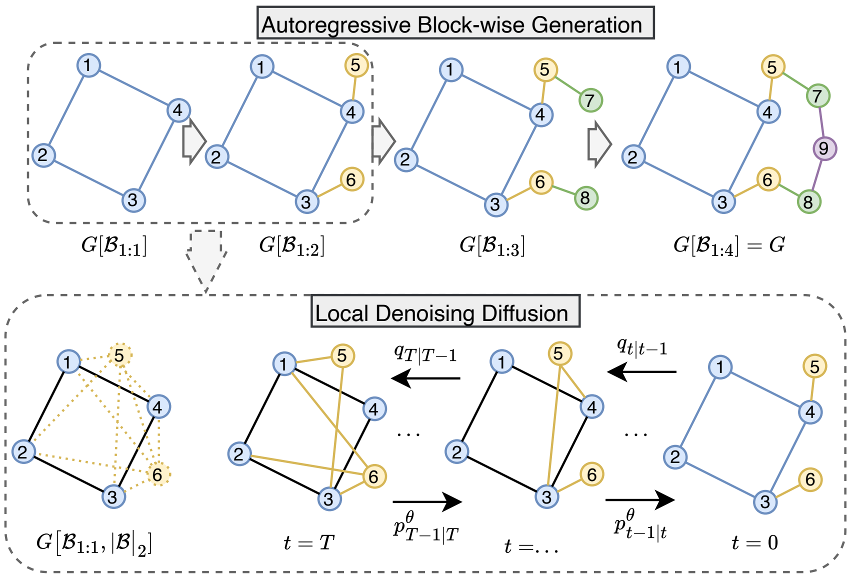


FIGURE 6.2: PARD integrates the autoregressive method with diffusion modeling. (top) PARD decomposes the joint probability into a series of block-wise enlargements, where each block's conditional distribution is captured with a shared discrete diffusion (bottom).

To summarize, we present PARD, the first permutation-invariant autoregressive diffusion model that integrates AR with denoising diffusion. PARD relies on a unique, permutation equivariant structural partial order ϕ (Algo. 5) to decompose the joint graph probability to the product of simpler conditional probabilities, based on Eq. (6.19). Each block's conditional probability is modeled with the product of a conditional block size probability and a conditional block enlargement probability as in Eq. (6.20), where the conditional block enlargement probability for every block is a shared discrete denoising diffusion model as described in §6.3.1. Figure 6.2 illustrates PARD's two parts : block-wise AR and local denoising diffusion at each AR step.

Notice that there are two tasks in Eq. (6.20); one for predicting the next block's size, and the other for predicting the next block's nodes and edges with diffusion. These two tasks can be trained together with a single network, although for better performance we use two different networks. For each block's diffusion model, we set the maximum time steps to 40 without much tuning.

Training and Inference Algorithm. We provide the training and inference algorithms for PARD. Specifically, Algo. 6 is used to train each next block's size prediction model; Algo. 7 is used to train the shared diffusion for block conditional probabilities; and Algo. 8 presents the generation steps.

Algorithm 6 Train blocksize distribution $p_\theta\big(|\mathcal{B}_i| \bigm| G[\mathcal{B}_{1:i-1}]\big)$

```
1: Input: $G$, maximum hop $K_h$, a network $f_\theta$ that takes a graph as input and output graph wise prediction.
2: Get structural partial order function $\phi$ of $G$ from Algo.5.
3: Using $\phi$ to get the sequence of node blocks $[\mathcal{B}_1, ..., \mathcal{B}_{K_B}]$ for $G$.
4: Minimize $\sum_{i=1}^{K_B} \text{CrossEntropy}(f_\theta(G[\mathcal{B}_i]), |\mathcal{B}_{i+1}|)$, with $|\mathcal{B}_{K_B+1}| = 0$
```

Algorithm 7 Train denoising diffusion for distribution $p_\theta\Big(G[\mathcal{B}_{1:i}] \setminus G[\mathcal{B}_{1:i-1}] \Bigm| G[\mathcal{B}_{1:i-1}] \cup \emptyset[\mathcal{B}_{1:i}]\Big)$

```
1: Input: $G$, max time T, maximum hop $K_h$, a network $f_\theta$ that inputs a graph and outputs nodes and edges predictions.
2: Get structural partial order function $\phi$ of $G$ from Algo.5.
3: Using $\phi$ to get the sequence of node blocks $[\mathcal{B}_1, ..., \mathcal{B}_{K_B}]$ for $G$.
4: Sample $t \sim U(1, ..., T)$
5: for $i = 1, ..., K_B$ do
6:     $M \leftarrow$ indice mask of $G[\mathcal{B}_{1:i}] \setminus G[\mathcal{B}_{1:i-1}]$
7:     Sample a noise graph $\tilde{G}[\mathcal{B}_{1:i}]$ from $q_{t|0}(G[\mathcal{B}_{1:i}])$ according to Eq. (6.9)
8:     $\tilde{G}[\mathcal{B}_{1:i}] \leftarrow M \odot \tilde{G}[\mathcal{B}_{1:i}] + (1 - M) \odot G[\mathcal{B}_{1:i}]$
9:     $X \leftarrow f_\theta(\tilde{G}[\mathcal{B}_{1:i}]) \odot M$
10:    $Y \leftarrow G[\mathcal{B}_{1:i}] \odot M$
11:    $l_i \leftarrow \mathcal{L}_t(X, Y) + 0.1 * \mathcal{L}_t^{CE}(X, Y)$, using $\mathcal{L}_t$ in Eq. (5.3) and $\mathcal{L}_t^{CE}$ in Eq. (6.15).
12: end for
13: Minimize $\sum_{i=1}^{K_B} l_i$.        (The for loop can be parallelized. )
```

Algorithm 8 Generation

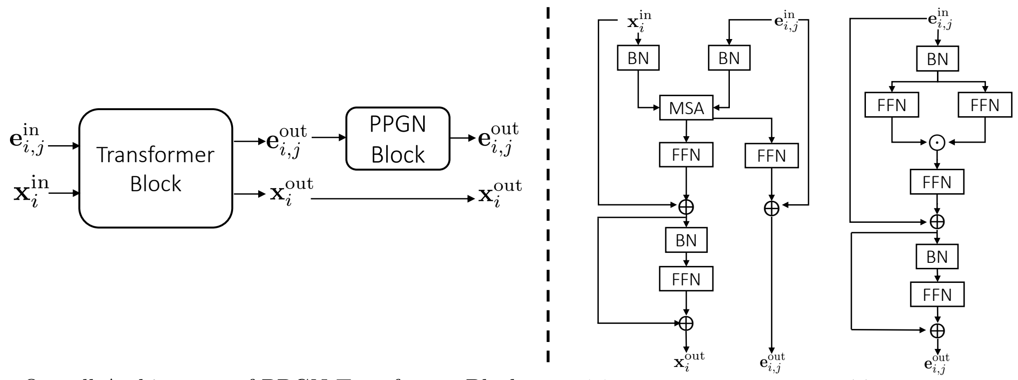
```
1: Input: blocksize model $g_\varphi$, diffusion model $f_\theta$; first blocksize distribution from TrainSet.
2: $G \leftarrow \emptyset$; $i \leftarrow 1$
3: Sample $n$ from the first block's size distribution.
4: while $n > 0$ do
5:    Add a new block $\mathcal{B}_i$ with $n$ nodes into $G$
6:    $M \leftarrow$ indice mask of $G[\mathcal{B}_{1:i}] \setminus G[\mathcal{B}_{1:i-1}]$
7:    $\tilde{G} \leftarrow$ For nodes and edges within $M$, sample from noise $\boldsymbol{m}_n$ and $\boldsymbol{m}_e$.
8:    for $j = 1 : T$ do
9:       $\mathrm{p} \leftarrow f_\theta(\tilde{G})$
10:      $S \leftarrow$ Sample according to p
11:      $\tilde{G} \leftarrow M \odot S + (1 - M) \odot \tilde{G}$
12:   end for
13:   $G \leftarrow \tilde{G}$
14:   $n \leftarrow$ Sample from $g_\varphi(G)$
15:   $i \leftarrow i + 1$
16: end while
17: Return: $G$
```

6.4 Architecture Improvement

Pard is a general framework that can be combined with any equivariant network. Nevertheless, we would like an equivariant network with enough expressiveness to process symmetries inside the generated blocks for modeling the next block's conditional probability. While there are many expressive GNNs like subgraph GNNs [Bev+22; Zha+22c] and higher-order GNNs [ZSA22; Mor+22], PPGN [Mar+19a] is still a natural choice that models edge (2-tuple) representations directly with 3-WL expressivity and $O(n^3)$ complexity in graph size. However, PPGN's memory cost is relatively high for many datasets.

6.4.1 Efficient and Expressive Higher-order Transformer

To enhance the memory efficiency of Probabilistic Graph Neural Networks (PPGN) while maintaining the expressiveness equivalent to the 3-Weisfeiler-Lehman (3-WL) test, we introduce a hybrid approach that integrates Graph Transformers with PPGN. Graph Transformers operate on nodes as the fundamental units of representation, offering better scalability and reduced memory consumption compared to PPGN, while utilizing edges as their primary representation units and therefore incur significantly higher memory requirements. However, the expressiveness of Graph Transformers (without position encoding) is limited to the 1-WL test [Cai+23]. By combining these two models, we can drastically decrease the size of edge representations while allocating larger hidden sizes to nodes. This synergistic approach not only substantially lowers the memory footprint but also enhances overall performance, leveraging the strengths of both architectures to achieve a balance between expressivity and efficiency. We provide the detailed design in Fig. 6.3. Note that we use GRIT [Ma+23] as the transformer block.


Transformer Block
PPGN Block
BN
MSA
FFN
$\mathbf{e}_{i,j}^{\text{in}}$
$\mathbf{x}_i^{\text{in}}$
$\mathbf{e}_{i,j}^{\text{out}}$
$\mathbf{x}_i^{\text{out}}$

(a) The Overall Architecture of PPGN Transformer Block (b) Transformer Block (c) PPGN Block

FIGURE 6.3: The Architecture of the PPGN-Transformer Block. In (b) and (c) we provide illustrations of how edge and node features are processed through Transformer and PPGN blocks.

6.4.2 Parallel Training with Causal Transformer

As shown in Eq. (6.19), for a graph G, there are K_B conditional probabilities $p_\theta\Big(G[\mathcal{B}_{1:i}] \setminus G[\mathcal{B}_{1:i-1}] \;\Big|\; G[\mathcal{B}_{1:i-1}]\Big)$ being modeled by a shared diffusion model f_θ. By default, these K_B number of inputs $\{G[\mathcal{B}_{1:i-1}]\}_{i=1}^{K_B}$ are viewed as separate graphs and the network passing $f_\theta(G[\mathcal{B}_{1:i-1}])$ for different $i \in [1, K_B]$ are not shared. This leads to a scalability issue; in effect enlarging the dataset by roughly K_B times and resulting in K_B times longer training.

To minimize computational overhead, it is crucial to enable parallel training of all the K_B conditional probabilities, and allow these processes to share representations, through which we can pass the full graph G to the network f_θ only once and obtain all K_B conditional probabilities. This is also a key advantage of transformers over RNNs. Transformers (GPTs) can train all next-token predictions simultaneously with representation sharing through causal masking, whereas RNNs must train sequentially. However, the default causal masking of GPTs is not applicable to our architecture as it contains both Transformer and PPGN.

To ensure representation sharing without risking information leakage, we first assign a "block ID" to every node and edge within graph G. Specifically, for every node and edge in $G[\mathcal{B}_{1:i}] \setminus G[\mathcal{B}_{1:i-1}]$, we assign the ID equal to i. To prevent information leakage effectively, it is crucial that any node and edge labeled with ID i are restricted to communicate only with other nodes and edges whose ID is $\leq i$. Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$, and $\mathbf{x} \in \mathbb{R}^n$. There are mainly two non-elementwise operations in Transformer and PPGN that have the risk of leakage: the attention-vector product operation $\boldsymbol{A}\mathbf{x}$ of Transformer, and the matrix-matrix product operation $\boldsymbol{AB}$ of PPGN. (We ignore the d dimension of $\boldsymbol{A}$ and $\mathbf{x}$ as it does not affect the information leakage.)

Let $\boldsymbol{M} \in \{0,1\}^{n\times n}$ be a mask matrix, such that $\boldsymbol{M}_{i,j} = 1$ if block_ID$(i) \geq$ block_ID(j) else 0. One can verify that

$$\begin{gathered}(\boldsymbol{A} \odot \boldsymbol{M})\mathbf{x} \\ (\boldsymbol{A} \odot \boldsymbol{M})\boldsymbol{B} + \boldsymbol{A}(\boldsymbol{B} \odot \boldsymbol{M}^\top) - (\boldsymbol{A} \odot \boldsymbol{M})(\boldsymbol{B} \odot \boldsymbol{M}^\top)\end{gathered} \tag{6.26}$$

generalize $\boldsymbol{A}\mathbf{x}$ and $\boldsymbol{AB}$ respectively and safely bypass information leakage. We use these operations in our network and enable representation sharing, along with parallel training of all K_B blocks for denoising diffusion as well as next block size prediction. In practice, these offer more than 10× speed-up, and the parallel training allows PARD to scale to large datasets like MOSES [Pol+20].

TABLE 6.1: Generation quality on QM9 with explicit hydrogens.

Model	**Valid.** ↑	**Uni.** ↑	**Atom.**↑	**Mol.** ↑
Dataset (optimal)	97.8	100	98.5	87.0
ConGress	86.7	**98.4**	97.2	69.5
DiGress (uniform)	89.8	97.8	97.3	70.5
DiGress (marginal)	92.3	97.9	97.3	66.8
DiGress (marg. + *feat.*)	95.4	97.6	98.1	79.8
PARD (*no feat.*)	**97.5**	95.8	**98.4**	**86.1**

6.5 Experiments

We evaluate PARD on 7 diverse benchmark datasets with varying sizes and structural properties, including both molecular (§6.5.1) and non-molecular/generic (§6.5.2) graph generation. A summary of the datasets is shown in Table 1.1.

6.5.1 Molecular Graph Generation

Datasets. We experiment with three different molecular datasets used across the graph generation literature: (1) QM9 [Ram+14] (2) ZINC250K [Irw+12], and (3) MOSES [Pol+20] that contains more than 1.9 million graphs. We use a 80%-20% train and test split, and among the train data we split additional 20% as validation. For QM9 and ZINC250K, we generate 10,000 molecules for stand-alone evaluation, and on MOSES we generate 25,000 molecules.

Baselines. The literature has not been consistent in evaluating molecule generation on well-adopted benchmark datasets and metrics. Among baselines, DiGress [Vig+23] stands out as the most competitive. We also compare to a list of many other baselines, where we report their performance values as sourced from the literature.

Metrics. The literature has adopted a number of different evaluation metrics that are not consistent across datasets. Most common ones include Validity (↑) (fraction of valid molecules without valency correction), Uniqueness (↑) (frac. of valid molecules that are unique), and Novelty (↑) (frac. of valid molecules that are not included in the training set).

For QM9, following earlier work [Vig+23], we report additional evaluations w.r.t. Atom Stability (↑) and Molecule Stability (↑), as defined by [Hoo+22b], whereas Novelty is not reported since QM9 contains small molecules that meet specific constraints, and generating novel molecules does not mean the network has accurately learned the data distribution.

On ZINC250K and MOSES, we also measure the Fréchet ChemNet Distance (FCD) (↓) between the generated and the training samples, which is based on the embedding learned by ChemNet [Li+18]. For MOSES, there are three additional measures: Filter (↑) score is the fraction of molecules passing the same filters as the test set, SNN (↑) evaluates nearest neighbor similarity using Tanimoto Distance, and Scaffold similarity (↑) analyzes the occurrence of Bemis-Murcko scaffolds [Pol+20].

Results. Table 6.1 shows generation evaluation results on QM9, where the baseline results are sourced from [Vig+23]. PARD outperforms DiGress and variants that do *not* use any auxiliary features, in terms of Atom Stability and especially Validity and Molecule Stability, with slightly lower Uniqueness. What is notable is that PARD, without using any extra features, achieves a similar performance gap against DiGress that uses specialized extra features.

Table 6.2 shows PARD's performance on ZINC250K, with baseline results carried over from [Kon+23] and [Yan+23]. PARD achieves the best Uniqueness, stands out in FCD alongside SwinGNN [Yan+23], and is the runner-up w.r.t. Validity.

TABLE 6.2: Generation quality on ZINC250K.

Model	Validity ↑	FCD ↓	Uni. ↑	Model Size
EDP-GNN	82.97	16.74	99.79	0.09M
GraphEBM	5.29	35.47	98.79	-
SPECTRE	90.20	18.44	67.05	-
GDSS	**97.01**	14.66	99.64	0.37M
GraphArm	88.23	16.26	99.46	-
DiGress	91.02	23.06	81.23	18.43M
SwinGNN-L	90.68	1.99	99.73	35.91M
PARD	95.23	**1.98**	**99.99**	4.1M

Finally, Table 6.3 shows generation quality of PARD on the largest dataset MOSES. We mainly compare with DiGress and its variant ConGress, which has been the only general-purpose generative model in the literature that is not based on molecular fragments or SMILES strings. All baseline performances are sourced from [Vig+23].

TABLE 6.3: Generation quality on MOSES. The top three methods use hard-coded rules, hence we do not highlight them.

Model	Val. ↑	Uni. ↑	Novel. ↑	Filters ↑	FCD ↓	SNN ↑	Scaf. ↑
VAE	97.7	99.8	69.5	99.7	0.57	0.58	5.9
JT-VAE	100	100	99.9	97.8	1.00	0.53	10.0
GraphINVENT	96.4	99.8	-	95.0	1.22	0.54	12.7
ConGress	83.4	99.9	**96.4**	94.8	1.48	0.50	**16.4**
DiGress	85.7	**100**	95.0	97.1	1.19	0.52	14.8
PARD	**86.8**	**100**	78.2	**99.0**	**1.00**	**0.56**	2.2

While the specialized models, excluding PARD and DiGress, have hard-coded rules to ensure high Validity, PARD outperforms those on several other metrics including FCD and SNN, and achives competitive performance on others. Again, it is notable here that PARD, *without* relying on any auxiliary features, achieves similarly competitive results as with DiGress which utilizes extra features.

6.5.2 Generic Graph Generation

Datasets. We use four generic graph datasets with various structure and semantic: (1) COMMUNITY-SMALL [You+18], (2) CAVEMAN [You18], (3) CORA [Sen+08], and (4) BREAST [GM+11]. We split each dataset into 80%-20% train-test, and randomly sample 20% of training graphs for validation. We generate the same number of samples as the test set.

Baselines. We mainly compare against the latest general-purpose GraphArm [Kon+23], which reported DiGress [Vig+23] and GDSS [JLH22] as top two most competitive, along with several other baselines.

Metrics. As with prior work [You+18], we measure generation quality using the maximum mean discrepancy (MMD) as a distribution distance between the generated graphs and the test graphs (↓), as pertain to distributions of (*i*) Degree, (*ii*) Clustering coefficient, and (*iii*) occurrence count of all Orbits with 4 nodes.

TABLE 6.4: Generation quality on generic graphs. All metrics are based on generated-to-test set MMD distances, the lower the better. Top performance is in **bold**, and Runner-up is underlined.

	COMMUNITY-SMALL			CAVEMAN			CORA			BREAST		
Model	**Deg.**	**Clus.**	**Orbit**	**Deg.**	**Clus.**	**Orbit**	**Deg.**	**Clus.**	**Orbit**	**Deg.**	**Clus.**	**Orbit**
GraphRNN	0.080	0.120	0.040	0.371	1.035	0.033	1.689	0.608	0.308	0.103	0.138	0.005
GRAN	0.060	0.110	0.050	0.043	0.130	0.018	0.125	0.272	0.127	0.073	0.413	0.010
EDP-GNN	0.053	0.144	0.026	0.032	0.168	0.030	0.093	0.269	0.062	0.131	0.038	0.019
GDSS	0.045	0.086	0.007	0.019	0.048	0.006	0.160	0.376	0.187	0.113	**0.020**	0.003
GraphArm	0.034	0.082	**0.004**	0.039	**0.028**	0.018	0.273	0.138	0.105	**0.036**	0.041	0.002
DiGress	0.047	**0.041**	0.026	0.019	0.040	0.003	0.044	0.042	0.223	0.152	0.024	0.008
PARD	**0.023**	0.071	0.012	**0.002**	0.047	**0.00003**	**0.0003**	**0.003**	**0.0097**	0.044	0.024	**0.0003**

Results. Table 6.4 provides the generation results of PARD against the baselines as sourced from [Kon+23]. PARD shows outstanding performance achieving SOTA or close runner-up results, while none of the baselines shows as consistent performance across datasets and metrics.

6.6 Conclusion

We presented PARD, the first permutation-invariant autoregressive diffusion model. PARD decomposes the joint probability of a graph autoregressively into product of several block conditional probabilities relying on a unique and permutation equivariant structural partial order. All conditional probabilities are then modeled with a shared discrete diffusion. PARD can be trained in parallel on all blocks, and efficiently scales to millions of graphs. PARD achieves SOTA performance on molecular and non-molecular datasets without using any extra features. We expect PARD to serve as a cornerstone for generative foundation modeling on graphs.

Part IV

Application: Graph-level Anomaly Detection

Chapter 7

Graph-level Anomaly Detection: Baselines and Issues

Chapter based on: Lingxiao Zhao and Leman Akoglu. "On using classification datasets to evaluate graph outlier detection: Peculiar observations and new insights". In: *Big Data* 11.3 (2023), pp. 151–180.

Outlier detection is a critical task that finds numerous applications in healthcare, security, finance, etc. [Agg15]. Simply put, the task is to identify observations that notably stand out within large collections of data so as to "arouse suspicions that [they were] generated by a different mechanism" [Haw80]. One of the key challenges of outlier detection is that it poses an *unsupervised* learning problem. Due to the rare nature of outlier instances, combined with the laborious manual (i.e., human) labeling, access to benchmark datasets with sufficiently many labeled ground-truth outliers is limited.

Motivation. Lack of labeled benchmark datasets for outlier detection is not only a challenge for learning, but also for the *evaluation* of outlier models. Even if one designs unsupervised models for the detection task, ground-truth labels that truly reflect the nature of outliers in a domain is essential for the reliable error estimation of various models. Thereby, the scarcity of representative labeled outliers in real-world datasets has motivated a couple of strategies for building benchmark datasets, mainly for evaluation.

One strategy is to inject realistic yet synthetic outliers into real-world datasets via simulation. For example, in fraud detection applications, one could simulate activities that reflect malicious schemes known to domain experts in order to obtain positive (i.e., anomalous) observations. Emmott *et al.* [Emm+13; Emm+15] present a systematic in-depth study on this subject. This approach is typically criticized for a couple of reasons. First, the simulated outliers are limited to the known anomalous behaviors and may not comprehensively reflect the outliers in the wild. Second, this type of approach may create an environment fertile to "leakage", where the outliers may be simulated in a biased way that aligns with how the detection model under evaluation works.

An alternative strategy to artificial outlier injection is to repurpose classification datasets so as to work with only real-world samples. (See [Cam+16] and citations therein.) A common practice is to use binary classification datasets, where samples from one of the classes (typically the one with the larger number of samples) is treated as the 'inlier' samples, and the other class is down-sampled (to a desired rate) to constitute the 'outlier' samples. This procedure conforms with the notion of outlierness as characterized by Hawkins [Haw80], in that the outliers are drawn from a data distribution (i.e., class) that is different from that generating the inliers. In their in-depth evaluation of unsupervised outlier detection models, Campos *et al.* [Cam+16] mainly adopt this strategy.

This paper. In this study we scrutinize this latter strategy, and pose the following questions: *Should one use classification datasets for evaluating outlier detection models? What issues should one be aware of in designing benchmark datasets in this manner?* Specifically, we study this issue in the context of outlier detection in *graph databases*, where given a collection of graphs, the task is to identify the outlier graphs that stand out. Graph data is widespread in finance, health care, cybersecurity, fault monitoring, etc. where the outlier detection task finds a long list of applications such as identifying rare transaction graphs [NLA20], command flow graphs [MMA16], and human poses [Mar+20], fake news [Mon+19], traffic events [Har+16], buggy software [Liu+05], money laundering [Web+19], and so on.

Before delving into details, we start by illustrating the intriguing "performance flip" issue empirically. Table 7.1 (See Sec. 7.2.1) shows the ROC-AUC performances of three graph embedding based outlier detectors based on four binary graph classification datasets (Additional results on more datasets, and using more graph embedding methods is available in Tables 7.4&7.5). Each dataset has two variants, each corresponding to one of the classes down-sampled as outlier. The difference in performances between the two variants is striking, consistently across models on most (although not all) datasets.

Related work. To the best of our knowledge, the performance flip issue has not been identified by any prior work on outlier mining, with the exception of work by Swersky *et al.* [Swe+16] which document similar ROC-AUC flip behavior on several datasets, however the authors have not recognized explicitly. Campos *et al.* [Cam+16] state that "random downsampling often leads to great variation in the nature of the outliers produced" and that "observations based on downsampling can vary considerably from sample to sample", based on which they repeat their downsampling procedure 10 times per dataset "to mitigate the impact of randomization". This, however, points to an orthogonal issue as it pertains to down-sampling *after* deciding (i.e. fixing) *which* class to down-sample. Repurposing classification datasets for evaluation of clustering has been questioned by Färber *et al.* [Fär+10], which alludes to the potential misalignment between the semantics of data clusters and class labels. Our study points to an issue orthogonal to semantics.

Contributions. Through extensive analysis, our study aims to (1) illustrate the issues with using graph classification datasets for creating outlier benchmarks for model evaluation, (2) identify the leading factors behind these issues, (3) propose concrete measures to quantify these factors and explain their possible driving mechanisms with a focus on propagation-based graph embedding methods, (4) analyze the root of the issue from three different perspectives (data, graph embedding method, assumption of outlier detector) and call community's attention to three important questions regarding (*i*) fair evaluation, (*ii*) model selection, and (*iii*) suitability of graph embedding method, and last but not the least (5) open source all methods and datasets used in our study (`https://github.com/LingxiaoShawn/GLOD-Issues`), to enable the community to use in their "GLOD" tasks and also to facilitate further investigation into the issues raised through our study. We summarize our main contributions as follows.

- **Study of Deep Graph-level Outlier Detection:** We start with the design and evaluation of two different categories of models for outlier detection in graph databases; namely, (1) two-stage models—pipelining unsupervised graph-level representation learning with off-the-shelf point-cloud outlier detectors, and (2) end-to-end models—learning representations simultaneously with optimizing an anomaly detection objective, such as one-class classification or reconstruction loss. (Sec. 7.1)

- **"Performance Flip" Issue with Using Classification Datasets for Evaluation:** To evaluate the aforementioned graph outlier models, we construct labeled benchmark datasets by repurposing binary graph classification datasets. Notably, we down-sample those datasets in *both* "directions", that is, we create *two* benchmark variants *per* classification dataset, respectively down-sampling one or the other class samples to constitute the outlier graphs. Surprisingly, we find that most models, while achieving high detection performance on one variant, fail considerably on the other. That is, we identify the intriguing issue of what we call "performance flip" depending on which class has been down-sampled. (Sec. 7.2.1)

- **Driving Factors behind "Performance Flip":** We find that the issue stems from the (mis)alignment between the inlier/outlier distributions created by graph embedding techniques and key underlying assumptions of the detection models. While one scenario creates a dense inlier distribution surrounded with dispersed outliers ('easy' task), the other creates a sparse inlier distribution that has overlapping support with a small set of outliers with relatively higher density ('hard' task). Since most models assume the former scenario in their formalism, they 'do well' on the respective 'easy' task.

  With an in-depth study over propagation based graph embedding methods, we identify two key leading factors behind the observed "performance flip" issue, particularly (1) *density disparity*; where the density of graph embeddings differ considerably between two classes, and (2) *overlapping support*; where the distributions of graph embeddings from the two classes exhibit overlapping support in the representation space. Moreover, we point out two contributing factors: (a) initial disparity between within-class sample similarities, and (b) amplification of this disparity by graph propagation – called *sparsification* – which is a property of some graph embedding models (Sec 7.2.2). We design quantitative metrics to concretely measure those factors (Sec. 7.2.3), and analyze the sparsification property via controlled simulations on k-regular graphs (Sec. 7.2.4). Finally, we present a detailed empirical study on real-world datasets (Sec. 7.3). We also present additional results on more datasets, using other detector and embedding methods in Sec. 7.3.4. Several additional observations are summarized which deepen our understanding of performance flip.

- **Insights for Graph-Level Outlier Detection and Beyond:** The performance flip issue is also observed for various other embedding methods and outlier detectors beyond propagation based methods, but which version of the downsample has higher ROC-AUC varies. The persistence of performance flip, but inconsistence of which version achieves higher ROC-AUC, raise several important problems to tackle GLOD: (1) as the performance flip is widely observed for all embedding methods, simply averaging performance among two versions of downsample seem problematic, as one version often has worse-than-random performance; (2) given the challenge of model selection for unsupervised methods, choosing which method to use is not only hard but also becomes risky as one may suffer from the worse-than-random performance; (3) as embedding methods play a large impact on outcomes, a better solution may involve designing an unsupervised graph embedding method that can generate clustered embeddings for different classes, which appears to be a hard problem for unsupervised tasks like outlier detection.

We also argue that issues we identify may extend beyond outlier detection, with possible implications on graph classification and clustering. Given the popularity of graph neural networks (GNNs), we point out that almost all GNN models employ a message-passing based propagation mechanism, as such, they also have the potential of suffering from sparsification. Specifically, this can cause severe overfitting for graph classification when the number of labeled samples is small. It can also adversely affect graph-level clustering tasks that aim to identify dense regions in the (representation) space. (Sec. 7.4)

7.1 GLAD Problem & Outlier Baselines

In this paper we focus on the graph-level outlier detection problem. The intriguing "performance flip" issue we observe arises from repurposing binary graph classification datasets for outlier detection evaluation. As far as we know, there is limited work studying the graph-level outlier detection problem, where the goal is to discover graphs with rare, unusual patterns which can be distinguished from the majority of graphs in a database. We call attention to the problem as it applies to many important real-world tasks from diverse domains such as drug discovery, money laundering, molecular synthesis, rare human pose detection [Mar+20], fake news detection [Mon+19], traffic events detection [Har+16], and buggy software detection [Liu+05].

7.1.1 Graph-Level Outlier Detection

Let $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ be an attributed or labeled graph with $\mathcal{V}$ and $\mathcal{E}$ depicting its vertex set and edge set, where each node $i \in \mathcal{V}$ is associated with a feature vector $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^T$ denotes the feature matrix, $n = |\mathcal{V}|$ being the total number of nodes. For labeled graphs, each node feature vector $\mathbf{x}_i \in \mathbb{R}^d$ is a one-hot encoded vector with d being the total number of unique (discrete) node labels.

Definition 7.1.1 (Graph-Level Outlier Detection Problem (GLOD))**.** Given a graph database $\mathcal{G} = \{G_1, \ldots, G_N\}$ containing N labeled or attributed graphs, find the graphs that differ significantly from the majority of graphs in $\mathcal{G}$.

The above problem is a general statement for graph-level outlier detection. In the real-world how one defines rareness or the degree of difference to the majority may be critical and may change depending on the application.

7.1.2 Graph-level Outlier Detection Models

Although there is no specifically designed method existing for GLOD, several methods for solving graph classification can be easily modified for tackling the problem. In this paper we mainly focus on three **propagation** based methods that can be categorized as two types: two-stage versus end-to-end. For the purposes of this paper, we find these three methods to be sufficiently illustrative of the issues we discover. **Notice that we focus on studying propagation based methods because all message-passing based GNNs belong to this category, which are the most promising and popular models for graph representation learning.** In additional experiments presented in Sec. 7.3.4, we also show results for two additional unsupervised graph embedding methods to demonstrate the persistence of the performance flip issue, namely Graph2Vec [Nar+17] and FGSD [VZ17]. Graph2Vec generates graph-level embeddings via Word2Vec by viewing motifs as "words" and the graph as a "document". FGSD embeds a graph as a histogram of all node spectral distances without using information of node labels. To avoid distraction from the carefully-studied propagation based methods, we omit their details and refer readers to the original papers ([Nar+17; VZ17]).

Two-Stage Graph Outlier Detection

Two-stage graph outlier detection approaches first transform graphs into graph embeddings or similarities between graphs by using unsupervised graph embedding methods (such as graph2vec [Nar+17] and FGSD [VZ17]) or graph kernels (such as feiler-Leman kernel [She+11] and propagation kernel [Neu+16]). Then traditional outlier detectors such as Isolation Forest [LTZ08], Local Outlier Factor (LOF [Bre+00]), and one-class SVM (OCSVM) [MY01] can be used to detect outliers in the embedding (vector) space. These approaches are easy to use and do not require much hyperparameter tuning, which makes them relatively stable for an unsupervised task like outlier detection. Nevertheless, two-stage methods may suffer from suboptimal solution as the feature extractor and outlier detector are independent. Moreover, most unsupervised graph feature extractors produce "hand-crafted" features that are deterministic without much room to improve, which further restrict the capacity of two-stage methods.

To illustrate the performance flip issue, we focus on graph kernel based two-stage approaches with two well-known outlier detectors used downstream: OCSVM and LOF. A graph kernel defines a kernel function $\mathcal{K}$ that outputs a similarity between two graphs. Formally it can be written as

$$\mathcal{K}(G, G') = \langle \phi(\mathcal{G}), \phi(\mathcal{G}') \rangle_{\mathcal{H}} \tag{7.1}$$

where $\mathcal{H}$ is a RKHS and $\langle \cdot, \cdot \rangle$ is the dot product in $\mathcal{H}$. The mapping $\phi(\mathcal{G})$ transforms graph $\mathcal{G}$ to an embedding vector in $\mathcal{H}$, which in our case contains counts of atomic subgraph patterns. Specifically we use the Weisfeiler-Leman subtree kernel and the propagation kernel, described as follows.

Weisfeiler-Leman Subtree Kernel. Inspired by Weisfeiler-Leman (WL) test of graph isomorphism [WL68] (a.k.a. the color-refinement algorithm), WL subtree kernel [She+11] processes a labeled graph by iteratively re-labeling each vertex with a new label compressed from a multiset label consisting of the vertex's original label and the sorted labels of its neighbors. This procedure repeats for L iterations for all graphs and outputs L re-labeled graphs $\{\mathcal{G}_1, ..., \mathcal{G}_L\}$ for every graph $\mathcal{G}$. One can easily show that each vertex in $\mathcal{G}_l$ at l iterations represents the subtree of the original vertex with depth l. WL subtree kernel compares two graphs by simply counting the number of co-occurrences of labels in both graphs at each iteration. The similarity score of two graphs is the summation of similarities across iterations. Formally, one can write it as

$$\mathcal{K}_{WL}(\mathcal{G}, \mathcal{G}') = \sum_{l=0}^{L} \langle \phi_{WL}(\mathcal{G}_l), \phi_{WL}(\mathcal{G}'_l) \rangle \tag{7.2}$$

where G_0 represents the original input graph, and ϕ_{WL} counts the frequency of all labels in the input graph by a vector with length equal to the number of unique labels.

Sparsification. Next we highlight a key property of the WL subtree kernel that is closely related to the performance flip issue we discover. Substructure-based graph kernels consider each substructure as a separate feature to compare among graphs. The total number of distinct substructures grows exponentially in the diameter of the substructures, which leads to the *sparsity problem* — that only a limited number of substructures would be shared among graphs. This property has also been referred to as *diagonal dominance* [YV15; Nar+16] — wherein each individual graph would mostly be similar only to itself but not much to any other graph. Being based on substructures, WL subtree kernel distinguishes each k-hop subgraph as a separate feature (re-labeled as a different label), as such, its feature space tends to grow exponentially in the number of iterations. The sparsification property of WL kernel is visualized in Fig. 7.1, where the diagonal dominance and diminishing similarity among graphs are observed clearly.

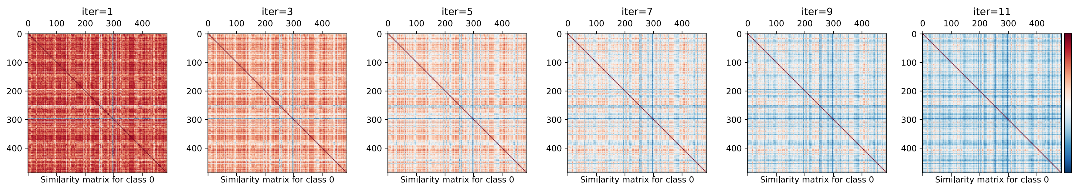


FIGURE 7.1: Sparsification in WL subtree kernel: Pairwise similarity of graphs (from DD dataset) decreases with increasing number of iterations (left to right).

Propagation Kernel. Propagation kernel (PK) [Neu+16] is inspired from the idea of propagating label information among nodes over the graph structure such as label propagation algorithm [ZG02] for semi-supervised node classification and can be used for both attributed graphs and one-hot encoded labeled graphs. For each graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, let $X_0 = \mathbf{X}$ denote the original feature matrix. Then PK generates a new feature matrix at each iteration by propagating the feature matrix using the transition matrix $T = D^{-1}A$ (where A is the adjacency matrix and D is the diagonal degree matrix) of the graph. Formally, $\mathbf{X}_{l+1} = T\mathbf{X}_l$. Similar to WL subtree kernel, PK compares two graphs at each iteration. The similarity between two graphs is measured based on propagated features through binning. Formally we can write the kernel as

$$\mathcal{K}_{PK}(\mathcal{G}, \mathcal{G}') = \sum_{l=0}^{L} \langle \phi_{PK}(\mathbf{X}_l^{\mathcal{G}}), \phi_{PK}(\mathbf{X}_l^{\mathcal{G}'}) \rangle \tag{7.3}$$

where $\phi_{PK}(\cdot)$ denotes the hash function that maps a given set of (feature) vectors into bins. To preserve locality and keep efficiency, locally sensitive hashing (LSH) [GIM99] is used for the binning.

Sparsification. The propagation kernel also exhibits the aforementioned sparsity problem, increasingly for larger number of iterations. Compared to WL subtree kernel that generates new features via re-labeling (a hard transformation), propagation kernel generates new features via multiplying by the transition matrix (a soft transformation). Thus, the feature space grows much slower for PK. As illustrated in Fig. 7.2, the diagonal dominance continues to hold but the sparsification occurs at a lower rate than WL subtree kernel (cf. Fig. 7.1).

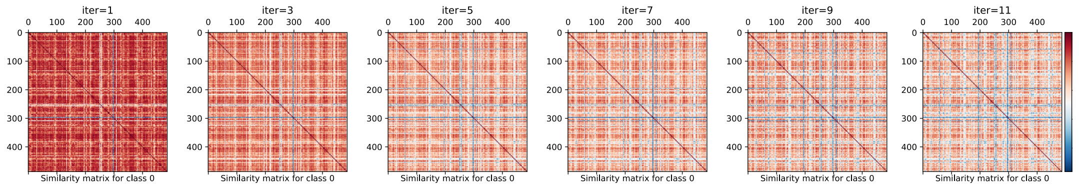


FIGURE 7.2: Sparsification in PK: Pairwise similarity of graphs (from DD dataset) decreases with increasing number of iterations (left to right).

End-to-End Deep Graph Outlier Detection

Deep learning methods have been used for outlier detection recently to enhance automatic feature learning for high-dimensional and structured data such as images. Recently graph neural network (GNN) has achieved great success in graph-structured data, and several works have successfully applied GNNs to node-level outlier detection on a single graph, such as OCGNN [Wan+20] and DOMINANT [Din+19]. However there is no deep model proposed for graph-level outlier detection.

Here we present a GNN model adapted from graph classification, and leverage a one-class classification objective function to address graph-level outlier detection. Compared with the widely used Graph Convolution Network (GCN) [KW17] model, Graph Isomorphism Network (GIN) [Xu+19] has been shown to be as powerful as the WL test of graph isomorphism, as such, we design a GIN based graph-level outlier detector. Note that an earlier GNN model called DGCNN [Zha+18] has discussed its connection to WL subtree kernel and propagation kernel. GIN builds on the ideas of DGCNN, and as a result also shares connection to these graph kernels. As we will present the issues we have discovered based on those graph kernels, we have also empirically verified that similar issues are observed for the GIN based model. In the following, we present our GIN based graph-level outlier detector, which is trained end-to-end through one-class classification loss.

Let $h_v^{(l)}$ be the l-th layer representation of node v in the GIN model. GIN updates node representations at each layer by

$$h_v^{(l)} = \mathrm{MLP}^{(l)}\Big((1+\epsilon^{(l)}) \cdot h_v^{(l-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(l-1)}\Big) \tag{7.4}$$

where MLP denotes a multi-layer perceptron (we use 2 layer) and $\mathcal{N}(v)$ denotes the direct neighbors of node v. Note that the summation operation around neighbor vectors can cause numerical explosion over iterations, thus batch normalization is applied between each GIN layer to prevent it.

After L layers, GIN generates the graph-level representation (i.e. graph embedding) using a readout function as follows.

$$h_{\mathcal{G}} = \mathrm{CONCAT}\Big(\ \mathrm{READOUT}(\{h_v^{(l)} | v \in \mathcal{G}\}) \mid l = 0, 1, \ldots, L\ \Big)\,. \tag{7.5}$$

While the original paper [Xu+19] proposed summation for the READOUT function to preserve maximum capacity, we use averaging (i.e. mean pooling) to account for different graph sizes in the database.

To build one-class classification into the GIN model, we borrow the idea from DeepSVDD [Ruf+18]. Specifically, we optimize the one-class deep SVDD objective at the output layer of the GIN model as

$$\min_{W} \quad \frac{1}{N} \sum_{i=1}^{N} \|\mathrm{GIN}(\mathcal{G}_i; W) - \mathbf{c}\|^2 + \frac{\lambda}{2} \sum_{l=1}^{L} \|W^l\|_F^2 \tag{7.6}$$

where W^l denotes the parameter of GIN at the l-th layer, $W = \{W^1, ..., W^l\}$, and $\mathbf{c}$ is the center of the hypersphere in the representation space that is obtained as the average of all graph representations upon initializing GIN model. Note that the second term corresponds to weight decay of deep models. As mentioned in [Ruf+18], deep one-class classification suffers from *feature collapse*, where the trained model maps all input instances to the (constant) $\mathbf{c}$. We employ the regularizations proposed therein to prevent this problem. After model training, the distance to center is used as the outlier score for each graph.

7.2 Using Classification Datasets for Outlier Model Evaluation: Issues

In this section we present in more detail the peculiar “performance flip” issue and related observations. Empirically we have observed that the issue is widely existing in many two-stage methods (see Table 7.4&7.5) and GNN models (we have evaluated a number of GNN model variants for GLOD in developing new algorithms, although we only present OCGIN.) on lots of datasets. Thus, having a clear understanding of this issue becomes critical for effectively and fairly evaluating detection models and consequently, being able to design better detectors and new models for GLOD.

In the following, we present the peculiar observations in detail (Sec. 7.2.1), state our hypothesis on the driving mechanisms behind these observations (Sec. 7.2.2), and introduce qualitative and quantitative measures for our empirical analysis (Sec. 7.2.3). In Sec. 7.3 we provide a measurement study using the measures proposed in Sec. 7.2.3 for propagation based methods to verify our hypothesis, as the underlying mechanism is consistent and easier to analyze. Also the close connection between GNN models and propagation based graph kernels [Zha+18] strengthens the importance of studying propagation methods. For the other two-stage methods, based on Graph2Vec and FGSD, we provide comprehensive performance evaluation and dicuss its implications. We omit the measurement study, however, as it is harder to analyze their underlying mechanisms.

7.2.1 Peculiar Observations

Graph classification is a widely studied problem with many public datasets available. As graph-level outlier detection is rarely studied with no available dataset, repurposing the graph classification datasets by downsampling one class as outlier can easily provide outlier detection tasks based on real-world samples. For binary classification dataset, there are two ways to down-sample (either down-sample the first class or the other class) to create two variants of outlier detection datasets. In this section, for the first time, we report several unexpected behaviors on various datasets created in this manner, with fixed downsampling rate 0.1. We split these binary classification datasets into two types: “X&Y” type and “X&Non-X” type. “X&Non-X” refers to datasets with one class representing a category (call it X) and the other class representing samples from any other categories other than X. “X&Y” type datasets have two classes specifically associated with two different real-world categories X and Y respectively.

Setup. We conducted GLOD task over 10 datasets (5 “X&Y” and 5 ‘X&Non-X”) using a total of 11 GLOD detectors. See Sec.7.3.1 for the detailed description, summary statistics of the datasets, and model configurations, and see Table 7.4 and Table 7.5 for comprehensive results. We summarize peculiar observations across datasets and GLOD detectors. We also give as examples to illustrate our observations on 4 datasets (DD, PROTEINS, NCI1, and IMDB, where IMDB is “X&Y” type and all others are “X&Non-X” type, with performance flip and related issues observed for the first 3 datasets but not for IMDB.), using 3 propagation based detectors (WL+LOF, PK+LOF, and OCGIN) due to space limitation.

Table 7.1: Average ROC-AUC performance (and standard deviation) of 3 different graph embedding based methods for graph outlier detection using 4 binary graph classification datasets. Each dataset has 2 down-sampled variants, where outliers are created by down-sampling one of two classes (class 0 or class 1) with rate= 0.1, averaged over 10 different down-samplings. **Performance flip observed** on DD, PROTEINS, and NCI1 for all 3 models, **where ROC-AUC is significantly larger on one variant than the other.** ROC-AUC values less than 0.5 are shown **in bold** as they indicate worse-than-random performance.

Dataset	Outlier Cls	OCGIN	WL+LOF	PK+LOF
DD	0	**0.327 (0.023)**	**0.186 (0.024)**	**0.194 (0.027)**
	1	0.720 (0.035)	0.815 (0.020)	0.824 (0.021)
PROTEINS	0	**0.370 (0.037)**	**0.276 (0.021)**	**0.389 (0.054)**
	1	0.681 (0.028)	0.664 (0.024)	0.557 (0.041)
NCI1	0	0.643 (0.030)	0.730 (0.012)	0.678 (0.019)
	1	**0.467 (0.028)**	**0.349 (0.022)**	**0.366 (0.027)**
IMDB	0	0.643 (0.039)	0.603 (0.038)	0.624 (0.030)
	1	0.508 (0.049)	0.651 (0.022)	0.581 (0.042)

Peculiar Observation 1: Performance Flip.

The main observation we make in this work is that **for any given outlier detecting models, the performances of detecting outlier appear to depend significantly on *which* class is down-sampled, resulting a large performance gap between two down-sampled variants.**. Table 7.1 shows the ROC-AUC on 4 datasets with their 2 variants at down-sampling rate 0.1 for 3 outlier detecting models. Results for other datasets and models are available at Table 7.4 and Table 7.5, with performance flip scenarios marked yellow.

Observation 1.1 (Performance Gap)**.** A large ROC-AUC gap is widely observed between the two different down-sampled variants of most datasets, consistently across all models.

Strikingly, not only the methods perform well on one variant and poorly on the other, but their performance is simply worse than random (!) in the latter case – since random ordering would achieve an ROC AUC of 0.5 in expectation. Perhaps more intriguingly:

Observation 1.2 (AUCs sum approximately to 1)**.** The sum of the two ROC-AUC values on the two variants of each dataset is approximately equal to 1, consistently across all models.

To understand this better, recall the probabilistic interpretation of the ROC-AUC: it is the probability of correctly ranking a random positive instance (i.e. outlier) above a random negative instance (i.e. inlier). Then these two observations together, which we refer to as the "performance flip" issue, suggest a *revised* statement: **the models always consider the graphs from one *fixed* class to be more outlier than those from the other, irrespective of which one is down-sampled.** In other words, it is not that the down-sampled class has impact on the performance, in contrast, the ranking by the models is agnostic to this so-called "ground-truth" but rather has a pre-determined bias toward one (fixed) class. Notice that this happens to be class 0 on DD and PROTEINS and class 1 on NCI1 for the three detectors.

Observation 1.3 (Performance flip is more severe for "X&Non-X" datasets)**.** For "X&Non-X" type datasets, the performance flip occurs more often and more severely (more often having larger performance gap) compared to "X&Y" type datasets.

In Table 7.1, the 3 datasets with performance flip are all "X&Non-X" datasets while IMDB is "X&Y" type. Furthermore, full results in Table 7.4 and Table 7.5 also support that performance flip occurs more often and performance gap is larger in distribution (e.g., at rate 67.3% vs. 30.6% for performance gap ≥ 0.2) on "X&Non-X" datasets than "X&Y" datasets.

Observation 1.4 (Correlation between performance and class semantics)**.** For propagation based methods, down-sampling "Non-X" class as outlier in"X&Non-X" dataset always achieves high performance. However it is not always true for other methods.

By definition "X" class refers to a category of instances that likely exhibit characteristic patterns of the class, while the "Non-X" class contains many patterns out of the "X" class. By aligning the performance result (Table 7.4) with class semantics shown in Table 7.2, we find clearly that *all* propagation based methods achieve high performance for down-sampling "Non-X". This supports our following analysis over propagation based methods, where we find that these methods have larger sparsification rate for more diverse classes.

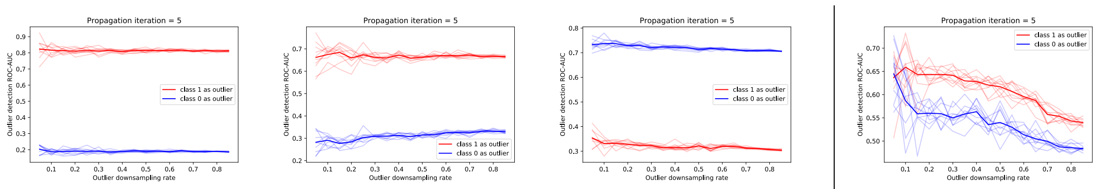
We conclude with four remarks. First, note that although the ranking of models by performance differs from dataset to dataset, the performance-flip and AUCs-sum-approximately-to-1 behaviors are consistent across models. Second, the issue does not appear to be universal as it does not arise on IMDB (cf. Table 7.1), where all models are better than random on both variants. Third, when performance flip is observed, which version of the downsample achieving high performance depends on both the embedding method as well as the downstream outlier detector, that is the performance gap can be reversed when using different type of graph embedding. Last, learning-based end-to-end model has the ability of capturing majority class distribution to a certain degree with performance flip not occurring for all "X&Y" type datasets, which points out a potentially promising direction to overcome performance flip.

Peculiar Observation 2: Invariance to Down-sampling Rate.

When down-sampling one class as outlier with a certain down-sampling rate, we would conjecture that a lower rate would make the outlier detection task easier as the density of outliers becomes lower. Fig. 7.3 shows the detection ROC-AUC of WL+LOF (with $L = 5$ iterations) for various down-sampling rates, from 0.05 to 0.85, on both variants of all datasets. The conjecture appears to hold only for IMDB – on which performance flip is not observed. In contrast, the performance is strikingly flat on DD, PROTEINS, and NCI1. Similar results hold for PK+LOF and OCGIN on these three datasets (See Sec. 7.3). More broadly, the observation holds for all methods and datasets when performance flip is observed (see additional result in our project webpage `https://github.com/LingxiaoShawn/GLOD-Issues`).

Observation 2 (Invariance to down-sampling rate)**.** Performance flip issue is not an artifact of the down-sampling rate, in fact, ROC-AUC appears to be invariant to the rate.

This observation is in agreement with Observation 1.2 (AUCs-sum-approximately-to-1). The probabilistic interpretation of ROC-AUC is regarding *any two* random positive-negative instances, irrespective of the total number of instances from those groups.



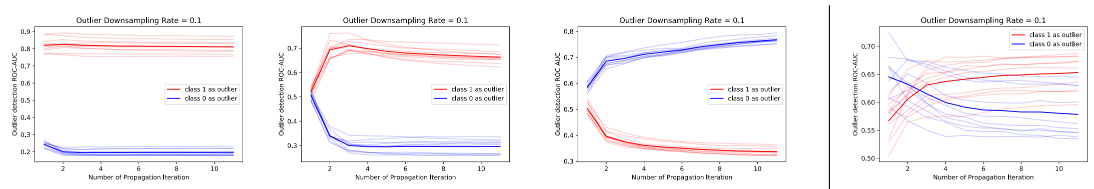
(A) DD (B) PROTEINS (C) NCI1 (D) IMDB

FIGURE 7.3: **Performance is invariant to downsampling rate** on DD, PROTEINS, and NCI1 for WL+LOF. Similar behavior is observed for methods and datasets when performance flip is occur.

Peculiar Observation 3: Growing Performance Gap with Propagation (propagation based methods only).

Observation 2 is mainly related to a property of the dataset generation. On the other hand, a key property of the outlier models we employ in this work is the number of iterations (for WL and PK) or the number of layers (for GIN), earlier denoted with L (See Sec. 7.1.2), both of which correspond to propagations over the graph. Here we look at how the performance behaves under varying L.

Fig. 7.4 shows the performance of WL+LOF on all datasets for two variants for L increased from 1 through 11. Results are qualitatively similar for PK+LOF, however OCGIN behaves differently (See Sec. 7.3).



(A) DD (B) PROTEINS (C) NCI1 (D) IMDB

FIGURE 7.4: **Performance gap between two variants tends to grow with increasing number of propagations** (i.e. iterations) of WL (subsequently paired with LOF), significantly on DD, PROTEINS, and NCI1. Similar behavior is observed for other propagation based methods and datasets.

Observation 3 (Growing gap with propagation)**.** The difference in ROC-AUC performances on two different down-sampled variants of a classification dataset tends to grow with increasing number of graph propagations for WL- and PK-based outlier models. For OCGIN there exists no obvious growth.

The growth is significant particularly on DD, PROTEINS, and NCI1 for which the performance flip occurs. It is interesting that this behavior is mainly associated with two-stage propagation models and not with our end-to-end model. We can reason about the two-stage models based on the sparsification property that both exhibit. (In contrast, OCGIN is optimized where it is harder to reason about how the learning of its many parameters affects performance.) Recall that sparsification, as discussed in Sec. 7.1.2, implies that the kernel distance between two graphs tends to increase with number of WL and PK iterations. Combined with Observation 1.4 we hypothesize that sparsification issue with increasing propagations is significant more noticeable for "Non-X" class (one with diverse patterns) and down-sampling this class as outlier results in an easier task where outliers are sparse and dispersed in the embedding space.

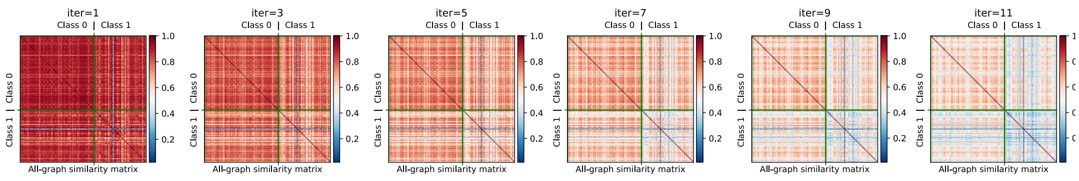


FIGURE 7.5: Pairwise similarities among all graphs in DD dataset (graphs grouped by class) based on WL subtree kernel over increasing iterations (left to right).

7.2.2 Hypothesis on Driving Mechanisms

Next we aim to build an understanding of the leading factors behind the unusual observations we have presented. As the underlying mechanism of propagation based methods is consistent and easier to analyze, we present our hypothesis and understanding based on propagation based methods. However the main argument of two factors (density disparity and overlapping support of graph embedding space) holds true for other non-propagation based methods as well. To that end, we focus on investigating pairwise similarities – either produced by the graph kernel or the dot product between graph embeddings – normalized in range $[0, 1]$. We consider all-pairs similarities among all graphs before down-sampling. These convey important information about the density of and the distance between the graphs across classes in the graph embedding space, which many downstream outlier detectors rely on, including the two – LOF and OCSVM – we have used.

Fig. 7.5 shows the pairwise similarities among all graphs as well as how those change with increasing number of graph propagations (i.e. iterations) based on the WL subtree kernel on DD dataset where 0 represents enzyme and 1 represents non-enzyme. (Similar plots for PK and OCGIN can (will) be found in `https://github.com/LingxiaoShawn/GLOD-Issues`[1]) The pairwise similarities are block-wise grouped based on the true class label. We emphasize two factors:

1. *Diagonal, block-wise (intra-class) similarities:* Clearly sparsification arises for *both* classes, where the graphs within the same class become more and more dissimilar to one another as the number of iterations increases. What is even more important to note is that *the speed at which sparsification occurs is different for the two classes.* As shown in Fig. 7.5 for DD, distribution of graphs in class 1 sparsifies much faster than that for class 0, leading to a *disparity between class-level densities.* Although less apparent from the first subfigure, density disparity exists even after a single iteration. The initial disparity difference aligns with the class semantic meaning where class 0 represents enzyme and is more compact (denser) than class 1. Increasing number of iterations only amplifies this disparity.

2. *Off-diagonal (inter-class) similarities:* Pairwise similarities among graphs within the sparser class (in Fig. 7.5, class 1 for DD) are *lower* than inter-class similarities on average. Put differently, graphs within the sparser class tend to look more similar to graphs in the other class than among themselves. This suggests that the class-level distributions of graphs in the embedding space have *overlapping support.*

[1]Our study involves an in-depth study of three different graph outlier detection models. Due to limited space for figures, we include additional or corresponding figures in the Appendix section of our arXiv submission [ZA23].

Based on the above, we conjecture the following conditions as the leading factors behind our peculiar "performance flip" and related observations:

- **(Growing) Density Disparity.** Density of outliers in the feature space is often inversely correlated with the difficulty of identifying them. That is, the more spread out and apart from inliers are the outliers, the easier the detection task gets. This suggests that designating as outliers the graphs from the class with sparser graph embeddings (or the class that sparsifies faster) would induce a relatively easier task, as compared to graphs from the other class with higher embedding density. What is more, graph propagation would contribute to a growing disparity between task difficulties, as density disparity grows with more propagation. For propagation based graph embedding methods the class density over embedding space aligns with the diversity of patterns in the original graph space, but this does not always hold for other graph embedding methods.

- **Overlapping Support.** Density disparity alone is not a sufficient condition to explain the performance disparity (or flip) issue. Two sets of graph embeddings that are fully separable in the feature space would both induce an 'easy' task, no matter how different their within-set (i.e. class) densities are. Unsupervised graph representation learning methods, however, do not necessarily have the ability (at least explicitly) to embed graphs from the same class closeby while maintaining inter-class separability. On the contrary, they tend to generate mixed embeddings that have a common/overlapping support among classes.

We conclude by forming the following hypothesis on the driving mechanisms behind "performance flip" and related observations. Provided graph embedding methods employed for outlier detection induce both density disparity as well as overlapping support, one down-sampling scenario creates a dense inlier distribution surrounded with dispersed outliers ('easy' task), whereas the other creates a sparse inlier distribution that has overlapping support with a small set of outliers with relatively higher density ('hard' task). Since most models assume the former scenario in their formalism, they 'do well' on the respective 'easy' task, and poorly on the other (Observation 1.1). More broadly, therefore, the performance flip issue stems from the (mis)alignment of embedding (i.e. inlier/outlier) distributions with the underlying assumptions of the detection models. As the embedding space is produced by graph embedding methods, which class has denser embeddings relies on the specific graph embedding method and may not align with semantic meaning of class labels. However when the original dataset has one class with clearly more diverse patterns (like "Non-X" class in "X&Non-X" datasets), the propagation based methods in general produce sparser embedding space for the semantically-more-diverse class (Observation 1.4 and Observation 1.3). When performance flip happens, perhaps more disturbingly, the models tend to always consider the graphs from the sparser class to be more outlying than the graphs from the other, denser class – no matter *which* one is down-sampled and at what rate (Observation 2). That is, their probability of ranking a sparser-class instance above a denser-class instance remains almost the same, which leads to the observed AUCs-sum-approximately-to-1 behavior (Observation 1.2) and consequently, the worse-than-random performance when the denser class is down-sampled. The issue is exacerbated with more graph propagation as it leads to a growing disparity of densities and respective task difficulties (Observation 3).

7.2.3 Measures for Analysis

In the previous section we pointed out overlapping support and (growing) density disparity to be two key factors leading to the observed unusual behaviors. Here we introduce concrete measures to quantify these two factors.

1. **Qualitative visualization of overlapping support and growing density disparity:** Pairwise similarities do not directly reflect how points (i.e. graphs) are distributed in the representation space. We use multi-dimensional scaling (MDS) to map the graph embeddings into 2-dimensions (MDS mapping aims to preserve the relative pairwise similarities) wherein points from different classes are colored differently. Overlapping support can be validated visually from the mixing of two colors. On the other hand, the growing density disparity can be supported by the varying spread of points across classes and the rate at which this spread changes for each class across MDS visualizations corresponding to increasing propagation by a detector.

We remark that the MDS visualizations provide only qualitative evidence, as the mixing of colors and the varying spread of points could also be attributed to MDS error, i.e. could simply be artifacts; due to the space in which the pairwise similarities are to be preserved is constrained to only 2 dimensions for visualization purposes. Therefore, we also analyze quantitative measures of these factors described as follows.

2. **Quantitative measure of density disparity:** The distance between any two graphs from the sparse class would be larger. Therefore, we use the so-called *NN-Radius* to quantify the degree of density.

   Definition 7.2.1. *NN-Radius is the distance (or* $1-$*similarity as normalized in range* $[0,1]$*) to the k-th*[2] *nearest neighbor (NN) of a graph in the embedding space.*

   A larger radius corresponds to lower local density. The distribution of the NN-radii of the graphs from each class measures the density of the class. A more left-shifted distribution would imply a denser class.

3. **Quantitative measure of overlapping support:** In the absence of overlap, assuming graphs from different classes are linearly well-separated in the embedding space (forming two disjoint clusters), we would expect the nearest neighbors of each graph to be from the same class. Therefore, we use the so-called *NN-Disagreement%* as the degree of overlap.

   Definition 7.2.2. *NN-Disagreement% is the percentage of graphs from the opposite class within the NN-Radius of a graph.*

   We then study the distribution of NN-impurities of the graphs from each class. A left- or right-shifted distribution would respectively show whether a class is more likely to be surrounded by its own members (i.e. well-clustered, dense) or not (i.e. dispersed, sparse).

[2] In the paper we report results for $k = 20$ and note that the take-aways are not sensitive to this choice.

7.2.4 A deeper analysis on embedding sparsity issue

In this section we perform various simulations to better understand the disparity in sparsification rates between two classes, that is how one sparsifies faster than the other (See e.g. Fig. 7.5). We use WL subtree kernel for the simulations as these do not require any parameter training and hence are easier to analyze. The simulations are designed to show: (1) Even a small difference between two graphs can be amplified by propagation, leading to sparsification; and (2) Various factors, including differences in label distribution and topology, contributes to differences in the rate of sparsification between two classes.

We consider simulations on k-regular graphs where each node has exactly k number of neighbors, i.e. the same degree. (Unlabeled) k-regular graphs are among the class of graphs on which 1-dimensional WL (1-WL) isomorphism test [WL68] fails to reject non-isomorphism. That is, 1-WL test cannot distinguish two structurally non-isomorphic k-regular graphs. In practice, node/edge labels or attributes may help diminish such difficulty by breaking the symmetry [Li+20]. This is exactly the route we take to induce asymmetry in a controlled fashion (from low to high). Specifically, we design two cases:

- **Case 1:** We start with two identical (k, n) regular graphs (n nodes, all with degree=k), where all nodes are labeled A. Then, we flip the labels of m randomly chosen nodes in each graph to B. As such, the graphs are structurally isomorphic, whereas B-labeled nodes are ensured to induce assymetry (and hence non-isomorphism) between the graphs.
- **Case 2:** We start with two identical (k, n) regular graphs, where the nodes are labeled with two different labels, A or B. To induce non-isomorphism, we perform degree-preserving rewiring between r randomly chosen edge pairs in one of the graphs.

Our goal is to start from a place where WL distance is zero (i.e. identical graphs) and to study the effect of increasing m or r on the distance growth. Fig. 7.6(a) illustrates Case 1 on two so-called Petersen graphs ($k = 3$, $n = 10$) where $m = 1$ node's label in each graph is flipped from gray (label A) to red (label B), inducing non-isomorphism. Case 2 is illustrated in (b) where we rewire $r = 1$ edge pair in one graph (right) to create a graph that is non-isomorphic to the other (left) graph.

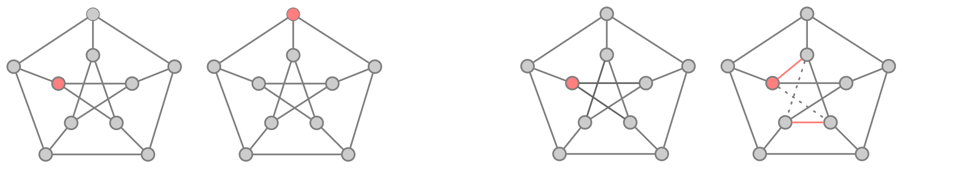

(a) non-isomorphism by node labeling (b) non-isomorphism by edge rewiring

FIGURE 7.6: Non-isomorphism is induced (a) between two unlabeled Petersen graphs by flipping node labels assymmetrically, and (b) between two labeled Petersen graphs by degree-preserving edge rewiring.

Without the perturbations (label flipping in Case 1, and rewiring in Case 2), WL distance between the two graphs would be equal to zero in both cases, irrespective of number of iterations. Just a small change, as shown in Fig. 7.6, "jump-starts" WL where the distance between the graphs starts growing with increasing WL iterations.

In Fig. 7.7 (a) and (b) we show how graph distance changes with increasing WL iterations as a function of number of node labels flipped in Case 1 and number of edge pairs rewired in Case 2, respectively. Notice that even after a relatively small perturbation on otherwise non-distinguishable graphs, their distance grows over iterations. The bigger the amount of perturbation (i.e. the difference between two graphs), the larger the distances as reflected by WL, and also the larger the growth rate (note the increasing gap between curves).

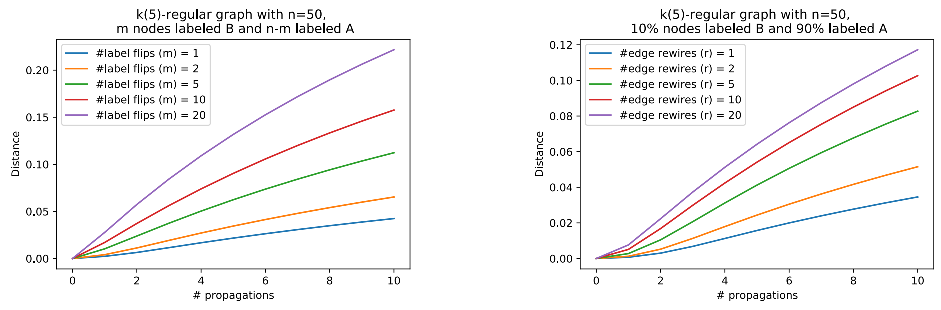


FIGURE 7.7: Distance between two k-regular graphs ($k = 5$, $n = 50$) as a function of (left, Case 1) number of node labels flipped, and (right, Case 2) number of edge pairs rewired. Each curve is averaged over 100 rounds to remove randomness.

The distance growth is also a consequence of graph topology. Fig. 7.8(a) shows graph distance over WL iterations on k-regular graphs with varying k, when number of label flips is fixed to $m = 5$ in Case 1. Similarly, Fig. 7.8(b) shows the same when the number of edge rewirings is fixed to $r = 10$ in Case 2. In both cases the graph distances are larger for increasing k across iterations. These show that the effect on distance of the same (even a small) amount of perturbation on two graphs varies depending on the topology.

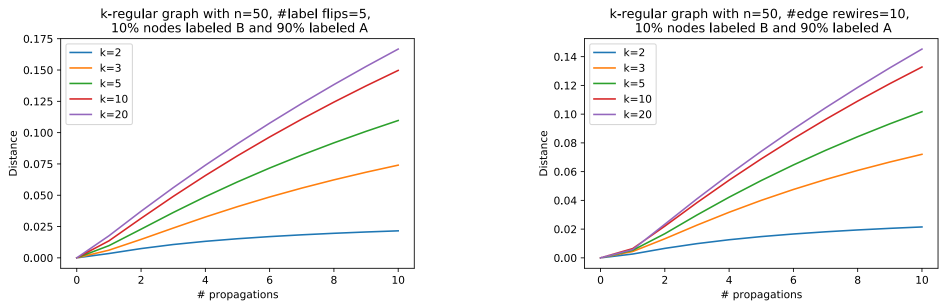


FIGURE 7.8: Distance between two k-regular graphs ($n = 50$) as a function of k for (left, Case 1) when $m = 5$ node labels are flipped, and for (right, Case 2) when $r = 10$ edge pairs are rewired. Each curve is averaged over 100 rounds to remove randomness.

All in all, sparsification is evident from our simulations – i.e. distances grow over iterations, or graph propagations. The growth rate of sparsification is sensitive to various factors (difference in label distribution, difference in topology, etc.). It is unlikely that real-world classes would consist of graphs with exactly the same variation for all factors. Therefore, it is not a freak occurrence that on average the distance between two graphs in one class sparsifies at a different rate than between those in another class. In fact, it is the opposite — it would be quite a coincidence for graphs from two different classes to sparsify at exactly the same rate. As a result, class-level density disparity as discussed in Sec. 7.2.2 appears to be an inevitable consequence of using these types of representation learning techniques.

7.3 Empirical Analysis

To reiterate, we hypothesize that the performance flip issue occurs when samples from different classes have large enough density disparity and overlapping support. Provided those two conditions hold, outlier models tend to rank the class with sparser graph embedding above the denser class no matter which class is down-sampled at what rate. In this section, we present further measurement and quantitative analysis based on measures in Sec. 7.2.3 to support our hypothesis. Notice that we mainly provide measurement study for propagation based methods, due to their importance and easy-to-study mechanism. Sec. 7.3.1 introduce the setup in detail regarding datasets and model configurations. Sec. 7.3.2 and Sec. 7.3.3 contain measurement study of propagation based methods for one dataset with performance flip (DD) and the other without (IMDB), results for other datasets will be available at `https://github.com/LingxiaoShawn/GLOD-Issues`. Sec. 7.3.4 provides a comprehensive performance study for 10 datasets and 11 detection models.

7.3.1 Experiment Setup

TABLE 7.2: Dataset class semantics.

Dataset	**Domain**	**Class Labels**	**Class Semantic**
DD [DD03]	Protein structure	0 1	Enzymes Non-enzymes
PROTEINS [Bor+05]	Protein structure	0 1	Enzymes Non-enzymes
NCI1 [WWK08]	Molecular (drug)	0 1	Inactive for anti-HIV Active for anti-HIV
IMDB-BINARY [YV15]	Actors' collaboration network	0 1	Movie genre: Action Movie genre: Romance
Mutagenicity [KMB05]	Molecular	0 1	Mutagens Not mutagens
AIDS [RB08]	Molecular	0 1	Active against HIV Inactive against HIV
ENZYMES [Sch+04]	Protein structure	0∼5	Different type of enzymes
REDDIT-5K [YV15]	Discussion thread	0∼4	5 type of subreddits: worldnews, videos, AdviceAnimals, aww and mildlyinteresting

Datasets. We use 10 real-world binary labeled graph classification datasets where 4 of them are derived from 2 multi-class classification datasets (ENZYMES and REDDIT-5K) by picking 2 classes out. These 10 datasets come from three domains: molecular chemistry, social networks, and bioinformatics. IMDB and REDDIT-5K are from social network domain, that contain unlabeled graphs, for which we create label by using their node degrees. We report their dataset statistics in Table 7.3. We also provide detailed class semantics for every class in each dataset in Table 7.2. Based on their class semantics we divide 10 datasets into 2 groups as mentioned in

TABLE 7.3: Dataset summary statistics.

Dataset	Class	#Graphs	#Node Labels	Avg. #Nodes	Avg. #Edges	Avg. Degree
DD	0	691	89	355.2	1806.6	5.04
	1	487	89	183.7	898.8	4.88
PROTEINS	0	663	3	50	188.1	3.79
	1	450	3	22.9	83.1	3.64
NCI1	0	2053	37	25.65	55.3	2.15
	1	2057	37	34.07	73.9	2.17
IMDB	0	500	-	20.1	193.5	9.1
	1	500	-	19.4	192.5	8.6
Mutagenicity	0	2401	14	29.3	60.5	2.1
	1	1936	14	31.4	62.7	2.0
AIDS	0	400	38	37.6	80.5	2.13
	1	1600	38	10.2	20.3	1.98
ENZYMES	0	100	3	36.2	132.7	3.84
	1	100	3	29.9	113.7	3.79
	2	100	3	28.9	111.2	3.86
	3	100	3	38.2	148.8	3.99
REDDIT-5K	0	1000	-	799.4	2035.5	2.52
	1	1000	-	852.1	1940.4	2.23
	2	1000	-	374.1	856.5	2.24
	3	1000	-	249.6	534.0	2.11
	4	1000	-	267.0	581.7	2.14

Sec.7.3.1: "X&Y" type (datasets in Table 7.5) and "X&Non-X" type (datasets in Table 7.4). "X&Non-X" datasets contain one compact class "X" and the other broader class containing samples not in "X", while "X&Y" datasets contain two compact/regular classes. We remark that even in "X&Non-X" type dataset one may hardly decide on the natural, semantic outlier class – e.g. in AIDS one can argue that being active against HIV should be outlier as it is rare and semantically important, yet one can also argue that in the context of developing new drugs for treating AIDS, detecting ineffective drug should be the outlier detection task. This suggests that in some cases repurposing graph classification datasets by down-sampling either class can be considered meaningful.

Model configuration of propagation based methods: We provide both performance study and measurement study for propagation based methods: two-stage models WL+LOF/OCSVM and PK+LOF/OCSVM [3], as well as end-to-end deep-one-class model OCGIN. We study the behavior of these models under different number of propagations; specifically WL and PK iterations range from 1 to 11, and OCGIN embeddings are extracted from layers 0 (i.e. input node vectors) through 5. For WL we specifically use WL subtree kernel. PK has a bin-width hyperparameter for hash function ϕ_{PK} (See Eq. 7.3), which is set to 0.1. Note that smaller bin-width leads to faster sparsification with a more severe performance flip. We use two different types of downstream outlier detectors: LOF (density-based) and OCSVM (one-class based). The LOF outlier detector is setup with default parameters (k =20 number of neighbors, and leaf size 30) from scikit-learn [Ped+11]. For OCSVM we use kernel-based SVM with the kernel output from graph kernels, and setup contamination factor $nu = 0.1$.

[3]PK and WL implementation use [Sig+20].

For OCGIN we use the default GIN implementation from [Xu+19], where we remove bias terms at all layers to prevent feature collapse [Ruf+18]. A graph's representation is produced from the summation of all previous layers' hidden representations, with a mean pooling over all nodes in the graph. Note that we train OCGIN only on down-sampled variants of a dataset, as it is trained end-to-end assuming outliers to be minority. For figures utilizing full data, we simply feed-forward all the graphs in the database over the *trained* model. We set number of layers to $L = 5$ and number of hidden units to 128 for all datasets. We use the Adam optimizer [KB14] to train OCGIN with a $5 \cdot 10^{-4}$ L_2 penalty on weights. The model is trained for 25 epochs. All other hyperparameters are picked from typical/default values, since our goal is to illustrate the performance flip and related issues instead of achieving best performance. Hyperparameter selection for unsupervised deep outlier detection is an important problem which is outside the scope of this paper.

Model configuration of non-propagation based methods: We only provide measurement study for propagation based methods and omit the same for others as the underlying mechanism is different and harder to analyze. We use two graph embedding methods (Graph2Vec and FGSD [4]) and 3 downstream outlier detectors (LOF, OCSVM, and Isolation Forest [LTZ08]). Graph2Vec views graph as "document" and subgraph as "word" and learn the embedding using the Word2Vec [Mik+13] algorithm. We use the default parameter implemented in KarateClub [RKS20] with number of WL iterations set to 3. FGSD uses histogram of spectral distances among all node pairs to create graph embeddings and it only uses graph embedding without considering node labels. Similarly we use the default implementation given in KarateClub. As for three downstream outlier detectors, LOF and OCSVM share the same configuration with propagation based methods. Isolation Forest is not used for PK and WL as it does not support kernel matrix as input. We use the default setup in sklearn for it.

Pairwise similarity matrix. Our proposed measures in Sec. 7.2.3 are computed on top of pairwise similarities among all graphs. For PK and WL kernels, the normalized kernel matrix is investigated. For OCGIN, where we only have access to graph embeddings, we calculate pairwise similarity as (1 - normalized pairwise distance) between two graphs, using Euclidean distance (i.e. L_2 norm). Similarity matrix is normalized to range $[0, 1]$ via dividing it by the largest element in the matrix.

Remark. We (will) include all figures corresponding to those presented in the following as well as previous sections for all propagation based models on all datasets in project website (`https://github.com/LingxiaoShawn/GLOD-Issues`). Only a subset of them are presented in this manuscript for brevity.

7.3.2 Measurement study: when performance flip occurs

We have observed performance flip on majority of datasets. In this section we present our measurement study and analysis based on DD dataset.

Analysis on full data.

We start with analyzing the inherent differences between two classes regarding density on DD. Fig. 7.9 (top row) visualizes the all-pairs similarity matrix based on WL over increasing iterations where sparsification can be observed for both classes (left to right). In the second row, 2-d MDS embeddings of all the graphs based on the corresponding pairwise similarities are shown. Again sparsification can be visually

[4]We use implementation in [RKS20].

confirmed based on the increasing spread of points (i.e. graphs) in each class. In addition, we notice that class 0 (green points) *sparsify faster* than class 1 (orange points), as the denser class 1 instances are surrounded by the dispersed class 0 instances. We quantify this difference via the *NN-Radius* measure, as in the third row, where the corresponding distributions of *NN-Radius* for all graphs are shown for each class. Over iterations both class distributions shift to the right (i.e. sparsify). The shift is more evident for class 0 (i.e. speed of sparsification is larger) as the (green) histogram spreads out while the other (orange) histogram remains relatively peaked.

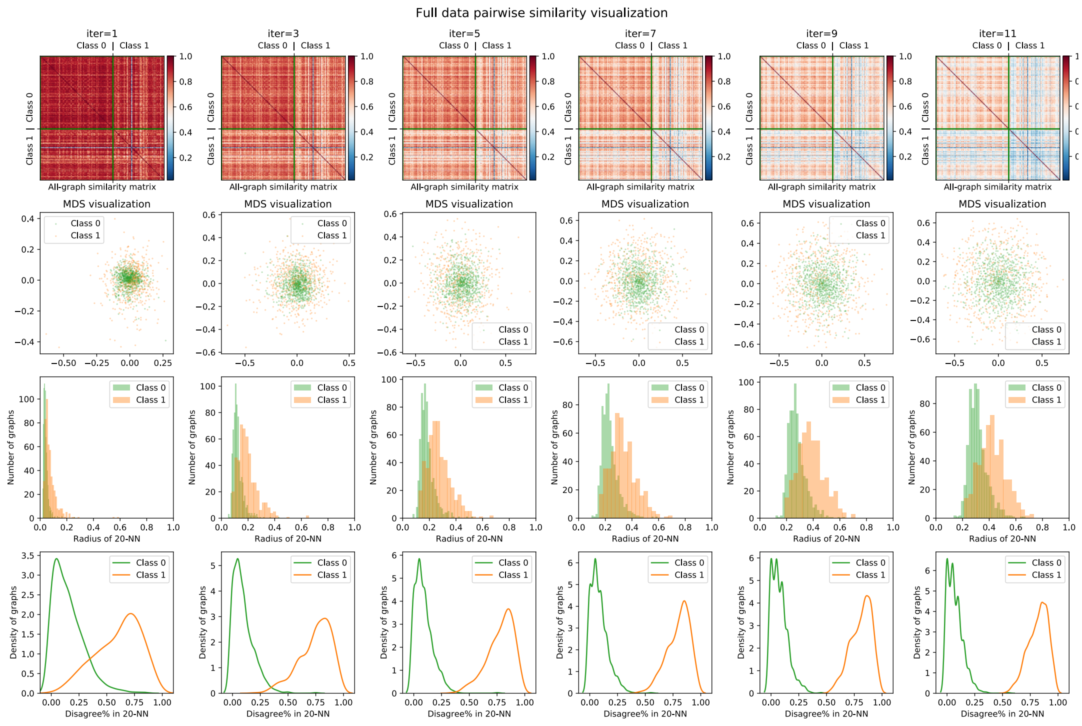


FIGURE 7.9: (Top row) Pairwise similarity matrix for all graphs in full DD dataset, based on WL subtree kernel over increasing iterations (left to right). (Second row) 2-d MDS visualization based on the similarity matrix. (Third row) Distribution of *NN-Radius* for all graphs in each class. (Last row) Distribution of *NN-Disagreement%* for all graphs in each class.

Next we analyze the overlapping support between two classes. The mixing of colors in the MDS visualizations is suggestive of overlap, however one can argue that it is an artifact of limited representation capacity of MDS in 2-d. To quantify overlap more concretely, we show the distributions of *NN-Disagreement%* for all graphs in each class in the last row of Fig. 7.9. The distributions for class 1 concentrate more on the left side (majority of neighbors are from the *same* class) whereas for class 0 they concentrate on the right (majority of neighbors are from the *opposite* class). Both density disparity and overlapping support are increasingly more evident with increasing number of iterations, which suggests that graph propagation amplifies the issue.

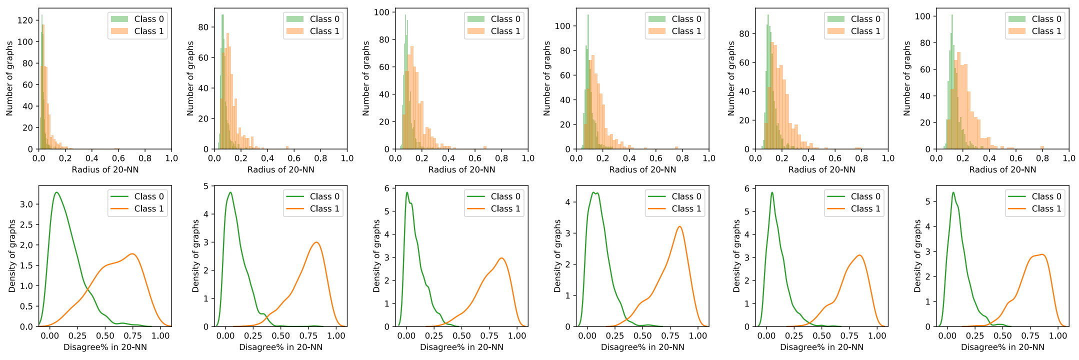
To summarize, this analysis confirms that class 0 in DD is the increasingly sparser class with larger *NN-Radius* (means sample density is smaller) and higher *NN-Disagreement%* (means higher overlapping support). As a result, down-sampling class 0 as outlier induces a relatively 'easy' task for WL+LOF. Moreover, the task becomes 'easier' with more propagation as the graphs in class 0 spread out even further, hence the increasing performance gap.

The conclusions are similar for PK on DD, as shown in Fig. 7.10(a). For OCGIN, while we continue to observe performance flip on DD variants, it is to a lesser degree. Specifically we find that the disparity between classes is not worsened with increasing graph propagation in this case, as suggested by Fig. 7.10(b). In fact, the difference in distributions seems to close especially at the last layer. Irrespectively, OCGIN performs considerably better when class 0 is down-sampled like the other models, meaning that it is not shielded from the performance flip issue that we identify. We believe this is due to the initial disparity (See Fig. 7.10(b) at layer 0, i.e. original input), which it cannot recover from, despite model training. This is akin to a bad initialization (for one variant) of OCGIN that ultimately leads to poor performance.

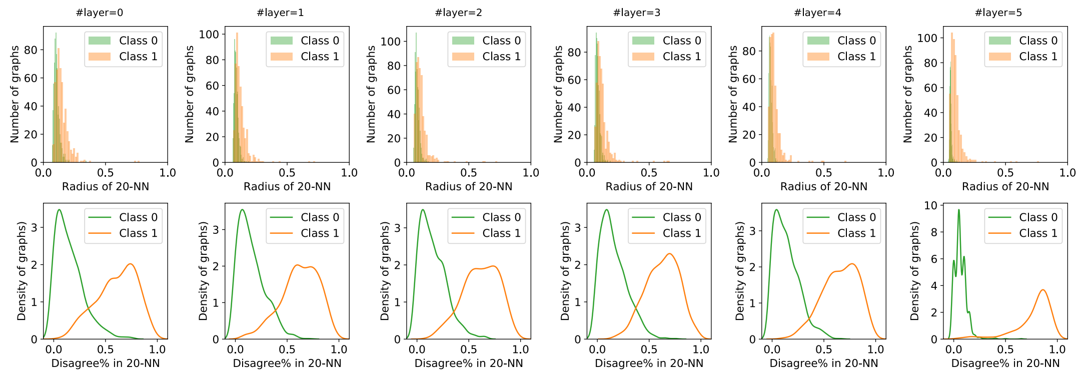
Analysis upon down-sampling.

Next we analyze the flip in performance upon down-sampling one class or the other via the contrast in *NN-Disagreement%* distributions. Fig. 7.11(top) and (bottom) respectively show those distributions when class 1 is down-sampled (denoted Outlier in red) and when class 0 is down-sampled (now denoted Outlier in red) for WL on DD. We notice the stark difference: At the (bottom), the inliers form a dense distribution with negligible mixing with the outliers. The outliers are dispersed, far from one another, as their NN-neighborhood mainly contains inliers. This is the 'easy' detection task that well aligns with the underlying assumption of most outlier detectors – hence the high peformance of WL+LOF.

In contrast, the (top) figures suggest that the inliers and outliers are intermixed, which worsens with propagation as the distributions become more and more indistinguishable. (Note that down-sampled class inherently has a right-shifted distribution



(a) PK



(b) OCGIN.

FIGURE 7.10: Quantitative measures of density disparity and overlapping support on DD for (a) PK and (b) OCGIN for increasing graph propagation (left to right).

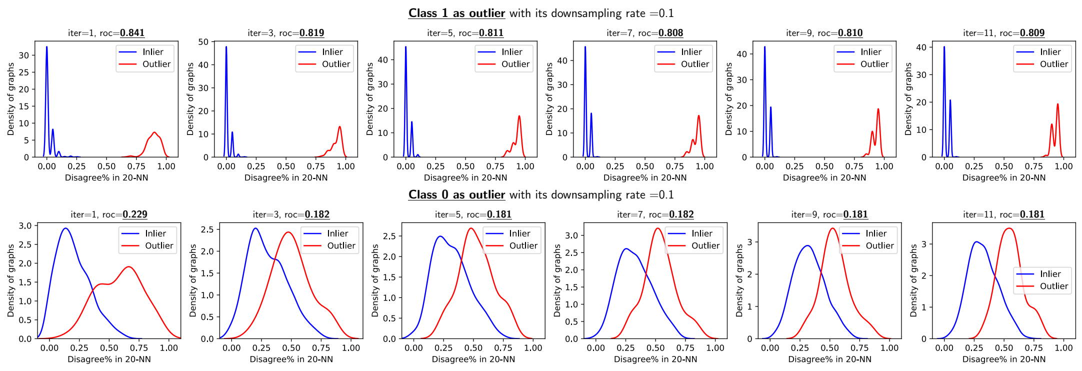


FIGURE 7.11: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over WL iterations (left to right) on DD.

as down-sampling induces sparsity.) This corresponds to the 'hard' detection task where it is difficult to distinguish inliers from outliers – hence the poor, in fact worse-than-random performance.

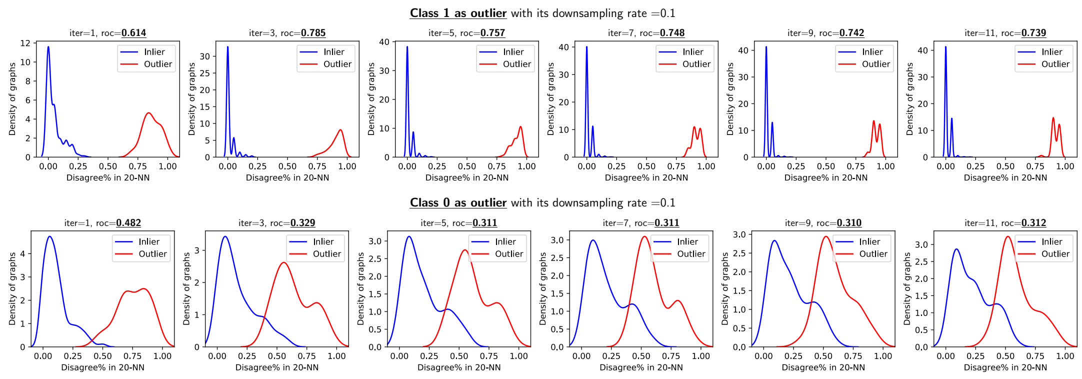
The conclusions are similar for WL+LOF on PROTEINS and NCI1, as shown in Fig. 7.12(a) and (b), respectively (except for NCI1 down-sampling class 0 happens to induce the 'hard' task). In fact, performance appears to be inversely correlated with the amount of overlap between the *NN-Disagreement%* distributions of inliers and outliers. We refer to the Appendix in [ZA23] for similar results on these three datasets for PK+LOF and OCGIN.

7.3.3 Measurement study: when performance flip does not occur

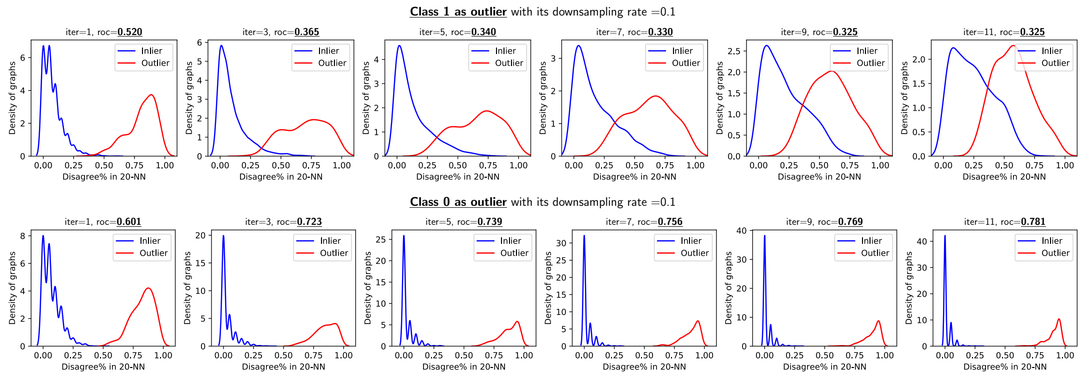
Although the performance flip issue occured on a considerable number of datasets we have experimented with, it is not always observed. In this section, we present a similar analysis for IMDB-BINARY for comparison purposes. Results for other datasets and propagation based methods will be available at `https://github.com/LingxiaoShawn/GLOD-Issues`.

As shown in Fig. 7.13, sparsification arises rather fast on IMDB-BINARY with increasing WL iterations – notice the drastic shift of the *NN-Radius* distributions from left-most to right-most (third row). This can also be visually confirmed from the pairwise similarity matrix (top row) as the heatmap gets darker blue (from left to right). Interestingly, the rate (or speed) of sparsification appears to be similar among the two classes. As such, density disparity does not seem to arise – notice the similar mixing among the colored points in 2-d MDS visualization (second row). On the other hand, we continue to observe the overlapping support (i.e. mixing) of graph embeddings to a large extent (last row).

These suggest that down-sampling any one of the classes as outlier would induce two detection tasks with similar difficulty. Fig. 7.14 confirms this hypothesis, where the distribution of inliers and outliers in the embedding space look similar between the two down-sampled variants of IMDB-BINARY. Moreover, the outliers are sparser and more dispersed as compared to the inliers, which consequently leads to better-than-random performance on both tasks. The ROC-AUC values are not too high due to the mixing of the embeddings that makes the detection task harder. In Sec. 7.2.2 we argued that density disparity alone is not a sufficient condition for performance flip. This result on IMDB-BINARY shows that overlapping support (i.e. mixing) alone is also not a sufficient condition for the performance flip. Both conditions together give rise to the issue.



(a) PROTEINS



(b) NCI1

FIGURE 7.12: *NN-Disagreement%* distribution of outliers (top rows: when class 1 is down-sampled and bottom rows: when class 0 is down-sampled, in red) and inliers (vice versa, in blue) over WL iterations (left to right) on (a) PROTEINS and (b) NCI1.

To conclude, our analysis sheds light onto the leading factors (density disparity and overlapping support) as well as contributing factors (graph propagation) behind the performance flip issue. These help us reason about both cases in which it occurs or not. However, we do not have a full understanding of why these factors arise for some datasets but not the others in the first place. Future research could focus on studying this disparity between datasets.

7.3.4 Performance study: All GLOD methods

In the previous subsections, we have demonstrated an understanding and analysis over propagation based GLOD methods. It is natural to ask whether only propagation based methods suffer from performance flip. Some observations are already summarized in Sec.7.2. FGSD operates in the Laplacian eigen-space, and Graph2Vec employs skip-gram based training via negative sampling, etc. As such, these other methods are harder to understand and analyze in detail. In this section we present a full performance study over 10 datasets and these additional methods to empirically answer the following questions:

- Q1: Is performance flip only restricted to propagation based methods?
- Q2: Do semantics of the dataset have any influence on performance flip?
- Q3: For two-stage methods (with first embedding and then outlier detection stages), how does each stage affect performance flip?

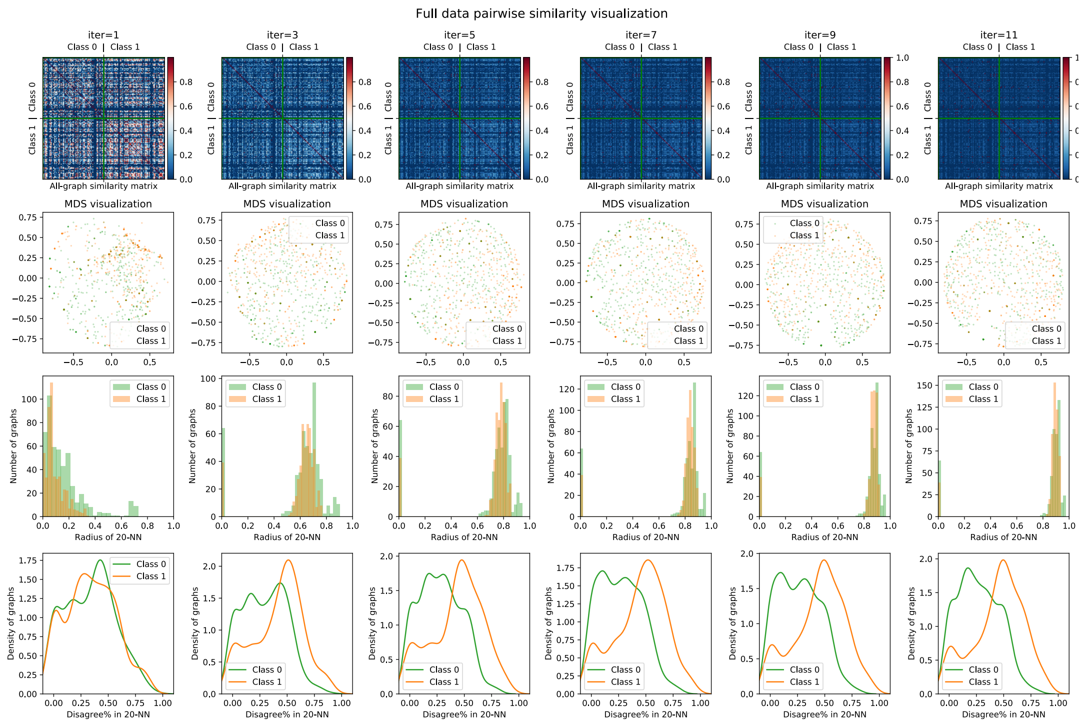


FIGURE 7.13: (Top row) Pairwise similarity matrix for all graphs in full IMDB-BINARY dataset, based on WL subtree kernel over increasing iterations (left to right). (Second row) 2-d MDS visualization based on the similarity matrix. (Third row) Distribution of *NN-Radius* for all graphs in each class. (Last row) Distribution of *NN-Disagreement%* for all graphs in each class.

- Q4: Does the end-to-end method have advantage over two-stage methods?

We answer these questions sequentially in the following sections A1 (Sec. 7.3.4), A2 (Sec. 7.3.4), A3 (Sec. 7.3.4), A4 (Sec. 7.3.4). Besides observations, we provide our understanding and hypothesis. Finally, we point out three key questions that we believe are important to GLOD task (Sec. 7.3.4). As discussed before, we divide the datasets into "X&Non-X" and "X&Y" type. We evaluate propagation based two-stage methods (WL, PK), end-to-end propagation based method (OCGIN), and graph embedding based two-stage methods (Graph2Vec, FGSD) for all datasets. Table 7.4 shows all the results on "X&Non-X" type datasets, and Table 7.5 contains all results on "X&Y" type datasets. We carefully analyze these results to answer the above questions next.

A1: Performance flip occurs across all methods.

To help summarize results we use yellow color to mark performance flip in Table 7.4 and Table 7.5. The dominance of yellow for both propagation based (first row-wise block) and non-propagation based methods (second row-wise block) empirically verify that performance flip is not restricted to propagation based methods but arises more generally for other methods as well.

The observation strengths the prevalence of performance flip for GLOD task, and raises the question of how to evaluate models for GLOD if the issue is persistent across datasets and models.

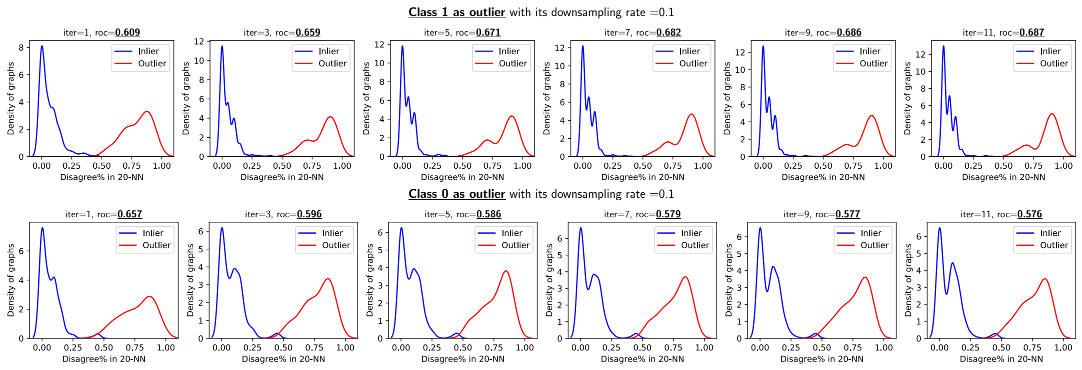


FIGURE 7.14: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over WL iterations (left to right) on IMDB-BINARY.

TABLE 7.4: Average ROC-AUC (over 10 random seeds) of 5 different graph embedding methods and 3 different outlier detectors over 5 "X&Non-X " type datasets. Each dataset has 2 down-sampled variants. 'DC' stands for down-sampled class, which is also outlier class. Cells colored with Red, Green, Yellow represent: performance of both variants are worse than random, both variants are better than random, and performance flip scenario, respectively. Performance flip is widely observed. Among all cases, **67.3%** have performance gap$\geq$ 0.2, **52.7%** cases have performance gap $\geq$ 0.3, **30.9%** have performance gap$\geq$ 0.4. (G2V=Graph2Vec)

Methods	DD		PROTEINS		NCI1		Mutagenicity		AIDS	
	DC=0 "X"	DC=1 "Non-X"	DC=0 "X"	DC=1 "Non-X"	DC=0 "Non-X"	DC=1 "X"	DC=0 "X"	DC=1 "Non-X"	DC=0 "X"	DC=1 "Non-X"
WL-LOF	0.186	0.815	0.276	0.664	0.730	0.349	0.460	0.629	0.193	0.950
PK-LOF	0.194	0.824	0.389	0.557	0.678	0.366	0.480	0.613	0.387	0.896
WL-OCSVM	0.179	0.820	0.189	0.794	0.653	0.341	0.500	0.540	0.048	0.972
PK-OCSVM	0.222	0.809	0.244	0.751	0.593	0.429	0.517	0.541	0.175	0.880
OCGIN-5	0.327	0.720	0.370	0.681	0.643	0.467	0.503	0.650	0.200	0.922
G2V-LOF	0.680	0.362	0.644	0.414	0.594	0.588	0.495	0.616	0.919	0.424
FGSD-LOF	0.628	0.425	0.468	0.422	0.712	0.417	0.458	0.634	0.934	0.290
G2V-OCSVM	0.631	0.336	0.569	0.466	0.332	0.634	0.492	0.491	0.903	0.035
FGSD-OCSVM	0.384	0.781	0.385	0.711	0.545	0.550	0.366	0.664	0.979	0.743
G2V-IF	0.656	0.335	0.552	0.449	0.341	0.636	0.446	0.586	0.904	0.035
FGSD-IF	0.745	0.400	0.773	0.272	0.384	0.637	0.476	0.569	0.984	0.018

TABLE 7.5: Same configuration as Table 7.4, this time over 5 "X&Y " type datasets. FGSD cannot run over REDDIT datasets. Performance flip is not observed for IMDB dataset across all methods. OCGIN has performance above random across all datasets. Performance flip is still widely observed for other methods. Among all cases, **30.6%** have performance gap$\geq$ 0.2, **22.4%** cases have performance gap $\geq$ 0.3, **12.2%** have performance gap$\geq$ 0.4.(G2V=Graph2Vec)

Methods	IMDB-BINARY		ENZYMES(c0&c1)		ENZYMES(c2&c3)		REDDIT(c0&c1)		REDDIT(c2&c3)	
	DC=0	DC=1	DC=0	DC=1	DC=2	DC=3	DC=0	DC=1	DC=2	DC=3
WL-LOF	0.603	0.651	0.518	0.519	0.758	0.399	0.261	0.734	0.189	0.810
PK-LOF	0.624	0.581	0.553	0.550	0.755	0.633	0.485	0.579	0.361	0.640
WL-OCSVM	0.524	0.571	0.598	0.385	0.607	0.462	0.268	0.739	0.180	0.821
PK-OCSVM	0.538	0.548	0.590	0.402	0.456	0.642	0.388	0.696	0.207	0.802
OCGIN-5	0.643	0.508	0.615	0.517	0.587	0.636	0.662	0.622	0.573	0.608
G2V-LOF	0.534	0.558	0.551	0.354	0.398	0.532	0.376	0.563	0.678	0.323
FGSD-LOF	0.606	0.505	0.644	0.412	0.449	0.608	-	-	-	-
G2V-OCSVM	0.526	0.551	0.565	0.419	0.492	0.649	0.465	0.528	0.679	0.329
FGSD-OCSVM	0.516	0.586	0.531	0.503	0.523	0.613	-	-	-	-
G2V-IF	0.516	0.562	0.572	0.346	0.410	0.604	0.505	0.545	0.695	0.326
FGSD-IF	0.522	0.617	0.537	0.456	0.300	0.708	-	-	-	-

A2: Dataset semantics play a role in performance flip.

Clearly, the percentage of yellow cells in Table 7.4 is noticeably larger than that in Table 7.4, which suggests that **(1)** performance flip occurs **more often** in "X&Non-X " type datasets than "X&Y " type datasets. We further investigate the performance gap (i.e. severity of performance flip) in both tables numerically. Table 7.4 has 67.3% cases with performance gap ≥ 0.2, 52.7% cases with performance gap ≥ 0.3, and 30.9% cases with performance gap ≥ 0.4. In contrast, Table 7.5 has only 30.6% cases with performance gap ≥ 0.2, 22.4% cases with performance gap ≥ 0.3, and 12.2% cases with performance gap ≥ 0.4. The dramatic difference of performance gap between two types of datasets suggests that **(2)** performance flip applies **more severely** to "X&Non-X " type datasets than "X&Y " type datasets. Finally, we also investigate which version of the downsample achieves high ROC-AUC. In Table 7.4 we annotate class labels 0 and 1 with corresponding "X" and "Non-X" based on class semantics, where "X" refers to the compact class. Interestingly, for propagation based methods, down-sampling "Non-X" class as outlier class consistently achieves high performance while down-sampling "X" has performance worse than random. We conclude that **(3)** downsampling the class with diverse patterns creates sparser, dispersed outliers in the embedding space when using propagation based methods, resulting in an easier outlier detection task with high ROC-AUC.

We remark that the correlation between class semantic and performance flip challenges whether we should repurpose $X\&Non-X$ type graph classification dataset to evaluate GLOD. Further, one may argue we can always down-sample class "Non-X" as outlier to evaluate, however the "Non-X" class may not be a semantic outlier class (see Mutagenicity where "Non-X" represents non-mutagen that should be normal class in semantic), and for other non-propagation based methods the performance is not always high (see AIDS).

A3: Both embedding method and outlier detector affect performance flip.

Previously we have observed that the propagation based methods consistently achieve high ROC-AUC when "Non-X" is down-sampled as outlier in "X&Non-X" datasets. However this is not always true for other methods and datasets. We observe that both graph embedding and downstream outlier detector affect which version of down-sample has high performance. See for example the AIDS dataset in Table 7.4, where using different graph embedding methods has dramatically reversed performance flip pattern. Also see the DD dataset where FGSD+OCSVM and FGSD+LOF have reversed performance flip. The observation can be explained by the driving mechanism identified in Sec.7.2.2. The performance of two-stage methods is determined by the (mis)alignment of embedding space generated by graph embedding methods as well as the assumptions of downstream outlier detector. In that sense, both graph embedding and outlier detection methods play a role in whether performance flip occurs and the amount of performance gap.

A4: End-to-end method can partially capture distribution of majority.

All two-stage methods we studied are unsupervised and produce deterministic embeddings regardless of dataset distribution. This is not ideal for outlier detection, as they do not have enough ability to generate embeddings that capture distribution of the majority class. On the other hand, end-to-end methods learn from the input dataset by optimizing a loss (e.g. reconstruction error or distance to center) contributed by each sample in the dataset, and should have certain degree of ability to capture the majority (by achieving lower average loss for the majority class). As we can see in Table 7.5, OCGIN is the only method that does not have performance flip for all 5 “X&Y” datasets. This empirically shows certain ability of capturing majority class for OCGIN. Nevertheless, performance flip still occurs widely for OCGIN in all “X&Non-X” datasets. Our hypothesis is that although OCGIN has certain ability to capture majority, the inductive bias (underlying mechanism) shared across all propagation based methods is too strong on “X&Non-X” datasets to be corrected by learning.

The relatively higher robustness of our end-to-end method, OCGIN, against performance flip, especially on “X&Y” type datasets, appears to be a promising direction for future work. This also motivates further analysis of other end-to-end GLOD methods.

Three key questions for GLOD

Based on our observations over all datasets and methods, we pinpoint three important questions for GLOD and aim to draw the community’s attention toward answering them:

- Given the widely observed performance flip issue across models and datasets, what is the best way to evaluate detectors when repurposing graph classification datasets? Is it fair to compare methods based on their average performance across all datasets, as one would usually do, while we now expect that some downsample versions will likely yield significantly worse-than-random performance?
- As graph embedding method play a significant role on GLOD, how can we design better graph-level embedding methods (either unsupervised or end-to-end) to overcome performance flip issue specifically, and achieve better GLOD performance overall at large?
- Given that methods do not agree on which version of downsample yields high performance, given any new detection task (i.e. dataset) and a set of candidate GLOD methods, how can we design unsupervised model selection strategies that can effectively choose a model that achieves high detection performance?

We believe these questions stand on the foundation of GLOD and provide solid starting points for the community toward progress on GLOD problems.

7.4 Conclusion

7.4.1 Summary

Many researchers and practitioners repurpose binary classification datasets for outlier detection via down-sampling one of the classes to constitute the outliers. In graph-level outlier detection, we found that most binary classification datasets, when used in this fashion, introduce class-level bias for different variants of downsample, resulting in dramatically different outlier detection performance. In this paper, we call this phenomenon “performance flip”. This bias exhibits itself as disparity in detection task difficulty determined by the alignment of graph embedding density and assumption of downstream outlier detector. Looking over embedding space, the issue stems from two factors: (1) density disparity; one class being more compact than the other in the embedding space; and (2) overlapping support; mixing of the embeddings between classes. Given that most outlier detectors assume a compact set of inliers and scattered outliers, down-sampling the class that is sparse in the embedding space induces such a scenario with scattered outliers and hence achieves high detecting performance. In contrast, down-sampling the other class with denser embeddings induces clustered outliers (that shares support with normal class) and results in significantly worse-than-random performance.

Under unsupervised learning, the embedding space is determined by two parts: the input dataset and the embedding method. These two parts can interact with each other. Categorizing datasets into two as 'X&Non-X' and 'X&Y' reveals that 'X&Non-X' type datasets have higher probability of performance flip with more severe performance gap. Empirical results of propagation based methods on all 'X&Non-X' type datasets suggest that the class with semantically diverse patterns can result in sparser embeddings and down-sampling this class leads to high performance. For non-propagation based methods, on the other hand, we observed a reversed pattern of which variant of downsample yields high performance as compared with that for propagation based methods. This suggests that it is hard to predict which class will appear sparser in the embedding space by solely knowing the data semantics, as it also depends on the embedding method itself.

Considering the importance of GNN based methods and its connection to propagation based methods, we carefully conducted a measurement study for certain propagation-based graph representation learning methods, namely Weisfeiler-Leman (WL) and propagation kernel (PK), and found the interaction between the mentioned factors and the sparsification property—where kernel similarity of two graphs decrease with increasing number of propagations. This property leads to amplified density disparity, as the originally sparser class of graphs sparsify at a relatively higher rate. This is either a blessing or a curse for detection, depending on which scenario is considered as a testbed. When class with diverse patterns is down-sampled, sparsification helps disperse the outliers further, making the task easier. Otherwise, it makes the inliers look more dispersed. Sharing the similar underlying mechanism, OCGIN, the one-class end-to-end GNN based oulier detector we developed, has similar performance behavior on all 'X&Non-X' datasets with WL and PK. However being an end-to-end trainable method, OCGIN has the ability of learning its parameters based mostly on the majority class, and surprisingly eliminates performance flip issue on all 'X&Y' datasets.

Based on all our observations and analysis, we remark three important questions regarding GLOD as fundamental problems in this area, from three different perspectives: evaluation, model selection, and graph embedding method. Given the widely observed performance flip issue across models, traditional evaluation of models that simply averages performance across all datasets becomes problematic, as including worse-than-random performance does not appear to be the right thing. To avoid the worse-than-random scenario for a given new dataset, one may argue that doing model selection from among all candidate models to pick up one model without worse-than-random performance would be a good way out in practice. But one should also consider the hardness of unsupervised model selection and the high risk of picking up the wrong model with poor performance. Another way to avoid evaluation under performance flip is to design better graph embedding methods that can surpass the performance flip issue.

7.4.2 Discussion

The three problems we outlined are essential but hard to answer. We present a discussion of our arguments in the following.

Considering the performance flip is persistent and hard to avoid, caution should be taken when evaluating new model(s) under performance flip. Based on Hawkins's definition of outliers [Haw80] – that the outliers are simply the minority instances that are generated by a different mechanism – no matter how we down-sample, defining outlier as the down-sampled class is meaningful. Then, evaluation by averaging performance across all datasets (including all variants of downsample) should be used. However, when worse-than-random performances are included, which skews the average performance, a model that appears competitive may result in considerably poor performance for certain cases. Another way to define the outlier class is based on dataset class semantics, where one class may be seen as a natural outlier class in the real world. However not all datasets have semantic outliers, and sometimes which class is outlier may be subjective and context-dependent (like in dataset AIDS, being active against HIV could be deemed as outlier since HIV is rare and semantically important, yet in the context of developing new drugs for treating AIDS, inactive drugs could be treated as the outliers).

A way to sidestep performance flip altogether is to design better graph embedding methods that can overcome density disparity and overlapping support. One can imagine that if the unsupervised embedding method can generate non-overlapping and clustered embeddings for each class, then performance flip would not arise. In other words, two-stage models are guaranteed to perform well for outlier detection if their graph embedding method can achieve good performance in graph-level clustering task. Another promising direction is to design effective end-to-end models that can generate clustered representation for the majority class, by making use of the natural property of learning. However the current end-to-end graph representation learning models are all message-passing based graph neural networks which suffer from the inductive bias shared by WL and PK.

In the prevalence of performance flip issue across existing models, it is not clear how to do fair and accurate model evaluation so as to pinpoint competitive models for GLOD. Assuming it is also hard to design new graph-level embedding models that can bypass performance flip no matter what dataset they are presented with, an open question remains as how to do GLOD in practice, i.e. in the wild, when presented with a new GLOD task. The answer is not trivial. Model evaluation and benchmarking is done exactly to identify a few competitive models to employ in practice. When evaluation has issues we have identified, a way out appears to do model selection. That is, instead of employing the "best" model based on average performance across many historical/benchmark datasets, the goal would become designing an effective model selection strategy to choose a model to employ from a pool of existing models for a new task. Unfortunately, however, unsupervised model selection for GLOD is notoriously difficult in the absence of any labels [ZRA20], and is perhaps an equally hard, if not harder, problem than grappling with performance flip and model evaluation (where labels are available for benchmark datasets, for evaluation). As with performance flip, it incurs the risk of selecting a poor model. An easier yet practical direction may be to do model selection using (only) a small amount of labeled data, which however, has its own issues such as the representativeness of outliers in a small sample.

7.4.3 Future work

Several future directions that immediately span out of this study are three. First, whether our findings regarding the sparsification property of certain propagation based models have negative implications on (graph-level) clustering and classification tasks warrants further research. Second, whether our findings transfer to non-graph settings requires future analysis. We find that there is historical evidence that they do.[5] Finally, our study can be extended to the analysis of class-level bias in model errors when multi-class classification datasets (with greater than two classes) are used for outlier benchmark creation.

[5]Arrhythmia, a (point-cloud) binary classification dataset from UCI Machine Learning Repository has been repurposed for outlier mining, by downsampling either one of the classes, `http://homepage.tudelft.nl/n9d04/occ/514/oc_514.html` versus `http://homepage.tudelft.nl/n9d04/occ/515/oc_515.html`, respectively inducing scattered versus clustered outliers. Notice that the performance of the detection models are significantly higher for the former scenario.

Chapter 8

Anomaly Detection of Attributed Multi-graphs with Metadata

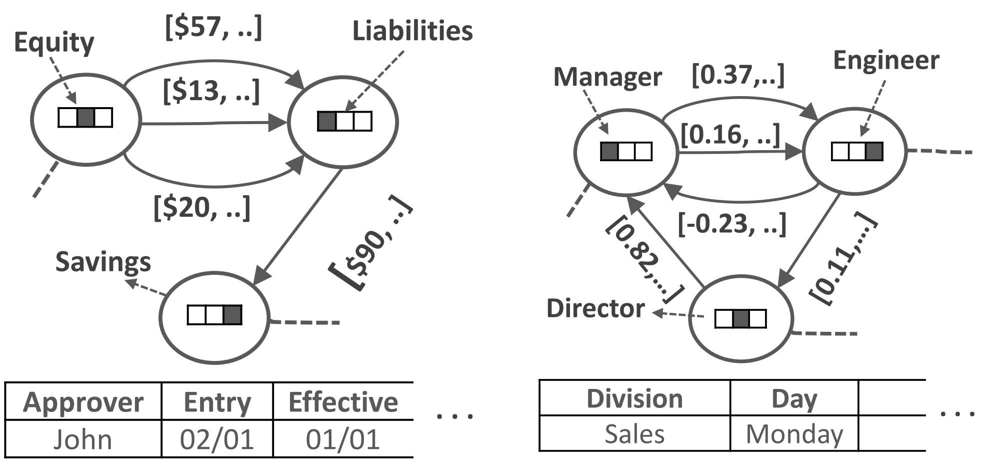
Chapter is based on Konstantinos Sotiropoulos*, Lingxiao Zhao*, Pierre Jinghong Liang, and Leman Akoglu. "ADAMM: Anomaly Detection of Attributed Multi-graphs with Metadata: A Unified Neural Network Approach". In: *2023 IEEE International Conference on Big Data (BigData)*. IEEE. 2023, pp. 865–874.

8.1 Introduction

Anomaly detection finds numerous practical applications in finance, manufacturing, monitoring, etc. as anomalies are typically indicators of faults, inefficiencies, malicious behavior, etc. in various real-world systems. One of the key challenges in real world settings is the complexity of the data, which exhibit multiple different modalities and heterogeneity—requiring new data representations and novel modeling designs.

This work is motivated by anomaly detection problems various real-world domains. The first example is from business management and particularly accounting/auditing, where the goal is to identify abnormalities (errors or fraud) among annual general-ledger journal entries from a given firm. Each entry consists of a series of line items of credit or debit transactions of various amounts between accounts with the total debited dollar amount equal to the total credited amount, following the double-entry bookkeeping rules. Accordingly, debits and credits within an entry create directed and weighted links between accounts, and multiple transactions may occur between the same pair of accounts or even within a single general-ledger account. Besides the relational information, each journal entry is also associated with meta-features, such as the approver, entry and effective dates, etc. This poses a multi-modal (relational and tabular) data problem setting. A second example arises from communication networks, where the problem is detecting significant events within a company based on e-mails exchanged between different entities.

Our goal is to design a novel solution that not only unifies the data modalities under a single, flexible model capable of managing complex relations based on directed multi-graphs, but also offers broad applicability across the domains mentioned earlier and potentially beyond. To this end, we represent the relational information with a node- and edge-attributed directed multi-graph, and the auxiliary or meta-data as tabular meta-features. (See Figure 8.1.) Our proposed solution, ADAMM (for <u>A</u>nomaly <u>D</u>etection of <u>A</u>ttributed <u>M</u>ulti-graphs with <u>M</u>etadata), is a unified neural network framework that learns an expressive graph-level representation for directed and attributed multi-graphs and then fuses it with the meta-features within a shared embedding space before feeding the joint embedding to an unsupervised anomaly



Approver	Entry	Effective	. . .
John	02/01	01/01	

Division	Day		. . .
Sales	Monday		

FIGURE 8.1: Modeling complex data. (left) E.g. from **accounting**: a journal entry's attributed *multi-graph* (multiple transactions between two accounts), with edge directions (credit/debit), edge features (e.g. $ amount), and node features (account type; e.g. equity, savings, etc.) plus aux. *meta-features* (approver, entry date, etc.); (right) E.g. from **communication networks**: a daily activity *multi-graph* (multiple e-mails between two company employees), with edge directions (to/from), edge features (e.g. text embedding) and node features (role in the company.) plus aux. *meta-features* (division, day, etc.).

detection objective. Notably, our objective is crafted to handle data heterogeneity; where for example, the journal entries may form multiple clusters (e.g. purchases vs. interest gains), and individual behaviors can reflect socio-demographic groups (e.g. single vs. married-with-children). Specifically, we replace the classic SVDD objective that aims to learn embeddings tightly centered around a single centroid [TD04], and instead employ an unsupervised loss to accommodate multiple centroids.

The literature is abound with anomaly detection techniques [Agg17; Pan+21; Gup+13; Cho+21; ATK15; Ma+21], where a vast body focuses on uni-modal data. Numerous prior work address outliers in tabular data [Agg17; Pan+21], possible due to its wide presence in industry and its efficient storage in databases. However, molding real world anomaly detection problems to tabular outlier detection requires "flattening" data from all modalities into manually-extracted features via laborious and often costly domain-expertise [Ako21].

On the other hand, graph anomaly detection has been studied mainly on a single graph for detecting node/edge-level anomalies, with much less emphasis on graph-level anomalies [ATK15; Ma+21]. Few existing traditional approaches to attributed multi-graph anomalies [Lee+21; NLA23] that are not neural network based are not learnable, restricted to handling single-value edge features, not scalable for larger graphs, and do not take auxiliary metadata into account. Similarly, the more recent neural network based models [ZA23; Qiu+22; Zha+22d; Zha+22a] are not designed to accommodate directed multi-graphs or graphs with metadata as in this work. (See related work in 1.2 for details.) Finally, we argue that a straightforward two-stage approach is naïve and nontrivial; the reasons are first, treating data modalities/sources separately misses the opportunity to capture inter-dependencies and second, the problem of how to combine multiple anomaly rankings/scores open many possibilities without a principled way to choose in the absence of any labels.

- **Anomaly Detection in Real-World Settings with Complex Data:** We formulate anomaly detection under data complexity/variety, exhibiting relational as well as auxiliary information, in an elegant framework that can jointly handle complex graphs with node/edge attributes, edge multiplicities, directions and self-loops, meta-features, as well as data heterogeneity. The formulation is driven by anomaly detection problems from two different real-world domains, namely accounting and human mobility, yet is general to apply to possibly other domains.

- **A Unified Detection Model:** We introduce ADAMM, a novel neural network architecture that can digest the aforementioned multi-modal data toward anomaly detection in a unified fashion. It tackles edge multiplicities through set representation learning, employs expressive graph-level embedding that is fused with meta-features in a learned shared embedding space, and finally, optimizes an unsupervised anomaly loss that can accommodate heterogeneous data with multiple latent underlying clusters.
- **Generality and Applications:** ADAMM offers a general framework, where the architecture can be extended to several other domains with data variety, using the idea of learning joint/shared-space embeddings and end-to-end anomaly loss optimization. Besides addressing the data variety challenge of big data, our ADAMM also targets business and societal value, as it is applied to two high-stakes domains; accounting (finance) and human mobility (urban). Through extensive experiments, we show that two-stage solutions are blind-sided and that ADAMM outperforms those as well as other existing baselines significantly on accounting data from three different firms, as well as human GPS trajectory simulations.

Reproducibility. To foster future work on anomaly detection on complex multi-graphs with metadata as well as for practical applications, we open-source the code for ADAMM at https://github.com/konsotirop/ADAMM.

8.2 Related Work

Anomaly detection (AD) has an extensive literature mainly considering outliers in tabular or vector data [CBK09; Agg15], including the recently emerging deep neural network based approaches (see surveys [CC19; Ruf+20; Pan+21]). However, these do not apply to AD for graphs with relational structure.

The majority of work on graph anomaly detection [ATK15], including the recent graph NN (GNN) based techniques [YHL15; Yu+18; Din+19; Wan+20] focus on node, edge, or subgraph anomalies within a *single* graph, rather than graph-level anomalies in a *database*.

Different from these earlier work, we consider graph-level AD among a set of graphs within a database. There exist traditional encoding or compression-based techniques [NC03; EHC09; Lee+21; NLA23] that aim to identify frequent structural motifs or graphlets that compress a graph database efficiently, and then flag those graphs with long encoding length as anomalous. Most recent work have shifted attention to employing deep learning and GNNs toward graph-based AD (for a recent survey, see [Ma+21]). The idea is to flatten each graph by leveraging their representation or embedding learning capability, and train the GNN parameters end-to-end through various AD objectives such as one-class [ZA23; Qiu+22], mutual information-based [Zha+22a], distributional distance [Zha+22d], contrastive [Luo+22] as well as distillation losses [Ma+22]. While these have made progress in graph-level anomaly detection, they do not handle *multi*-graphs, nor are they designed to admit multi-modal input such as graphs with meta-features as in our case.

Examples of prior work on multi-graphs address summarization [BLA22], partitioning [TLD09; PAI13; Kan+20], as well as anomaly detection [NLA23; MGF11], however without considering additional meta-features. An earlier work on node-level (fake reviewers) AD in a single (reviewer-to-product) graph has attempted to bridge node-level meta-features with graph data—by first using the meta-features to estimate node outlierness scores and then propagating those over the graph to capture guilt-by-association [RA15]. Their method, however, does not generalize to graph database anomalies with graph-level meta-features.

In summary our proposed ADAMM, to our knowledge, is the first method for graph-level anomaly detection for directed node/edge-attributed multi-graphs with meta-features. It leverages (*i*) end-to-end *multi*-graph embedding, (*ii*) *joint* multi-modal representation learning and (*iii*) a *multi-centroid* AD loss to effectively capture complexities in the input data.

8.3 Preliminaries

We consider anomaly detection on a large database $\mathcal{G} = \{(G_1, M_1), \ldots, (G_n, M_n)\}$ of n pairs of directed, node/edge attributed, multi-graphs (multiple edges may exist between two endpoints), and their associated metadata-level features.

Definition 8.3.1. (Directed, attributed, multi-graph). A graph $G_i = (V_i, E_i, \tau)$ is a directed, attributed, multi-graph, endowed with a function $\tau : V_i \mapsto \mathbb{R}^d$ that assigns a real-valued feature vector to every node in G_i. Moreover, E_i is a multi-set, where an element $e_t = (u, v, \mathbf{f}_t)$ is a directed edge between nodes u and v associated with an edge-feature vector $\mathbf{f}_t \in \mathbb{R}^k$.

Definition 8.3.2. (Metadata). Each graph G_i is associated with a vector $\mathbf{Z_{M_i}} \in \mathbb{R}^{d_M}$ reflecting tabular features.

Usually, we operate on sets (or multi-sets) of variable lengths, where there is no specific order of the elements. In such cases, we need functions that are permutation invariant.

Definition 8.3.3. (Set-function). A function f acting on sets is called a **set function** if it is permutation invariant to the order of objects in the set. That is, for any permutation $\pi : f(\{x_1, \ldots, x_n\}) = f(\{x_{\pi(1)}, \ldots, x_{\pi(n)}\}$.

Neural network architectures, like DeepSet [Zah+17], can implement arbitrary set functions, while the work of Xu et al. [Xu+19] extends such functions for multi-sets.

Graph Neural Network (GNN) model: We use the provably expressive GIN model of [Xu+19], where the embedding of a node v is updated during the l^{th} layer/iteration using the following aggregation function:

$$\mathbf{x}_v^{(l)} = MLP^{(l)}((1+\epsilon)\cdot \mathbf{x}_v^{(l-1)} + \sum_{u\in\mathcal{N}(v)} \text{ReLU}(\mathbf{x}_u + \mathbf{f}_{vu}); \boldsymbol{\theta}_l) \tag{8.1}$$

where MLP is a multi-layer perceptron, ϵ a learnable parameter, $\mathcal{N}(v)$ the neighborhood of node v, $\mathbf{f}_{uv}$ is the feature vector of edge (u, v) and $\boldsymbol{\theta}_l$ a vector of trainable parameters.

To obtain a graph-level representation $\mathbf{Z}_G$ for the whole graph G we can use a permutation-invariant function READOUT that aggregates node embeddings after the final layer/iteration L, i.e.,

$$\mathbf{Z}_G = READOUT(\{\mathbf{x}_v^{(L)} | v \in V\}) \tag{8.2}$$

Our problem can be defined (informally) as follows:

Problem 1. (Anomaly Detection of Attributed Multi-graphs with Metadata (ADAMM).) Given a database $\mathcal{G} = \{(G_i, M_i)\}_{i=1}^{n}$ of n node- and edge-attributed multi-graphs and their associated metadata; the goal is to identify the abnormal (graph, metadata) pairs that differ significantly from the majority in the database.

8.4 ADAMM: Anomaly Detection of Multi-graphs with Metadata

8.4.1 Data Representation

The input to ADAMM is a database comprising of pairs of graphs and their associated metadata vectors. In what follows, we describe the capabilities of it in representing complex graph data, the fusion of them with metadata vectors, as well as two concrete examples from the accounting and human mobility domains where this unified representation can be used to model real-world scenarios.

Graph representation. ADAMM is designed to handle complex graph data of virtually any type. Specific design choices allow the model to be able to represent:

1. *Node attributes (or Node labels):* Nodes can have attributes that are updated after each graph convolution step. When nodes do not have attribute vectors but categorical labels, we can learn representations for these node labels using an embedding layer.

2. *Edge features:* Edges can have features containing important information about the link between two nodes in the graph. Thus, node representations are updated taking into account not only the embeddings of the neighboring nodes, but also those of incident edges (see also Eq. Eq. (8.1)).

3. *Edge direction:* In various domains as with transactions (accounting) and trips (mobility), edge direction is semantically important. Thus, we enhance edge features with an encoding of the direction of the edge. Specifically, for each multi-edge $(u, v, \mathbf{f}_t)$, between nodes u and v, we encode it using label "1" if the edge is present in the graph, i.e. $(u, v, \mathbf{f}_t, \text{“1”})$, and in the meanwhile, augment another reversed edge $(v, u, \mathbf{f}_t, \text{“2”})$ with label "2" into the graph. Also, we reserve label "0" for self-loop edges, i.e. $(u, u, \mathbf{f}_t)$ becomes $(u, u, \mathbf{f}_t, \text{“0”})$. These edge direction labels are then applied as input to an embedding layer that produces the edge direction representation vector $\mathbf{d}_t$. The final representation vector $\mathbf{f}'_t$ for the edge e_t is then obtained as the sum of the edge features vector $\mathbf{f}_t$ and of the edge direction vector, that is, $\mathbf{f}'_t = \mathbf{f}_t + \mathbf{d}_t$.

4. *Multiple edges:* Currently, GNNs are not able to handle multi-edges. Instead, they assume there exists a unique edge between two nodes, as in Eq. Eq. (8.1) for GIN convolution. Multi-edges, however, model the multiple interactions that can occur between two nodes in a network and each has its own feature vector. ADAMM is designed to handle multi-edges by learning a single edge representation from the multi-set of edges. More precisely, we treat the edge features of the multi-edges $F = \{\mathbf{f}'_1, \ldots, \mathbf{f}'_T\}$ as a multi-set and we use a permutation-invariant multi-set function $f : F \mapsto \mathbb{R}^{d_e}$ to learn an edge-level d_e-dimensional representation vector.

The versatility of ADAMM in handling complex graph-data allows it to be used in a wide variety of domains. We present two exemplar ones as follows.

(i) Bookkeeping Graphs [Lia23]: Each graph is a representation of a *journal entry*: a detailed transaction record. Every account present in the entry is associated with a node, with its label being the account type (e.g. equity, revenue, etc.). A directed edge represents monetary flow from a credited account to a debited account and the feature of an edge is the monetary value associated with this transaction. Directed multi-edges capture multiple credit/debit flows that can take place between two accounts.

(ii) Human Mobility (or Activity) Graphs [Sch+13]: These represent the mobility or activity behavior of an agent within a time-frame (e.g., a day of the week). Nodes represent visited locations, while node labels represent the Points Of Interest (POI) type in that location (school, restaurant, etc.). Directed multi-edges stand for the trips between those locations and their features capture information about the trip (duration, distance, etc.).

Metadata representation. ADAMM is able to fuse the graph-level representation with associated metadata vectors that contain auxiliary information. We give examples of such information in the two aforementioned domains below.

(i) Metadata for Bookkeeping Graphs: The metadata vector contains information regarding the ID of the user that created the specific journal entry, the approver of this entry, total credit amount, a binary indicator of whether it is a reversal, the date transactions took effect, or the date transactions were recorded in the journal, etc.

(ii) Metadata for Human Mobility Graphs: For activity graphs the metadata vector could contain information about the day of the week this activity took place (e.g., Tuesday), a vector representation of the agent it describes (or simply a unique ID), or other information that could contain GPS related information, like speed-limit violations, etc.

8.4.2 A Unified Neural Network Architecture

ADAMM provides a unified architecture for anomaly detection in a database of graphs and their associated metadata features. Figure 8.2 presents an overview of our model and the steps it involves. The input is two-pronged: a directed, node/edge attributed multi-graph and its associated metadata vector. Following the set representation learning of the multi-edges, a GNN is employed to learn a graph-level embedding, which is then fused with the metadata vector to obtain the final joint embedding. A parameter estimation network decides on the (soft) membership of the final embeddings to one of K clusters. ADAMM is trained in an end-to-end fashion, i.e. all of its parameters are optimized jointly with respect to a suitable objective function that minimizes the weighted distance of embeddings to the K centroids of the clusters. Additional regularization terms are introduced to spread-out the centroids as well as to nudge the estimation network toward more confident assignments of cluster memberships. We describe each of these steps in greater detail next.

Graph-level Embedding

We learn a graph-level embedding in two steps:

- Multi-edge Representation Learning: As noted, GNNs (including GIN) can **not** readily handle multi-edges. For this reason, we “flatten" all directed multi-edges between two nodes to a single undirected edge and its associated feature vector by learning a permutation-invariant multi-set function based on a

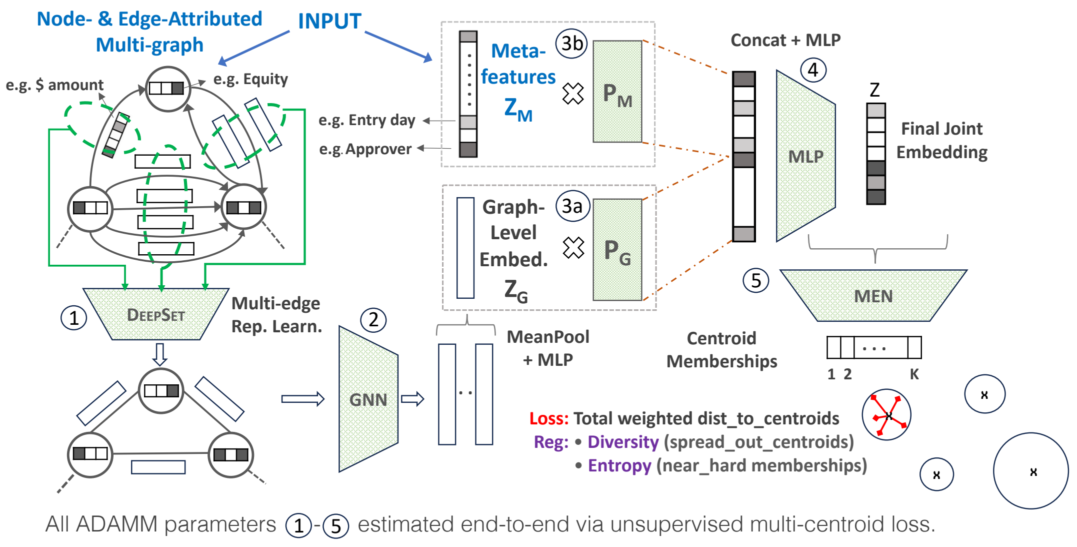

Node- & Edge-Attributed Multi-graph
INPUT
e.g. $ amount
e.g. Equity
e.g. Entry day
e.g. Approver
Meta-features Z_M
3b
P_M
Graph-Level Embed. Z_G
3a
P_G
Concat + MLP
4
MLP
Z
Final Joint Embedding
1
DEEPSET
Multi-edge Rep. Learn.
2
GNN
MeanPool + MLP
Centroid Memberships
5
MEN
1 2 K
Loss: Total weighted dist_to_centroids
Reg: • Diversity (spread_out_centroids)
• Entropy (near_hard memberships)
All ADAMM parameters ①-⑤ estimated end-to-end via unsupervised multi-centroid loss.

FIGURE 8.2: A workflow overview of the ADAMM architecture. Given two-pronged input (in blue), i.e. attributed multi-graph and metadata, ADAMM first processes the former by ① learning a multi-set representation of the multi-edges, and ② flattening the resulting graph via GNN into node representations that are pooled into a graph-level embedding. Then, ③ graph-level embed. and meta-features are projected, ④ followed with a joint embedding learning. Finally, ⑤ the output layer employs an unsupervised regularized multi-centroid anomaly loss where soft assignments are learned via a membership estimation network (MEN). It is notable that ADAMM provides a unified multi-modal framework where parameters of all the modules (in green) are estimated end-to-end.

DeepSet[Zah+17] architecture. This procedure is depicted in Figure 8.2 (see step ①), where the input attributed multi-graph is transformed using a learnable multi-set function to an attributed graph with single edges among its pairs of nodes.

- Node Embeddings: The transformed graph, where all multi-edges have been replaced by an attributed single edge, is used as input to a GNN model (step ② in Figure 8.2). In ADAMM we opt to use GIN [Xu+19] as a provably expressive GNN. GIN learns node embeddings by performing the graph convolution of Eq. Eq. (8.1), also incorporating the edge features.

To obtain a graph-level embedding we use a READOUT function on the node embeddings (see Eq. 8.2). We implement this function by performing mean pooling over the node embeddings followed by a multi-layer perceptron (MLP) to learn the final graph-level embedding as

$$\mathbf{Z}_G = MLP\left(\frac{1}{|V|}\sum_{v\in V}\mathbf{x}_v^{(L)};\boldsymbol{\theta}_G\right) \ . \tag{8.3}$$

A Unifying Embedding Space for Graph and Metadata

After having obtained a graph-level embedding, we learn a joint representation of the graph and its metadata in a unifying embedding space. We are motivated by the CLIP-style latents [Ram+22] that learn a shared embedding space for images and their associated text captions, analogous to our graph and metadata pairs. More precisely, we first linearly project the graph-level embedding vector $\mathbf{Z}_G$ as well as the metadata vector $\mathbf{Z}_M$ by learning two projection functions (with separate parameters) $P_G(\mathbf{Z}_G;\boldsymbol{\theta}_G) : \mathbb{R}^{d_G} \mapsto \mathbb{R}^{d_P}$ and $P_M(\mathbf{Z}_M;\boldsymbol{\theta}_M) : \mathbb{R}^{d_M} \mapsto \mathbb{R}^{d_P}$ to obtain two new vectors $\mathbf{Z}'_G$ and $\mathbf{Z}'_M$ of the same length d_P as they share the same space. We normalize both vectors to have unit l_2 norm and then concatenate them. Finally, we employ an MLP to obtain the final joint embedding, denoted $\mathbf{Z} \in \mathbb{R}^d$ (see steps ③ - ④ in Figure 8.2) as

$$\mathbf{Z} = MLP\left(\text{CONCAT}(\mathbf{Z}'_G, \mathbf{Z}'_M); \boldsymbol{\theta}_J\right) \ . \tag{8.4}$$

8.4.3 Anomaly Detection Loss

Objective functions used in anomaly detection, like One-Class DeepSVDD [Ruf+18], make the somewhat strong assumption that all of the normal instances come from the same distribution. Hence, their objective is to use a deep neural network to map all normal instances as close to the center of a *single* hypersphere, with anomalous instances identified as those mapped farther from this centroid. However, this objective does not take into account the multiple modalities or heterogeneities that may exist in real world data. For this reason, we introduce a new objective function that accommodates *multiple* clusters of the input samples. ADAMM estimates the (soft) cluster membership of each sample using a membership estimation network and tries to minimize the total average weighted distance from the K centroids, where hyperparameter K is carefully tuned (as discussed later in 8.4.4). Contrary to One-Class DeepSVDD, where the center of the hypersphere is fixed during the training process, the K centroids in our case are inferred from the membership estimation network and the final embedding vectors.

Membership Estimation Network (MEN). The MEN is an MLP with a softmax activation function (step ⑤ of Fig. 8.2) that gives the membership predictions for the final embedding vectors $\mathbf{Z}$ in Eq. Eq. (8.4). That is,

$$\widehat{\boldsymbol{\gamma}} = \text{softmax}(\text{MLP}(\mathbf{Z}; \boldsymbol{\theta}_{MEN})) \ , \tag{8.5}$$

where $\widehat{\boldsymbol{\gamma}}$ is a K-dimensional vector depicting the soft membership probability predictions of $\mathbf{Z}$.

In what follows, and for a batch of N pairs of (graph, metadata) samples, where $\mathbf{Z}_i$ is the embedding of the i^{th} sample, we denote by $\widehat{\boldsymbol{\Gamma}}$ the $N \times K$ matrix of cluster membership estimations from Eq. Eq. (8.5) and by $\widehat{\gamma}_{ik}$ each entry of this matrix.

Then, the cluster centroids $\widehat{\boldsymbol{c}}_k \in \mathbb{R}^d$ can be calculated using the cluster membership estimations from Eq. Eq. (8.5) and embedding vectors $\mathbf{Z}_i$, for $i = 1, \dots, N$, by

$$\widehat{\boldsymbol{c}}_k = \frac{\sum_{i=1}^{N} \widehat{\gamma}_{ik} \mathbf{Z}_i}{\sum_{i=1}^{N} \widehat{\gamma}_{ik}} \ . \tag{8.6}$$

Loss Function and Anomaly Score. Having estimated the embedding vectors and cluster membership estimations, the anomaly score of a sample $T_i = (G_i, M_i)$ is then defined as the weighted sum of the Euclidean distance between the final embedding vector $\mathbf{Z}_i$ and the cluster centroids $\widehat{\boldsymbol{c}}_k$'s, i.e.

$$\text{score}(T_i) = \sum_{k=1}^{K} \widehat{\gamma}_{ik} \|\mathbf{Z}_i - \widehat{\boldsymbol{c}}_k\|^2 \ , \tag{8.7}$$

which also serves as the *anomaly score* of a sample T_i (the higher, the farther and the more anomalous).

ADAMM is then trained in an end-to-end fashion to optimize the following unsupervised objective.

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \widehat{\gamma}_{ik} \|\mathbf{Z}_i - \widehat{\boldsymbol{c}}_k\|^2 + \lambda_1 \cdot H(\widehat{\boldsymbol{\Gamma}}) + \lambda_2 \cdot D(\widehat{\mathbf{C}}) \tag{8.8}$$

where $\boldsymbol{\theta}$ depicts all ADAMM parameters collectively and λ_1, λ_2 are hyperparameters that aim to strike a balance between the distance-to-centroids anomaly loss (first term) and two regularization terms, respectively, Entropy and Diversity, which we describe as follows.

- The first is an entropy regularization that forces the network to be more confident on the cluster to which it estimates an input sample to belong. More specifically, we aim to minimize the average entropy over the rows of the $\widehat{\mathbf{\Gamma}}$ membership estimation matrix as

$$H(\widehat{\mathbf{\Gamma}}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{i=1}^{K} -\widehat{\gamma}_{ik} \log(\widehat{\gamma}_{ik}) \ . \tag{8.9}$$

- The second is a diversity term that promotes separation between the cluster centroids to avoid the undesired mode-collapse solutions where the network collapses all centroids to the same point. Letting $\widehat{\mathbf{C}}$ depict the $K \times d$ matrix containing the cluster centroids $\widehat{\boldsymbol{c}}_k$'s as its rows;

$$D(\widehat{\mathbf{C}}) = -\log(\det(\mathrm{Cov}(\widehat{\mathbf{C}})) \ , \tag{8.10}$$

  where $\det(\mathrm{Cov}(\widehat{\mathbf{C}}))$ is the determinant of the covariance matrix of $\widehat{\mathbf{C}}$. In effect, the larger the determinant of the covariance matrix, the more the centroids are dispersed, promoting separation between and diversity among the cluster centroids. A similar term has also been used in [DDK12] for selecting diverse features in regression settings.

8.4.4 Model Selection

ADAMM, as with other deep neural networks based models, is configured with a set of hyperparameters (HPs), such as the number of layers, weight decay and learning rates, number of training epochs, among others. In addition, our multi-centroid anomaly objective incurs the number of centroids K, and the λ_1 and λ_2 terms from Eq. Eq. (8.8). Each different configuration of these results in a different model, with potentially drastic differences in anomaly detection effectiveness. The challenge is that anomaly detection is an unsupervised task, where we usually lack ground-truth labels of whether a sample is an anomaly. As a result, we do not have a labeled validation set for hyperparameter tuning. For this reason, we devise an unsupervised validation score, *without using any labels*, toward selecting an effective model that performs better in anomaly detection than what we would have obtained by picking at random (in absence of any other guidance).

Given a family of models, $\mathcal{M}$, we opt to choose the model m that minimizes the sum of the weighted distances of the N samples in the training set from the K centroids as our model selection criterion, specifically,

$$\sum_{i=1}^{N} \sum_{k=1}^{K} \widehat{\gamma}_{ik} \|\mathbf{Z}_i - \widehat{\boldsymbol{c}}_k\|^2 \ . \tag{8.11}$$

This rule favors the model that succeeds into learning a tight representation of the training instances into each of the K clusters by better extracting their shared patterns, which in effect helps reveal the anomalies that deviate from these patterns. In experiments, we compare the effectiveness of our model selection criterion against random picking (i.e. average/expected performance over possible HP configurations).

8.5 Experiments

8.5.1 Experimental Setup

Datasets. For evaluation we use four datasets from two different domains, each

Table 8.1: Dataset Summary Statistics

Name	Graphs	Nodes	Multi-edges	Node-attr.	Edge-attr.	Meta-feat.
SH	39,011	[1,15]	[1,338]	11	1	11
KD	152,105	[1,91]	[1,774]	10	1	9
HW	90,274	[1,25]	[1,897]	11	1	7
MobiNet	140,000	[1,22]	[1,59]	41	4	9

containing a large database of graphs and their associated metadata. Those include annual general-ledger journal entries from three different firms, in collaboration with PwC. The fourth dataset involves simulated human GPS trajectories. Summary of datasets is given in Table 8.1.

- *Accounting Datasets*: Three datasets from accounting consist of all annual journal entries from different firms anonymized as SH, HW, and KD. Each dataset contains tens of thousands of bookkeeping graphs [Lia23] capturing itemized transactions between impacted accounts along with dollar amounts, and metadata entries capturing auxiliary journal information including entry and effective date, requester, approver, reversal indicator, and so on.

- *Human Mobility Dataset*: The fourth dataset, referred to as MobiNet, contains the trajectories of $10,000$ simulated agents over a period of two weeks. We use these trajectories to extract daily activity graphs [Sch+13] of the places (i.e. POI) visited by each agent and the trips between them, as described in Section 8.4.1. The metadata contains information about individual trips and are transformed to a single vector using a DeepSet architecture during the end-to-end training of ADAMM.

Baselines. For comparison, we use as baselines existing graph-level anomaly detectors and tabular data outlier detectors. Unlike ADAMM, existing graph-level anomaly detectors can not handle multi-edges. Therefore, we collapse all multi-edges to a single edge and use the average representation of their feature vectors. To the best of our knowledge, there is also no prior work that fuses graphs and metadata and provides a single anomaly score. For this reason, we employ two-stage baselines: First, we create a ranking of the samples with respect to their anomaly score as obtained by a graph-level anomaly detector. Then, we create a second ranking by using a tabular data outlier detector. We combine these two rankings to obtain a single graph&metadata anomaly ranking using two well-established aggregation methods detailed as follows.

(*a*) *Graph-level Anomaly Detectors*: We first aim to detect graph-level anomalies using the following baselines:

  (1) Weisfeiler-Lehman (WL) graph kernel [She+11], followed by the OCSVM outlier detector [TC00] that can admit a kernel matrix as input.

  (2) graph2vec [Nar+17], for graph-level embedding, followed by the OCSVM detector.

  (3) DOMINANT [Din+19], a GNN-based node anomaly detector, from which we average the scores to obtain a graph-level anomaly score.

(*b*) *Tabular Data Outlier Detectors*: We use the tree-ensemble based Isolation Forest algorithm [LTZ08] to score outlierness on the meta-features, which is the state-of-the-art tabular data outlier detector [Emm+15].

TABLE 8.2: Anomaly Detection Results for all methods across all datasets based on AUROC. For baseline methods we run the experiments over a grid of hyperparameters and report the average performance, along with the std. dev. ADAMM employs a model selection criterion and outputs a unique ranking. Last row reports significance test results, where (**) and (***) denote that ADAMM is significantly better than baselines w.r.t. the Wilcoxon Signed Rank Test at $p = 0.05$ and $p = 0.01$, respectively.

Dataset	Anomaly Type	ADAMM	WL+BFS	WL+IR	G2V+BFS	G2V+IR	DOM.+BFS	DOM.+IR
SH	GA1	**0.992**	0.925 ± 0.01	0.922 ± 0.01	0.839 ± 0.09	0.833 ± 0.09	0.824 ± 0.01	0.821 ± 0.01
	GA2	**0.968**	0.827 ± 0.02	0.829 ± 0.02	0.854 ± 0.02	0.854 ± 0.02	0.834 ± 0.01	0.837 ± 0.01
	MA1	**0.846**	0.591 ± 0.02	0.610 ± 0.03	0.586 ± 0.01	0.592 ± 0.01	0.602 ± 0.02	0.613 ± 0.02
	MA2	**0.918**	0.638 ± 0.02	0.642± 0.02	0.614 ± 0.01	0.618 ± 0.01	0.615 ± 0.01	0.618 ± 0.01
	GA1 + MA1	**0.955**	0.899 ± 0.01	0.897 ± 0.02	0.807 ± 0.01	0.800± 0.01	0.811 ± 0.01	0.807 ± 0.02
	GA2 + MA1	**0.977**	0.841 ± 0.03	0.840 ± 0.01	0.871 ± 0.01	0.869±0.02	0.836 ± 0.02	0.838 ± 0.01
KD	GA1	**0.928**	0.885 ± 0.01	0.880 ± 0.01	0.835 ± 0.04	0.828 ± 0.04	0.462 ± 0.01	0.450 ± 0.01
	GA2	**0.939**	0.825 ± 0.03	0.828 ± 0.04	0.820 ± 0.01	0.824 ± 0.01	0.543 ± 0.01	0.528 ± 0.01
	MA1	**0.841**	0.727 ± 0.01	0.716 ± 0.03	0.729 ± 0.01	0.718 ± 0.01	0.610 ± 0.01	0.588 ± 0.01
	MA2	**0.854**	0.738 ± 0.02	0.743 ± 0.02	0.736 ± 0.02	0.741 ± 0.02	0.518 ± 0.01	0.505 ± 0.01
	GA1 + MA1	**0.933**	0.901 ± 0.01	0.895 ± 0.01	0.814 ± 0.07	0.805 ± 0.01	0.458 ± 0.01	0.448 ± 0.01
	GA2 + MA1	**0.916**	0.849 ± 0.03	0.849 ±0.04	0.818 ± 0.01	0.805 ± 0.07	0.537 ± 0.01	0.522 ± 0.01
HW	GA1	**0.973**	0.922 ± 0.01	0.916 ± 0.01	0.922 ±0.03	0.920 ± 0.03	0.713 ± 0.21	0.710 ± 0.21
	GA2	**0.994**	0.895 ± 0.01	0.888 ± 0.01	0.666 ± 0.05	0.660 ± 0.05	0.400 ± 0.07	0.406 ± 0.07
	MA2	**0.967**	0.691 ± 0.02	0.661 ± 0.02	0.706 ± 0.02	0.694 ± 0.01	0.535 ± 0.01	0.527 ± 0.01
MobiNet	GA1	0.526	0.676 ± 0.01	**0.678** ± 0.02	0.466 ± 0.07	0.467 ± 0.05	0.320 ± 0.01	0.323 ± 0.01
	GA2	0.491	0.487 ± 0.02	0.482 ± 0.03	0.499 ± 0.05	**0.505** ± 0.04	0.341± 0.01	0.346 ± 0.01
	MA3	0.441	0.558 ± 0.01	0.563 ± 0.01	0.556 ± 0.02	**0.574** ± 0.03	0.411± 0.01	0.415 ± 0.01
	MA4	0.450	0.492 ± 0.01	0.490 ± 0.01	0.517 ± 0.02	**0.524** ± 0.01	0.349± 0.01	0.353 ± 0.0
	GA1 + MA3	0.563	0.750 ± 0.01	**0.754** ± 0.02	0.451 ± 0.01	0.454 ± 0.02	0.326 ± 0.01	0.329 ± 0.03
	GA1 + MA4	0.678	0.743 ± 0.02	**0.747** ± 0.01	0.457 ± 0.02	0.460 ± 0.01	0.350± 0.01	0.353 ± 0.02
	GA2 + MA3	0.470	0.483 ± 0.01	0.480 ± 0.02	0.505 ± 0.01	**0.510** ± 0.02	0.329 ± 0.04	0.332 ± 0.01
	GA2 + MA4	0.477	0.494 ± 0.02	0.489 ± 0.01	0.520 ± 0.01	**0.526** ± 0.03	0.332 ± 0.01	0.336 ± 0.01
Average AUROC		0.787	0.730	0.728	0.678	0.677	0.524	0.522
Average Rank		2.13	3.08 (**)	2.78 (**)	4.09 (***)	3.74 (***)	6.17 (***)	6 (***)

After having obtained a ranking from (a) the graph-level anomaly detectors and (b) the tabular data anomaly detectors, we create a unique ranking for pairs of graphs and metadata, by using: (i) a BFS-style aggregation that first sorts the results of each stage in descending order of their anomaly score, and then selects the next object that has the highest anomaly score by visiting the lists in a BFS fashion [LK05]; and (ii) the Inverse Rank (IR) aggregation method, in which we score each sample by $\frac{1}{r_a} + \frac{1}{r_b}$, where r_a is the rank by the graph-level anomaly detector and r_b by the tabular data outlier detector.

Overall we construct 6 baselines based on (1)–(3) × (i)–(ii).

Labeled Anomalies. Our datasets do not come with any ground truth anomalies, therefore, we use the guidance of experts in the fields of accounting and human mobility to simulate anomalies that are typically present in these datasets and of interest in detecting them. We create two types of graph-level anomalies, as well two types of metadata-level anomalies.

1) Graph anomalies involve small perturbations in nodes and edges as follows.

- *Label change* (**GA1**): We change the label of a node to a randomly chosen new label. This injection corresponds to entry-error in an accounting dataset, or an unusual visit to a new POI in human mobility behavior.
- *Path injection* (**GA2**): We delete an edge between nodes u and v and rewire through an intermediary, creating a path u-z-v. This injection mimics money-laundering in finance, where funds are passed through an intermediate account instead of being transferred directly. For human mobility, this corresponds to an unusual stop.

2) Metadata anomalies perturb feature values and reflect different semantics in both domains.

For Accounting Datasets:

- *Unusual back-dating* (**MA1**): We pick a subset of entries with effective date

close to the entry date (up to 3 days before), and change the entry date randomly to one of $\{7, 14, 21\}$ days after the effective date. This corresponds to an unusual late entry date. We lack information about entry date in HW, hence our experiments do not use this type of anomaly for this dataset.
- *Combination of unrelated transactions* (**MA2**): We merge two unrelated transactions by creating a new one with a unique Journal ID. We set the metadata entries to a randomly chosen value from the two initial transactions. Note that this injection also modifies the graph structure into the representation of the merged journals.

For Human Mobility Dataset:
- *Unusual start time* (**MA3**): We change the start time of a trip to a very early (or late one). This corresponds to a trip occurring in an unusual time.
- *Unusual trip duration* (**MA4**): We change the duration of a trip to an unusually long one.

3) Potpourri anomalies involve a combination of graph and metadata level anomaly injections, where we pick one graph level anomaly and one metadata level anomaly from above and inject both to a sample.

We inject anomalies on 5% of the samples in each dataset, where we use half of the original dataset for training, and the remaining half with the injected anomalies for testing. Note that the labeled anomalies are used only for evaluation purposes and not during model training or model selection.

Model Selection. Anomaly detection is typically a fully unsupervised task, where we lack ground truth labels of which samples are anomalous. As a result, we do not have a validation set for hyperparameter tuning. For this reason, for each of the baseline methods, we consider a set of hyperparamater configurations, across which we report the average performance. This corresponds to the expected performance of each method if one were to select a configuration at random. For ADAMM we show that our proposed model selection criterion presented in 8.4.4 can consistently yield better results than what we would expect when picking hyperparameters at random (in the absence of any other guidance).

Hyperparameter (HP) Configurations: Detailed HP configurations can be found in Tab.8.3.

TABLE 8.3: Hyperparameter Configurations

Hyperparameter	**Configurations**
ADAMM	
# of centroids K	1, 2, 4
learning rate	1e-4, 1e-3
weight decay	1e-5, 1e-4
λ_1	0.1
λ_2	0, 0.1
WL Kernel	
WL iterations	1, 2, 4, 8, 16
G2V	
G2V iterations	1, 2, 4, 8, 16
DOMINANT	
α	0.4, 0.6

8.5.2 Detection Results

In evaluating proposed ADAMM, we conducted a series of experiments to answer the following questions:

- **Q1) Effectiveness:** How effective is ADAMM in detecting graph- and metadata-level anomalies, as compared to the two-stage baseline approaches?
- **Q2) Model Selection:** Can our proposed unsupervised model selection criterion for ADAMM select a model (i.e. hyperparameter configuration) that is better than random picking (i.e. avg. performance across config.s)?
- **Q3) Ablation:** How important are key components of ADAMM in the detection results?

A1: To answer the first question, we conduct extensive experiments using all four datasets and graph, metadata as well as potpourri anomalies. The results are presented for each method across all datasets and injection types in Table 8.2 based on the Area Under the Receiver Operator Characteristic Curve (AUROC), and in Table 8.4 based on the Area Under Precision-Recall Curve (AUPRC). We observe that ADAMM succeeds in detecting both graph-level and metadata-level anomalies effectively. It outperforms the baseline methods in 3 out of 4 datasets and across all injection types, where the baselines do not show consistent performance. ADAMM performs consistently well for all types of anomalies, whether they are graph, metadata-level, or even of mixed type. The only exception is the MobiNet dataset, where the nature of metadata information (multiple vectors for a single graph that have to be aggregated) poses significant challenges[1]

The superior performance of ADAMM over baselines is also validated using the Wilcoxon signed rank test. ADAMM not only has the lowest average rank among the competitors, but is also significantly better at p-value $p = 0.05$.

A2: The problem of model selection is an important one in unsupervised anomaly detection. The lack of labels and of a validation set makes it challenging to choose an effective model. For ADAMM we provide a validation criterion tightly connected to its loss function (recall Eq. Eq. (8.11)). In Figure 8.3 we see that the model ADAMM chooses based on its unsupervised criterion achieves consistently better performance than random picking that corresponds to the average performance across all configurations. This makes ADAMM not only able to spot graph and metadata level anomalies, but also robust against different hyperparameter choices.

A3: ADAMM exhibits three key building blocks; multi-edge representation learning, graph-metadata fusion, and a suitable anomaly detection loss. Accordingly, we perform an *ablation study* and design threevariants of ADAMM, each excluding the respective design component to demonstrate its added benefit.

V1. ADAMM **without Metadata Fusion**: Here we remove the metadata fusion component and instead we input only the graph-level embeddings $\mathbf{Z}_G$ to the membership estimation network. Our goal is to explore if the metadata component interferes with the graph-level component by having a negative influence on graph-level anomaly detection when only such anomalies are present.

[1]For baseline methods, we score all vectors separately and assign the maximum score as the anomaly score of the sample. This gives better results than taking the average as the latter dilutes the signal among multiple vectors.

TABLE 8.4: Anomaly Detection Results for all methods across all datasets based on AUPRC (Area Under Precision-Recall Curve). For baselines, average performance across hyperparameters along with the std. dev. is reported. ADAMM outputs a unique ranking based on a model selection criterion. Last row reports significance test results, where (**) and (***) denote that ADAMM is significantly better than baselines w.r.t. the Wilcoxon Signed Rank Test at $p = 0.05$ and $p = 0.01$, respectively.

Dataset	Anomaly Type	ADAMM	WL+BFS	WL+IR	G2V+BFS	G2V+IR	DOM.+BFS	DOM.+IR
SH	GA1	**0.939**	0.470 ± 0.01	0.461 ± 0.01	0.270 ± 0.01	0.327 ± 0.01	0.327 ± 0.02	0.320 ± 0.01
	GA2	**0.708**	0.214 ± 0.02	0.217 ± 0.02	0.269 ± 0.04	0.252 ± 0.01	0.252 ± 0.01	0.256 ± 0.01
	MA1	**0.280**	0.125 ± 0.01	0.126 ± 0.01	0.118 ± 0.01	0.125 ± 0.01	0.125 ± 0.01	0.126 ± 0.01
	MA2	**0.599**	0.125 ± 0.01	0.125 ± 0.01	0.120 ± 0.01	0.121 ± 0.01	0.121 ± 0.01	0.122 ± 0.01
	GA1 + MA1	**0.877**	0.458± 0.01	0.450 ± 0.01	0.255 ± 0.09	0.246 ± 0.09	0.080 ± 0.01	0.077 ± 0.01
	GA2 + MA1	**0.836**	0.224 ± 0.02	0.225 ± 0.04	0.286 ± 0.03	0.241 ± 0.01	0.097± 0.01	0.093 ± 0.01
KD	GA1	**0.553**	0.363 ± 0.02	0.347 ± 0.01	0.255 ± 0.09	0.246 ± 0.09	0.08 ± 0.01	0.080± 0.01
	GA2	**0.430**	0.284 ± 0.04	0.283 ± 0.04	0.234± 0.014	0.240 ± 0.01	0.102 ± 0.01	0.098 ± 0.01
	MA1	0.141	0.142 ± 0.01	0.136 ± 0.01	**0.148** ± 0.002	0.144 ± 0.01	0.106 ± 0.01	0.101 ± 0.01
	MA2	**0.342**	0.117 ± 0.001	0.104 ± 0.01	0.162 ± 0.01	0.164 ± 0.01	0.089 ± 0.01	0.086 ± 0.01
	GA1 + MA1	**0.677**	0.443 ± 0.01	0.424 ± 0.01	0.256 ± 0.12	0.247 ± 0.12	0.080 ± 0.01	0.077 ± 0.01
	GA2 + MA1	**0.351**	0.261 ± 0.04	0.261 ± 0.04	0.237 ± 0.01	0.240 ± 0.01	0.097 ± 0.01	0.094 ± 0.01
HW	GA1	**0.866**	0.446 ± 0.03	0.438 ± 0.03	0.455 ± 0.10	0.451 ± 0.01	0.314 ± 0.21	0.310 ± 0.21
	GA2	**0.941**	0.292 ± 0.01	0.280 ± 0.02	0.146 ± 0.03	0.144 ± 0.03	0.106 ± 0.04	0.105 ± 0.04
	MA2	**0.815**	0.165 ± 0.01	0.151 ± 0.01	0.175 ± 0.01	0.168 ± 0.01	0.114 ± 0.01	0.110 ± 0.01
MobiNet	GA1	0.050	0.100 ± 0.01	**0.102** ± 0.01	0.046 ± 0.01	0.046 ± 0.01	0.066± 0.01	0.065 ± 0.01
	GA2	0.043	0.047 ± 0.01	0.047 ± 0.03	0.054 ± 0.01	0.054 ± 0.01	**0.069** ± 0.01	**0.069** ± 0.01
	MA3	0.040	0.054 ± 0.01	0.055 ± 0.01	0.055 ± 0.01	0.057 ± 0.01	**0.076** ± 0.01	**0.076**± 0.01
	MA4	0.040	0.047 ± 0.01	0.047 ± 0.01	0.051 ± 0.01	0.051 ± 0.01	0.069 ± 0.01	**0.067** ± 0.01
	GA1 + MA3	0.052	0.165 ± 0.01	**0.166** ± 0.01	0.043 ± 0.01	0.044 ± 0.01	0.067 ± 0.01	0.067 ± 0.01
	GA1 + MA4	0.074	0.157 ± 0.01	**0.158** ± 0.01	0.046 ± 0.01	0.046 ± 0.01	0.069 ± 0.01	0.068 ± 0.01
	GA2 + MA3	0.052	0.044± 0.01	0.044 ± 0.01	0.054 ± 0.01	0.053 ± 0.01	**0.066** ± 0.01	0.065 ± 0.01
	GA2 + MA4	0.041	0.046 ± 0.01	0.046 ± 0.01	0.055 ± 0.01	0.056 ± 0.01	**0.066** ± 0.01	0.066 ± 0.01
Average AUPRC		0.443	0.208	0.204	0.165	0.163	0.131	0.130
Average Rank		2.13	3.91 (***)	4.26 (***)	4.73 (***)	3.95 (***)	4.52 (***)	4.48 (***)

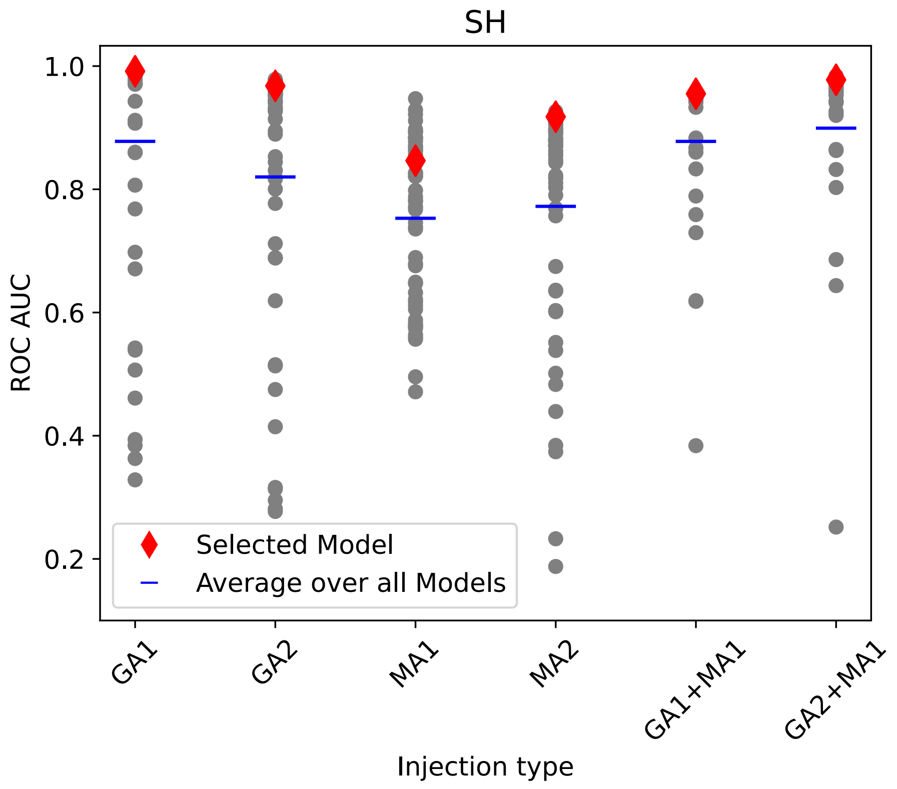


FIGURE 8.3: Model selection for ADAMM over all models with different hyper-parameter configurations. The model selected consistently performs better than random picking, i.e. average/expected performance over all models.

V2. ADAMM **without DeepSet**: In this version, we remove the DeepSet component that aims to learn a single representation of the attributed multi-edges. Instead,

TABLE 8.5: Ablation Study Results - Comparing ADAMM against its three variants: (i) ADAMM without Metadata Fusion, (ii) ADAMM without DeepSet & (iii) ADAMM with One-Class DeepSVDD loss (OCDL)

Dataset	Type	ADAMM		**w/o Metadata**		**w/o DeepSet**		**with OCDL**	
		AUROC	AUPRC	AUROC	AUPRC	AUROC	AUPRC	AUROC	AUPRC
SH	GA1	**0.992**	**0.938**	0.989	0.920	0.988	0.920	0.986	0.894
	GA2	**0.968**	**0.708**	0.961	0.622	0.965	0.758	0.930	0.489
	MA2	**0.918**	**0.598**	0.898	0.580	0.816	0.427	0.868	0.467
MobiNet	GA1	0.526	0.049	**0.779**	**0.179**	0.588	0.060	0.577	0.061
	GA2	0.491	0.043	**0.678**	**0.079**	0.475	0.042	0.472	0.045

we simply average the attributes over each multi-edge.

V3. ADAMM **with One-Class DeepSVDD loss**: Finally, we compare ADAMM and its loss function in Eq. Eq. (8.8) with a varint where we replace it with the loss introduced by the One-Class DeepSVDD[Ruf+18] method which, as we described in 8.4.3, maps all normal instances to a *single* hypersphere centered around a *fixed* centroid.

Results of the ablation study are given in Table 8.5 for SH (as a representative of the transaction datasets) as well as the MobiNet dataset. We see that ADAMM outperforms all of its variants on the *SH* dataset, demonstrating the importance of the various components in anomaly detection. The improvement is particularly noticeable for the MA2 type anomalies (merge of unrelated transactions), which is of mixed type (both metadata and graph). We note that ADAMM without metadata also performs well for graph-only anomalies of type GA1 and GA2. In fact, excluding metadata lifts the interference on MobiNet, leading to better detection.

8.5.3 Case Studies

Through quantitative experiments in 4.5.2 we showed that ADAMM can successfully spot expert-guided injected anomalies. To further validate the effectiveness of our method, we consider the original *SH* dataset that contains **no** injected anomalies. That is, we use the whole dataset of 39, 011 graphs with metadata for training and inspect their anomaly scores obtained by Eq. Eq. (8.7). As presented in Figure 8.4 (left), we see that ADAMM is able to highlight a small fraction of the samples as standing out from the majority.

As ADAMM is unique in handling complex directed graphs with attributes and multi-edges, we take a closer look at two example graphs as shown in Figure 8.4 (right). Self-loops are the common feature of these graphs: the first has one self-loop with a large dollar amount ($1.5M), while the second contains 38 self-loops in one graph. From an accounting domain perspective, self-loops represent transactions recorded by moving dollars *within* the same general ledger (GL) account. From a bookkeeping standpoint, these within-GL movements indicate the presence of misidentification of the correct sub-ledger account in the recording of a prior transaction. The self-loop in the current journal entry is then designed to correct such a misidentification at a later date. In the first graph A, a total of $1.5M was recorded in a sub-ledger incorrectly, necessitating the current self-loop transaction to correct the cumulative mistakes made earlier. The second graph B (with 38 self-loops) is even more pronounced in terms of the number of corrections involved as well as the presence of errors beyond the simple one illustrated in the first example.

Based on this transaction-level bookkeeping analysis, these two spotted transactions are indeed unusual and worthy of the auditor's attention to examine further.

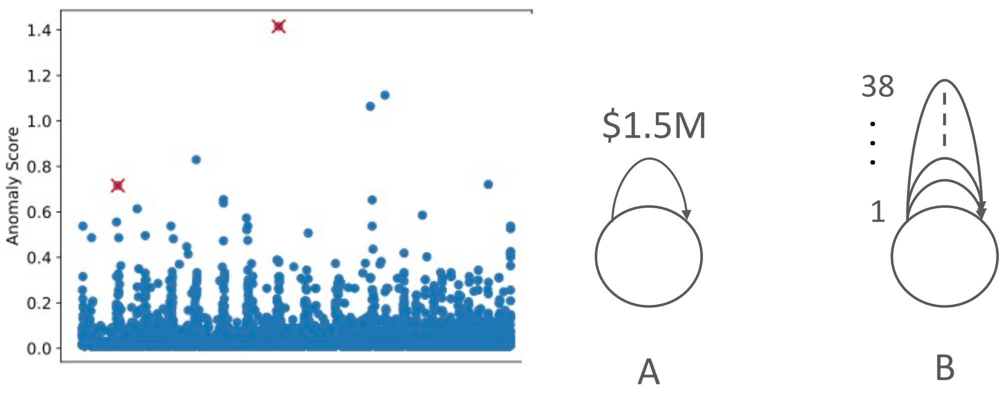


FIGURE 8.4: Analyzing detected accounting anomalies. (Left) Anomaly scores (vs journal ID) of all 39,011 entries in the SH dataset. (Right) Two example graphs, A and B, that are identified as anomalous by ADAMM in SH.

ADAMM's ability to spot these anomalies involving edge-attributes and multi-edges can be of assistance to the accounting/auditing practitioners.

8.6 Conclusion

In this work we addressed an anomaly detection problem that relates to one of the key challenges of big data mining, that is, data complexity. In particular, we considered a graph database consisting of node- and edge-attributed directed multi-graphs with associated metadata, and proposed a new multi-modal anomaly detection approach called ADAMM. In a unified neural network framework, ADAMM first captures a set representation of the multi-edges, learns a graph-level embedding, fuses the graph and metadata in a joint embedding space, on which it finally employs an unsupervised anomaly loss based on a multi-centered data distribution. To our knowledge, ADAMM is the first unified method that can tackle anomaly detection on complex data of this nature in an end-to-end fashion. Through extensive experiments on datasets from two real-world domains, namely accounting and urban mobility, we showed that ADAMM significantly outperforms all two-stage baselines that handle graphs and metadata separately. We open-source ADAMM's code for future research as well as practical use on possibly other real-world domains.

Part V

Conclusion

Chapter 9

Summary and Future Directions

9.1 Summary

This thesis has explored four distinct yet interconnected aspects of Graph Neural Networks (GNNs): the node-level fundamental challenge like oversmoothing, graph-level expressiveness towards universal function approximation, permutation-invariant generative models on graphs, and applications of GNNs such as graph-level anomaly detection. These four parts cover a wide range of topics within GNNs, spanning fundamental problems, theoretical understandings, model development, learning paradigms, and practical applications, with the shared goal of achieving effective graph representations that encode as much rich structural information within graphs as possible.

Recently, the concept of foundation model [Bom+21] attracts increasing attention across various domains. Large language models (LLM) like GPT-4 [Ope23], one type of the powerful foundation model in language domain, has revolutionized our daily life. The possibility of having powerful foundation models on graphs that are capable of providing representations for solving wide range of downstreaming graph tasks is still unknown and is a hot and active research topic. Many topics explored in this thesis are intended to pave the way for building a graph foundation model. To understand the prerequisites for a graph foundation model, let's examine the success of large language models (LLMs). Apart from the unique characteristics of the language domain, which contains dense human knowledge, the success of LLMs relies on at least three crucial components: the availability of a large amount of data, the power of the transformer architecture as a universal function approximator [Yun+20], and effective generative pretraining using the autoregressive next-token prediction task. Without the function approximation universality of transformer architecture, there will potentially be a ceiling for the model during parameter scaling, causing the scaling laws of LLMs to become less effective. Additionally, the simple yet highly effective next-token prediction as a generative loss captures the rich information within the abundant training data by modeling the entire joint distribution of sequences, making human-level intelligence possible. Based on these insights and hypotheses, I have focused on enhancing the expressiveness of graph neural networks (GNNs) toward becoming universal function approximators for functions on graphs to achieve more knowledge distillation and storage for emergence and homogenization capabilities. Furthermore, I have concentrated on the generative modeling of graphs to facilitate generative pretraining, such that the pretext tasks can condense rich knowledge within data. Of course, these explorations only cover limited pieces of graph foundation model. Given the intrinsic distinction between graphs and languages, there are numerous difficulties to achieve emergence and homogenization in graph domain. For example, different from language tasks, graphs are abstract data used for diverse applications and domains without context intersection, such that cross-domain homogenization is significantly challenging. Finding shared cornerstones and elements (for tokenization) is the key to achieve homogenization. Furthermore, killer applications are still lacking in the graph domain compared to the widely used chatbots in the language domain that are useful and approachable for everyone. This limits the resources and attention devoted to developing graph foundation models. While drug discovery has the potential to be a killer application, it remains a challenging and time-consuming field due to the difficulties of having sophisticated domain expert knowledge and lengthy process involved in testing and conducting clinical trials. Recently, [Liu+23] gives a comprehensive overview of the current progress of developing graph foundation models, highlighted many challenges and directions. This thesis contributes to the journey toward achieving a graph foundation model, and additional research will be conducted in the future to continue in this direction. I hope that in the near future, large-scale graph models will have groundbreaking applications in various domains.

9.2 Future directions

There are many important questions left in the area of GNNs, and many new problems are arising in the intersection between LLMs and GNNs. This section contains problems that interest me the most.

9.2.1 Left problems in GNNs

Many theories have been developed, such as the expressiveness hierarchy of GNNs [Zha+23a], approximation power of graph functions [GR22], ability of solving combinatorial NP-hard problems [GR22], generalization bound of GNNs [Ju+23] and extrapolation ability [Xu+21]. These theories are far from being fully developed.

Fine-grained expressivity measurement. Previous studies on expressiveness, based on the graph isomorphism test framework, equate universality with a complete isomorphism test [Che+19b]. However, the isomorphism test is not sufficiently fine-grained to characterize detailed function approximation abilities. A model's ability to distinguish two given graphs does not necessarily imply its capability to model any functions on these graphs. Therefore, other fine-grained measurements of expressiveness and approximation ability are essential to determine what functions can be approximated. Recently, Zhang et al. [Zha+24a] proposed a new expressivity measurement called homomorphism expressivity, which provides a comprehensive and practical tool for fine-grained and quantitative expressivity comparison in counting graphs under homomorphism. In the future, identifying the exact graph function family that can be approximated by a given graph network would be exciting. These new measurements can further drive the development of highly expressive and scalable graph networks, such as by improving higher-order graph networks.

Approximately equivariant model. A fundamental limitation of graph networks is their need to be permutation equivariant. This requirement imposes a strict constraint on the model architecture, which has both advantages and disadvantages. On the positive side, this constraint ensures that all approximated functions are naturally permutation equivariant, aligning with the default inductive bias and reducing the amount of data needed for learning. However, it also restricts the model's expressiveness, preventing it from being universal and limiting its ability to approximate certain functions, such as counting complex substructures. A well-known family of highly expressive equivariant models is the higher-order models based on the higher-order Weisfeiler-Lehman test [Zha+22b], which rely on exponential computational and memory complexity. This indicates that, in a fully equivariant setting, models trade expressivity for computational cost, making them impractical for real-world applications. Recently, AlphaFold3 [Abr+24] demonstrates that equivariance is not necessary in the structure prediction module, showing significant improvements over AlphaFold2, which is a fully equivariant model. Similarly, AlphaTensor employs an approximately equivariant architecture to discover faster tensor operations. By breaking equivariance, these models achieve greater simplicity and an expanded function space. The evil of equivariance is also discussed in Pard [ZDA24], which shows that without breaking the symmetry, general graph transformation from one graph to another modified graph is not possible. Future research should systematically study the influence of varying degrees of equivariance. Developing approximately equivariant models that expand the function space while preserving a certain degree of symmetry is a promising direction to balance between full equivariance and completely ignoring inherent symmetries.

Generative modeling improvement. Generative pretraining is a key cornerstone for foundation models, making generative modeling essential in graph representation learning. While Pard [ZDA24] has significantly improved the generation quality of small graphs, many challenges remain in this area. Pard combines autoregressive approach with diffusion model, with the categorical diffusion model being crucial due to the discrete nature of graph edges and node features. Pard relies on the improved discrete diffusion [Zha+24b] for better scalability and variational loss computation. However, supporting classifier guidance conditional generation [DN21] based on the pretrained unconditional model is nontrivial. Classifier guidance for continuous-state diffusion was initially developed by [SD+15], using the locality property of Gaussian distributions and Taylor expansion to make sampling tractable. This method does not work for categorical distributions, which lack the locality assumption and cannot be approximated using Taylor expansion as they are not differentiable. Some approaches transform the categorical distribution back to an approximate Gaussian distribution [Vig+23], but this can introduce significant approximation errors. Another challenge is to build the generative model for a single giant graph like recommendation system contains trillions of nodes and edges. Previous generative model focuses on modeling the distribution of a collection of small graphs, with each graph being i.i.d.. How to model the distribution of a single giant graph is unclear, with both computaton challenge and the non-i.i.d. nature of all elements in the graph.

Large-scale graph foundation model. Recently there are many achievements of building foundation model in domains like language and vision, and even multimodality, it is still not clear for the promise of graph foundation model. First, unlike the domain of language that contains dense and crucial human knowledge, graph data is diverse and more domain specific. Hence the effectiveness of pretrained large scale graph foundation model may only benefits a particular narrow domain. Second, the development of generative pre-training method is not mature yet, while diffusion models like [ZDA24] show a promising approach. Also, the expressiveness of current model is far from being universal function approximator, which can limit the ability of pretrained model. Recently, there is some explorations of training large scale pretrained model in specific domain like molecular [Ji+24], it is exciting to see a domain-specific foundation model being developed first. Notice that the famous AlphaFold3 [Abr+24] is also type of graph foundation model, with additional 3D geometric structure information available outside of the plain graph.

9.2.2 New problems in the intersection of LLMs and GNNs

Recent study [Wan+23] shows that while GPT-4 can understand basic graphs, it struggles with more complex graph-related problems, such as calculating graph properties.

This limitation highlights that current pre-trained models, despite their multimodal abilities, lack sufficient graph reasoning skills. Importantly, graph reasoning ability is crucial for human for solving complex tasks, and graph as a mathematical modeling tool is widely used in daily life to simplify real-world problems to derive optimal solutions like shortest-path trajectory or navigation with A* algorithm. Hence, improving LLMs with graph reasoning is a key direction for the next-generation LLMs like GPT-5, potentially enabling it to solve challenging tasks without extensive multi-step prompting.

There are many important directions that I have interest. 1) **Using Graphs to Enhance Data Generation Pipelines**. With the scaling law of LLMs, data quality and size become the essential part of improving model abilities. For a vision-language pretrained foundation model, the data creation within language and vision are already widely explored with diminishing room for discovering new augmentation strategies. Therefore, it would be intriguing to generate graph datasets with various graph reasoning tasks, forming (graph, reasoning task) pairs to augment the data pipeline. 2) **Vision-language-graph multi-modal pretraining.** Graphs can be viewed as a unified compression language for both text and vision. For example, a paragraph or an essay can be condensed into a graph with nodes representing entities and edges representing their relationships, effectively creating a compact knowledge graph. Similarly, images can be summarized as scene graphs. Developing (image, graph) and (text, graph) pairs could be a significant step towards a multimodal foundation model that incorporates graphs. This would also provide users with simplified control over text and image generation by editing the underlying graph. Moreover, more complex reasoning tasks in language and image domains can be facilitated with the aid of graphs in the task generation pipeline. My work on generative graph models will be crucial for designing training objectives and adapting transformers to support graph-involved multimodal generative pretraining. 3) **Retrieval augmented generation (RAG) with relational content.** RAG, which allows LLMs to utilize external content and databases, is a valuable direction. When working with relational external databases, such as knowledge graphs and relational tables, developing a vector database that encodes graph structural information for content ranking is important. Equipping LLMs with the ability to directly process graphs within RAG is also a significant development. 4) **LLM Planning with Graph-Based Decomposition.** Research shows that LLMs often struggle with complex tasks but can improve significantly through multi-step reasoning approaches like chain-of-thought or graph-of-thought. Implementing text-to-graph based conditional generative models for task decomposition could enhance LLMs' abilities during inference. Additionally, this approach can involve multiple LLM agents working collaboratively to solve complex tasks.

Bibliography

[Abb+21] Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. “The Surprising Power of Graph Neural Networks with Random Node Initialization”. In: *Proceedings of the Thirtieth International Joint Conference on Artifical Intelligence (IJCAI)*. 2021.

[Abr+24] Josh Abramson et al. “Accurate structure prediction of biomolecular interactions with AlphaFold 3”. In: *Nature* (2024), pp. 1–3.

[AEH+19] Sami Abu-El-Haija et al. “Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing”. In: *international conference on machine learning*. PMLR. 2019, pp. 21–29.

[Agg17] Charu C Aggarwal. *An introduction to outlier analysis*. Springer, 2017.

[Agg15] Charu C Aggarwal. “Outlier analysis”. In: *Data mining*. Springer. 2015, pp. 237–263.

[Ako21] Leman Akoglu. “Anomaly Mining: Past, Present and Future”. In: *International Conference on Information & Knowledge Management*. 2021, pp. 1–2.

[ATK15] Leman Akoglu, Hanghang Tong, and Danai Koutra. “Graph based anomaly detection and description: a survey”. In: *Data mining and knowledge discovery* 29 (2015), pp. 626–688.

[AY21] Uri Alon and Eran Yahav. “On the Bottleneck of Graph Neural Networks and its Practical Implications”. In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=i80OPhOCVH2.

[AH18] Namrata Anand and Possun Huang. “Generative modeling for protein structures”. In: *Advances in neural information processing systems* 31 (2018).

[And12] William J Anderson. *Continuous-time Markov chains: An applications-oriented approach*. Springer Science & Business Media, 2012.

[Arv+20] Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. “On weisfeiler-leman invariance: subgraph counts and related graph properties”. In: *Journal of Computer and System Sciences* 113 (2020), pp. 42–59.

[Aus+21] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. “Structured denoising diffusion models in discrete state-spaces”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 17981–17993.

[AL21] Waiss Azizian and Marc Lelarge. “Expressive Power of Invariant and Equivariant Graph Neural Networks”. In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=lxHgXYN4bwl.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: *CoRR* abs/1607.06450 (2016).

[BES80] László Babai, Paul Erdos, and Stanley M Selkow. “Random graph isomorphism”. In: *SIaM Journal on computing* 9.3 (1980), pp. 628–635.

[BG98] Suresh Balakrishnama and Aravind Ganapathiraju. “Linear discriminant analysis-a brief tutorial”. In: *Institute for Signal and information Processing* 18.1998 (1998), pp. 1–8.

[Bal04] R Balakrishnan. “The energy of a graph”. In: *Linear Algebra and its Applications* 387 (2004), pp. 287–295.

[Bal+21] Muhammet Balcilar, Pierre Héroux, Benoit Gaüzère, Pascal Vasseur, Sébastien Adam, and Paul Honeine. “Breaking the Limits of Message Passing Graph Neural Networks”. In: *Proceedings of the 38th International Conference on Machine Learning (ICML)*. 2021.

[Bar+21] Pablo Barceló, Floris Geerts, Juan Reutter, and Maksimilian Ryschkov. “Graph Neural Networks with Local Graph Parameters”. In: (2021).

[Bea+21] Dominique Beani, Saro Passaro, Vincent Létourneau, Will Hamilton, Gabriele Corso, and Pietro Lió. “Directional Graph Networks”. In: *Proceedings of the 38th International Conference on Machine Learning (ICML)*. 2021, pp. 748–758.

[BCV13] Y. Bengio, A. Courville, and P. Vincent. “Representation Learning: A Review and New Perspectives”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (Aug. 2013). Zu bearbeitendes Review, pp. 1798–1828. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2013.50. URL: http://ieeexplore.ieee.org/document/6472238/.

[BLA22] Dimitris Berberidis, Pierre J Liang, and Leman Akoglu. “Summarizing Labeled Multi-graphs”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2022, pp. 53–68.

[Bev+22] Beatrice Bevilacqua et al. “Equivariant Subgraph Aggregation Networks”. In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=dFbKQaRk15w.

[Bod+21a] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yu Guang Wang, Pietro Liò, Guido Montúfar, and Michael Bronstein. “Weisfeiler and Lehman Go Cellular: CW Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021.

[Bod+21b] Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montufar, Pietro Lió, and Michael Bronstein. “Weisfeiler and Lehman Go Topological: Message Passing Simplicial Networks”. In: *Proceedings of the 38th International Conference on Machine Learning (ICML)*. 2021, pp. 1026–1037.

[Bom+21] Rishi Bommasani et al. “On the opportunities and risks of foundation models”. In: *arXiv preprint arXiv:2108.07258* (2021).

[Bon+20] Angela Bonifati, Irena Holubová, Arnau Prat-Pérez, and Sherif Sakr. “Graph generators: State of the art and open challenges”. In: *ACM computing sunrveys (CsunR)* 53.2 (2020), pp. 1–30.

[Bor+05] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. “Protein function prediction via graph kernels”. In: *Bioinformatics* 21.suppl_1 (2005), pp. i47–i56.

[Bou+20] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. “Improving graph neural network expressivity via subgraph isomorphism counting”. In: *arXiv preprint arXiv:2006.09252* (2020).

[Bre+00] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. “LOF: identifying density-based local outliers”. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 93–104.

[Bro+20] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[Cai+23] Chen Cai, Truong Son Hy, Rose Yu, and Yusu Wang. “On the Connection Between MPNN and Graph Transformer”. In: *International Conference on Machine Learning* (2023).

[CFI92] Jin-Yi Cai, Martin Fürer, and Neil Immerman. “An optimal lower bound on the number of variables for graph identification”. In: *Combinatorica* 12.4 (1992), pp. 389–410.

[Cam+22] Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. “A continuous time framework for discrete denoising models”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 28266–28279.

[Cam+16] Guilherme O Campos et al. “On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study”. In: *Data mining and knowledge discovery* 30.4 (2016), pp. 891–927.

[CC19] Raghavendra Chalapathy and Sanjay Chawla. “Deep learning for anomaly detection: A survey”. In: *arXiv preprint arXiv:1901.03407* (2019).

[Cha+15] Tsung-Han Chan, Kui Jia, Shenghua Gao, Jiwen Lu, Zinan Zeng, and Yi Ma. “PCANet: A simple deep learning baseline for image classification?” In: *IEEE Transactions on Image Processing* 24.12 (2015), pp. 5017–5032.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly detection: A survey”. In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[Cha+22] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. *MaskGIT: Masked Generative Image Transformer*. 2022. arXiv: 2202.04200 [cs.CV].

[CSZ06] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. 2006.

[Che+20a] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. “Measuring and relieving the over-smoothing problem for graph neural networks from the topological view”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 2020, pp. 3438–3445.

[CCB20] Lei Chen, Zhengdao Chen, and Joan Bruna. “On graph neural networks versus graph-augmented mlps”. In: *arXiv preprint arXiv:2010.15116* (2020).

[Che23] Ting Chen. “On the importance of noise scheduling for diffusion models”. In: *arXiv preprint arXiv:2301.10972* (2023).

[Che+21] Xiaohui Chen, Xu Han, Jiajing Hu, Francisco Ruiz, and Liping Liu. “Order Matters: Probabilistic Modeling of Node Sequence for Graph Generation”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 1630–1639.

[Che+20b] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. “Can graph neural networks count substructures?” In: *arXiv preprint arXiv:2002.04025* (2020).

[Che+19a] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. “On the equivalence between graph isomorphism testing and function approximation with GNNs”. In: *Advances in Neural Information Processing Systems*. 2019.

[Che+19b] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. “On the equivalence between graph isomorphism testing and function approximation with GNNs”. In: *Advances in neural information processing systems* 32 (2019).

[Chi+19] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. “Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 257–266.

[Cho+21] Kukjin Choi, Jihun Yi, Changhwa Park, and Sungroh Yoon. “Deep learning for anomaly detection in time-series data: review, analysis, and guidelines”. In: *IEEE Access* 9 (2021), pp. 120043–120065.

[Cor+20] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. “Principal Neighbourhood Aggregation for Graph Nets”. In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 13260–13271.

[CMR21] Leonardo Cotta, Christopher Morris, and Bruno Ribeiro. “Reconstruction for powerful graph representations”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[DDK12] Abhimanyu Das, Anirban Dasgupta, and Ravi Kumar. “Selecting diverse features via spectral regularization”. In: *NeurIPS* 25 (2012).

[Das+20] George Dasoulas, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux. “Coloring Graph Neural Networks for Node Disambiguation”. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. International Joint Conferences on Artificial Intelligence Organization, 2020, pp. 2126–2132.

[De+22] sunparna De, Maria Bermudez-Edo, Honghui Xu, and Zhipeng Cai. “Deep generative models in the industrial internet of things: a sunrvey”. In: *IEEE Transactions on Industrial Informatics* 18.9 (2022), pp. 5728–5737.

[DCK18] Nicola De Cao and Thomas Kipf. “MolGAN: An implicit generative model for small molecular graphs”. In: *arXiv preprint arXiv:1805.11973* (2018).

[DCO20] Ahmet Demirkaya, Jiasi Chen, and Samet Oymak. “Exploring the role of loss functions in multiclass classification”. In: *2020 54th annual conference on information sciences and systems (ciss)*. IEEE. 2020, pp. 1–5.

[Den+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[DN21] Prafulla Dhariwal and Alexander Nichol. “Diffusion models beat gans on image synthesis”. In: *Advances in neural information processing systems* 34 (2021), pp. 8780–8794.

[Din+19] Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. “Deep anomaly detection on attributed networks”. In: *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM. 2019, pp. 594–602.

[DD03] Paul D Dobson and Andrew J Doig. “Distinguishing enzyme structures from non-enzymes without alignments”. In: *Journal of molecular biology* 330.4 (2003), pp. 771–783.

[Don+17] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. *MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment*. 2017. arXiv: 1709.06298 [eess.AS].

[DDL22] Mohammed Haroon Dupty, Yanfei Dong, and Wee Sun Lee. “PF-GNN: Differentiable particle filtering based approximation of universal graph representations”. In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=oh4TirnfSem.

[Duv+15] David K Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. “Convolutional Networks on Graphs for Learning Molecular Fingerprints”. In: *Advances in neural information processing systems*. 2015.

[Dwi+20] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. “Benchmarking graph neural networks”. In: *arXiv preprint arXiv:2003.00982* (2020).

[EHC09] William Eberle, Lawrence Holder, and Diane Cook. “Identifying threats using graph-based anomaly detection”. In: *Mach. Learn. in Cyber Trust*. 2009.

[Elt+19] Daniel C Elton, Zois Boukouvalas, Mark D Fuge, and Peter W Chung. “Deep learning for molecular design—a review of the state of the art”. In: *Molecular Systems Design & Engineering* 4.4 (2019), pp. 828–849.

[Emm+15] Andrew Emmott, Shubhomoy Das, Thomas Dietterich, Alan Fern, and Weng-Keen Wong. “A meta-analysis of the anomaly detection problem”. In: *arXiv preprint arXiv:1503.01158* (2015).

[Emm+13] Andrew F Emmott, Shubhomoy Das, Thomas Dietterich, Alan Fern, and Weng-Keen Wong. “Systematic construction of anomaly detection benchmarks from real data”. In: *Proceedings of the ACM SIGKDD workshop on outlier detection and description*. 2013, pp. 16–21.

[Err+20] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. “A Fair Comparison of Graph Neural Networks for Graph Classification”. In: *International Conference on Learning Representations (ICLR)*. 2020. URL: `https://openreview.net/forum?id=HygDF6NFPB`.

[ERO20] Patrick Esser, Robin Rombach, and Björn Ommer. *Taming Transformers for High-Resolution Image Synthesis*. 2020. arXiv: `2012.09841 [cs.CV]`.

[Fan+19] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. “Graph neural networks for social recommendation”. In: *The World Wide Web Conference*. 2019, pp. 417–426.

[Fan+23] Wenqi Fan et al. “Generative diffusion models on graphs: Methods and applications”. In: *arXiv preprint arXiv:2302.02591* (2023).

[Fär+10] Ines Färber et al. “On using class-labels in evaluation of clusterings”. In: *MultiClust: 1st international workshop on discovering, summarizing and using multiple clusterings held in conjunction with KDD*. 2010, p. 1.

[FYW20] M. Fey, J. G. Yuen, and F. Weichert. “Hierarchical Inter-Message Passing for Learning on Molecular Graphs”. In: *ICML Graph Representation Learning and Beyond (GRL+) Workhop*. 2020.

[Fey19] Matthias Fey. “Just Jump: Dynamic Neighborhood Aggregation in Graph Neural Networks.” In: *CoRR* abs/1904.04849 (2019).

[FL19] Matthias Fey and Jan E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.

[For96] Scott Fortin. “The graph isomorphism problem”. In: (1996).

[Fra+22] Fabrizio Frasca, Beatrice Bevilacqua, Michael M Bronstein, and Haggai Maron. “Understanding and extending subgraph gnns by rethinking their symmetries”. In: *arXiv preprint arXiv:2206.11140* (2022).

[Gal+08] Brian Gallagher, Hanghang Tong, Tina Eliassi-Rad, and Christos Faloutsos. “Using ghost edges for classification in sparsely labeled networks.” In: *International Conference on Knowledge Discovery & Data Mining*. ACM, 2008, pp. 256–264. URL: `http://dblp.uni-trier.de/db/conf/kdd/kdd2008.html#GallagherTEF08`.

[GJJ20] Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. “Generalization and representational limits of graph neural networks”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3419–3430.

[Gee20] Floris Geerts. “The expressive power of kth-order invariant graph networks”. In: *arXiv preprint arXiv:2007.12035* (2020).

[GR22] Floris Geerts and Juan L Reutter. “Expressiveness and Approximation Properties of Graph Neural Networks”. In: *International Conference on Learning Representations*. 2022. URL: `https://openreview.net/forum?id=wIzUeM3TAU`.

[GK86] Paul Geladi and Bruce R Kowalski. “Partial least-squares regression: a tutorial”. In: *Analytica chimica acta* 185 (1986), pp. 1–17.

[Gil01] Daniel T Gillespie. “Approximate accelerated stochastic simulation of chemically reacting systems”. In: *The Journal of chemical physics* 115.4 (2001), pp. 1716–1733.

[Gil77] Daniel T Gillespie. “Exact stochastic simulation of coupled chemical reactions”. In: *The journal of physical chemistry* 81.25 (1977), pp. 2340–2361.

[Gil+17] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. “Neural message passing for quantum chemistry”. In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[GIM99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. “Similarity Search in High Dimensions via Hashing”. In: *The VLDB Journal*. 1999, pp. 518–529. URL: http://citeseer.ist.psu.edu/203242.html.

[GB10] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*. 2010, pp. 249–256.

[GV89] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1989.

[GM+11] Laura Gonzalez-Malerva et al. “High-throughput ectopic expression screen for tamoxifen resistance identifies an atypical kinase that blocks autophagy”. In: *Proceedings of the National Academy of Sciences* 108.5 (2011), pp. 2058–2063.

[GJR20] Nikhil Goyal, Harsh Vardhan Jain, and Sayan Ranu. “Graphgen: A scalable approach to domain-agnostic labeled graph generation”. In: *Proceedings of The Web Conference 2020*. 2020, pp. 1253–1263.

[Gro21] Martin Grohe. “The logic of graph neural networks”. In: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE. 2021, pp. 1–17.

[GO15] Martin Grohe and Martin Otto. “Pebble games and linear equations”. In: *The Journal of Symbolic Logic* 80.3 (2015), pp. 797–844.

[GL16] Aditya Grover and Jure Leskovec. “node2vec: Scalable feature learning for networks”. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864.

[Gup+13] Manish Gupta, Jing Gao, Charu C Aggarwal, and Jiawei Han. “Outlier detection for temporal data: A survey”. In: *IEEE Transactions on Knowledge and data Engineering* 26.9 (2013), pp. 2250–2267.

[GLZ09] Ivan Gutman, Xueliang Li, and Jianbin Zhang. “Graph energy”. In: *Analysis of Complex Networks: From Biology to Linguistics* (2009), pp. 145–174.

[HYL17] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive representation learning on large graphs”. In: *Advances in neural information processing systems*. 2017, pp. 1024–1034.

[Har+16] Christopher R Harshaw, Robert A Bridges, Michael D Iannacone, Joel W Reed, and John R Goodall. “Graphprints: Towards a graph analytic method for network anomaly detection”. In: *Proceedings of the 11th Annual Cyber and Information Security Research Conference*. 2016, pp. 1–4.

[Haw80] Douglas M Hawkins. *Identification of outliers*. Vol. 11. Springer, 1980.

[Haw04] Douglas M Hawkins. “The problem of overfitting”. In: *Journal of chemical information and computer sciences* 44.1 (2004), pp. 1–12.

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2016, pp. 770–778.

[He+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international Conference on Computer Vision*. 2015, pp. 1026–1034.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6840–6851.

[Ho+22] Jonathan Ho et al. “Imagen video: High definition video generation with diffusion models”. In: *arXiv preprint arXiv:2210.02303* (2022).

[Hoo+22a] Emiel Hoogeboom, Alexey A. Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. “Autoregressive Diffusion Models”. In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=Lm8T39vLDTE.

[Hoo+21] Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. “Argmax flows and multinomial diffusion: Learning categorical distributions”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 12454–12465.

[Hoo+22b] Emiel Hoogeboom, Víctor Garcia Satorras, Clément Vignac, and Max Welling. “Equivariant Diffusion for Molecule Generation in 3D”. In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 8867–8887. URL: https://proceedings.mlr.press/v162/hoogeboom22a.html.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[Hu+20a] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. “Strategies for Pre-training Graph Neural Networks”. In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=HJlWWJSFDH.

[Hu+20b] Weihua Hu et al. “Open Graph Benchmark: Datasets for Machine Learning on Graphs”. In: *arXiv preprint arXiv:2005.00687* (2020).

[HZ20] Kexin Huang and Marinka Zitnik. “Graph meta learning via local subgraphs”. In: *Advances in Neural Information Processing Systems* 33 (2020).

[Hyv07] Aapo Hyvärinen. “Some extensions of score matching”. In: *Computational statistics & data analysis* 51.5 (2007), pp. 2499–2512.

[IS15] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” In: *CoRR* abs/1502.03167 (2015).

[Irw+12] John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. “ZINC: a free tool to discover chemistry for biology”. In: *Journal of chemical information and modeling* 52.7 (2012), pp. 1757–1768.

[Ji+24] Xiaohong Ji, Wang Zhen, Zhifeng Gao, Hang Zheng, Linfeng Zhang, Guolin Ke, et al. “Uni-Mol2: Exploring Molecular Pretraining Model at Scale”. In: *arXiv preprint arXiv:2406.14969* (2024).

[Jia+13] Bo Jiang, Chris Ding, Bio Luo, and Jin Tang. “Graph-Laplacian PCA: Closed-form solution and robustness”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 3492–3498.

[JLH22] Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. “Score-based generative modeling of graphs via the system of stochastic differential equations”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 10362–10383.

[Ju+23] Haotian Ju, Dongyue Li, Aneesh Sharma, and Hongyang R Zhang. “Generalization in graph neural networks: Improved pac-bayesian bounds on graph diffusion”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2023, pp. 6314–6341.

[KC18] Seokho Kang and Kyunghyun Cho. “Conditional molecular design with deep generative models”. In: *Journal of chemical information and modeling* 59.1 (2018), pp. 43–52.

[Kan+20] Zhao Kang, Guoxin Shi, Shudong Huang, Wenyu Chen, Xiaorong Pu, Joey Tianyi Zhou, and Zenglin Xu. “Multi-graph fusion for multi-view spectral clustering”. In: *Knowledge-Based Systems* 189 (2020), p. 105102.

[Kar+22] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. “Elucidating the design space of diffusion-based generative models”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 26565–26577.

[KMB05] Jeroen Kazius, Ross McGuire, and Roberta Bursi. “Derivation and validation of toxicophores for mutagenicity prediction”. In: *Journal of medicinal chemistry* 48.1 (2005), pp. 312–320.

[KP19] Nicolas Keriven and Gabriel Peyré. “Universal invariant and equivariant graph neural networks”. In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 7092–7101.

[Kim+22] Jinwoo Kim, Tien Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. “Pure Transformers are Powerful Graph Learners”. In: *arXiv* abs/2207.02505 (2022). URL: https://arxiv.org/abs/2207.02505.

[KB14] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[KW17] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *International Conference on Learning Representations (ICLR)*. 2017.

[Kle00] Jon Kleinberg. “The small-world phenomenon: An algorithmic perspective”. In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. 2000, pp. 163–170.

[KBG19] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. "Predict then Propagate: Graph Neural Networks meet Personalized PageRank". In: *International Conference on Learning Representations (ICLR)*. 2019.

[Kol31] Andrei Kolmogoroff. "Über die analytischen Methoden in der Wahrscheinlichkeitsrechnung". In: *Mathematische Annalen* 104 (1931), pp. 415–458.

[Kon+23] Lingkai Kong, Jiaming Cui, Haotian Sun, Yuchen Zhuang, B Aditya Prakash, and Chao Zhang. "Autoregressive diffusion model for graph generation". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 17391–17408.

[Krä+16] Philipp Krähenbühl, Carl Doersch, Jeff Donahue, and Trevor Darrell. "Data-dependent initializations of convolutional neural networks". In: *International Conference on Learning Representations (ICLR)*. 2016.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems*. 2012, pp. 1097–1105.

[LK05] Aleksandar Lazarevic and Vipin Kumar. "Feature bagging for outlier detection". In: *ACM SIGKDD*. 2005, pp. 157–166.

[Lee+19] John Boaz Lee, Ryan A Rossi, Xiangnan Kong, Sungchul Kim, Eunyee Koh, and Anup Rao. "Graph convolutional networks with motif-based attention". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019, pp. 499–508.

[Lee+21] Meng-Chieh Lee, Hung T Nguyen, Dimitris Berberidis, Vincent S Tseng, and Leman Akoglu. "GAWD: graph anomaly detection in weighted directed graph databases". In: *IEEE/ACM ASONAM*. 2021, pp. 143–150.

[Li+19a] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. "Can GCNs Go as Deep as CNNs?" In: *CoRR* abs/1904.03751 (2019).

[Li+20] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. "Distance Encoding: Design Provably More Powerful Neural Networks for Graph Representation Learning". In: (2020).

[LHW18] Qimai Li, Zhichao Han, and Xiao-Ming Wu. "Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning". In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. 2018, pp. 3538–3545.

[Li+19b] Qimai Li, Xiao-Ming Wu, Han Liu, Xiaotong Zhang, and Zhichao Guan. "Label efficient semi-supervised learning via graph filtering". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9582–9591.

[Li+22] Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. "Diffusion-lm improves controllable text generation". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 4328–4343.

[LPL21] Yibo Li, Jianfeng Pei, and Luhua Lai. "Structure-based de novo drug design using 3D deep generative models". In: *Chemical science* 12.41 (2021), pp. 13664–13675.

[LZL18] Yibo Li, Liangren Zhang, and Zhenming Liu. “Multi-objective de novo drug design with conditional graph generative model”. In: *Journal of Cheminformatics* 10 (2018), pp. 1–24.

[Li+18] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. “Learning deep generative models of graphs”. In: *arXiv preprint arXiv:1803.03324* (2018).

[Lia23] Pierre Jinghong Liang. “Bookkeeping Graphs: Computational Theory and Applications”. In: *Foundations and Trends® in Accounting* 17.2 (2023), pp. 77–172.

[Lia+19] Renjie Liao et al. “Efficient graph generation with graph recurrent attention networks”. In: *Advances in neural information processing systems* 32 (2019).

[Liu+05] Chao Liu, Xifeng Yan, Hwanjo Yu, Jiawei Han, and Philip S Yu. “Mining behavior graphs for “backtrace” of noncrashing bugs”. In: *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM. 2005, pp. 286–297.

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation forest”. In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, pp. 413–422.

[Liu+23] Jiawei Liu et al. “Towards graph foundation models: A survey and beyond”. In: *arXiv preprint arXiv:2310.11829* (2023).

[Lou20a] Andreas Loukas. “How hard is to distinguish graphs with graph neural networks?” In: *Advances in Neural Information Processing Systems*. 2020.

[Lou20b] Andreas Loukas. “What graph neural networks cannot learn: depth vs width”. In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=B1l2bp4YwS.

[LTT17] Cheng-Yaw Low, Andrew Beng-Jin Teoh, and Kar-Ann Toh. “Stacking PCANet+: An overly simplified convnets baseline for face recognition”. In: *IEEE Signal Processing Letters* 24.11 (2017), pp. 1581–1585.

[Luo+22] Xuexiong Luo et al. “Deep graph level anomaly detection with contrastive learning”. In: *Scientific Reports* 12.1 (2022), p. 19867.

[Lyu09] Siwei Lyu. “Interpretation and generalization of score matching”. In: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. 2009, pp. 359–366.

[Ma+23] Liheng Ma et al. “Graph Inductive Biases in Transformers without Message Passing”. In: *Proc. Int. Conf. Mach. Learn.* 2023.

[Ma+22] Rongrong Ma, Guansong Pang, Ling Chen, and Anton van den Hengel. “Deep graph-level anomaly detection by glocal knowledge distillation”. In: *WSDM*. ACM, 2022, pp. 704–714.

[Ma+21] Xiaoxiao Ma et al. “A comprehensive survey on graph anomaly detection with deep learning”. In: *IEEE Trans. on Knowledge and Data Engineering* (2021).

[Ma+20] Yao Ma, Xiaorui Liu, Tong Zhao, Yozen Liu, Jiliang Tang, and Neil Shah. “A unified view on graph neural networks as graph signal denoising”. In: *arXiv preprint arXiv:2010.01777* (2020).

[MY01] Larry M Manevitz and Malik Yousef. “One-class SVMs for document classification”. In: *Journal of machine Learning research* 2.Dec (2001), pp. 139–154.

[MMA16] Emaad A. Manzoor, Sadegh M. Milajerdi, and Leman Akoglu. “Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs.” In: *KDD*. ACM, 2016, pp. 1035–1044. URL: http://dblp.uni-trier.de/db/conf/kdd/kdd2016.html#ManzoorMA16.

[Mar+20] Amir Markovitz, Gilad Sharir, Itamar Friedman, Lihi Zelnik-Manor, and Shai Avidan. “Graph Embedded Pose Clustering for Anomaly Detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10539–10547.

[Mar+19a] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. “Provably powerful graph networks”. In: *Advances in neural information processing systems* 32 (2019).

[Mar+19b] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. “Invariant and Equivariant Graph Networks”. In: *International Conference on Learning Representations*. 2019.

[Mar+19c] Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. “On the universality of invariant networks”. In: *International conference on machine learning*. PMLR. 2019, pp. 4363–4371.

[MGF11] Koji Maruhashi, Fan Guo, and Christos Faloutsos. “Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis”. In: *ASONAM*. IEEE/ACM. 2011, pp. 203–210.

[Mik+13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[MM15] Dmytro Mishkin and Jiri Matas. “All you need is a good init”. In: *arXiv preprint arXiv:1511.06422* (2015).

[Mon+17] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. “Geometric deep learning on graphs and manifolds using mixture model cnns”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5115–5124.

[Mon+19] Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. “Fake news detection on social media using geometric deep learning”. In: *arXiv preprint arXiv:1902.06673* (2019).

[MOB18] Federico Monti, Karl Otness, and Michael M Bronstein. “Motifnet: a motif-based graph convolutional network for directed graphs”. In: *2018 IEEE Data Science Workshop (DSW)*. IEEE. 2018, pp. 225–228.

[MKM17] Christopher Morris, Kristian Kersting, and Petra Mutzel. “Glocalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs”. In: *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2017, pp. 327–336.

[Mor+20] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. “TUDataset: A collection of benchmark datasets for learning with graphs”. In: *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*. 2020. arXiv: 2007.08663. URL: www.graphlearning.io.

[Mor+22] Christopher Morris, Gaurav Rattan, Sandra Kiefer, and Siamak Ravanbakhsh. “SpeqNets: Sparsity-aware Permutation-equivariant Graph Networks”. In: *arXiv preprint arXiv:2203.13913* (2022).

[MRM20] Christopher Morris, Gaurav Rattan, and Petra Mutzel. “Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings”. In: *Advances in Neural Information Processing Systems* 33 (2020).

[Mor+19] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. “Weisfeiler and leman go neural: Higher-order graph neural networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2019.

[Mur+19] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. “Relational pooling for graph representations”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 4663–4673.

[Nar+16] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. “subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs”. In: *arXiv preprint arXiv:1606.08928* (2016).

[Nar+17] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. “graph2vec: Learning distributed representations of graphs”. In: *arXiv preprint arXiv:1707.05005* (2017).

[Nea+18] Brady Neal, Sarthak Mittal, Aristide Baratin, Vinayak Tantia, Matthew Scicluna, Simon Lacoste-Julien, and Ioannis Mitliagkas. “A modern take on the bias-variance tradeoff in neural networks”. In: *arXiv preprint arXiv:1810.08591* (2018).

[Neu+16] Marion Neumann, Roman Garnett, Christian Bauckhage, and Kristian Kersting. “Propagation kernels: efficient graph kernels from propagated information”. In: *Machine Learning* 102.2 (2016), pp. 209–245.

[NLA20] Hung T. Nguyen, Pierre J. Liang, and Leman Akoglu. *Anomaly Detection in Large Labeled Multi-Graph Databases*. 2020. arXiv: 2010.03600 [cs.DB].

[NLA23] Hung T Nguyen, Pierre J Liang, and Leman Akoglu. “Detecting anomalous graphs in labeled multi-graph databases”. In: *ACM Transactions on Knowledge Discovery from Data* 17.2 (2023), pp. 1–25.

[NAK16] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. “Learning convolutional neural networks for graphs”. In: *International conference on machine learning*. PMLR. 2016, pp. 2014–2023.

[NDV20] Giannis Nikolentzos, George Dasoulas, and Michalis Vazirgiannis. “k-hop graph neural networks”. In: *Neural Networks* 130 (2020), pp. 195–205.

[Niu+20] Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. “Permutation invariant graph generation via score-based generative modeling”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 4474–4484.

[NC03] Caleb C. Noble and Diane J. Cook. “Graph-based anomaly detection.” In: *KDD*. ACM, 2003, pp. 631–636.

[NM19] Hoang NT and Takanori Maehara. “Revisiting Graph Neural Networks: All We Have is Low-Pass Filters”. In: *CoRR* abs/1905.09550 (2019).

[OS20] Kenta Oono and Taiji Suzuki. “Graph Neural Networks Exponentially Lose Expressive Power for Node Classification”. In: *International Conference on Learning Representations (ICLR)*. 2020. URL: https://openreview.net/forum?id=S1ldO2EFPr.

[OVK18] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. *Neural Discrete Representation Learning*. 2018. arXiv: 1711.00937 [cs.LG].

[Ope23] OpenAI. “GPT-4 Technical Report”. In: *ArXiv* abs/2303.08774 (2023). URL: https://arxiv.org/abs/2303.08774.

[PC99] Victor Y Pan and Zhao Q Chen. “The complexity of the matrix eigenproblem”. In: *Proceedings of the 31th annual ACM Cymposium on Theory of Computing*. 1999, pp. 507–516.

[PSH21] Xuran Pan, Shiji Song, and Gao Huang. *A Unified Framework for Convolution-based Graph Neural Networks*. 2021. URL: https://openreview.net/forum?id=zUMD--Fb9Bt.

[Pan+21] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. “Deep learning for anomaly detection: A review”. In: *ACM computing surveys (CSUR)* 54.2 (2021), pp. 1–38.

[PAI13] Evangelos Papalexakis, Leman Akoglu, and Dino Ience. “Do more views of a graph help? community detection and clustering in multi-graphs”. In: *International Conference on Information Fusion*. IEEE. 2013, pp. 899–905.

[Ped+11] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “Deepwalk: Online learning of social representations”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710.

[Pol+20] Daniil Polykovskiy et al. “Molecular Sets (MOSES): A Benchmarking Platform for Molecular Generation Models”. In: *Frontiers in Pharmacology* (2020).

[Qia+22] Chendi Qian, Gaurav Rattan, Floris Geerts, Christopher Morris, and Mathias Niepert. “Ordered subgraph aggregation networks”. In: *arXiv preprint arXiv:2206.11168* (2022).

[Qiu+22] Chen Qiu, Marius Kloft, Stephan Mandt, and Maja Rudolph. “Raising the bar in graph-level anomaly detection”. In: *arXiv preprint arXiv:2205.13845* (2022).

[QBT19] Meng Qu, Yoshua Bengio, and Jian Tang. “GMNN: Graph Markov Neural Networks”. In: *International Conference on Machine Learning*. 2019, pp. 5241–5250.

[Raf16] Colin Raffel. “Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching”. PhD thesis. Columbia University, 2016. DOI: https://doi.org/10.7916/D8N58MHV.

[Ram+14] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. “Quantum chemistry structures and properties of 134 kilo molecules”. In: *Scientific data* 1.1 (2014), pp. 1–7.

[Ram+22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. “Hierarchical text-conditional image generation with clip latents”. In: *arXiv preprint arXiv:2204.06125* 1.2 (2022), p. 3.

[RA15] Shebuti Rayana and Leman Akoglu. “Collective opinion spam detection: Bridging review networks and metadata”. In: *SIGKDD*. 2015, pp. 985–994.

[RC77] Ronald C Read and Derek G Corneil. “The graph isomorphism disease”. In: *Journal of graph theory* 1.4 (1977), pp. 339–363.

[RB08] Kaspar Riesen and Horst Bunke. “IAM graph database repository for graph based pattern recognition and machine learning”. In: *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer. 2008, pp. 287–297.

[Rin+95] Andy Rindos, Steven Woolet, Ioannis Viniotis, and Kishor Trivedi. “Exact methods for the transient analysis of nonhomogeneous continuous time Markov chains”. In: *Computations with Markov Chains: Proceedings of the 2nd International Workshop on the Numerical Solution of Markov Chains*. Springer. 1995, pp. 121–133.

[Ron+19] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. “The Truly Deep Graph Convolutional Networks for Node Classification.” In: *CoRR* abs/1907.10903 (2019).

[Ros+20] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. “Sign: Scalable inception graph neural networks”. In: *arXiv preprint arXiv:2004.11198* (2020).

[RKS20] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. “Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs”. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 3125–3132.

[Ruf+20] Lukas Ruff et al. “A Unifying Review of Deep and Shallow Anomaly Detection.” In: *CoRR* abs/2009.11732 (2020). URL: http://dblp.uni-trier.de/db/journals/corr/corr2009.html#abs-2009-11732.

[Ruf+18] Lukas Ruff et al. “Deep one-class classification”. In: *International conference on machine learning*. 2018, pp. 4393–4402.

[SK16] Tim Salimans and Durk P Kingma. “Weight normalization: A simple reparameterization to accelerate training of deep neural networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 901–909.

[SVH21] Dylan Sandfelder, Priyesh Vijayan, and William L Hamilton. “Ego-GNNs: Exploiting Ego Structures in Graph Neural Networks”. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pp. 8523–8527.

[San+21] Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. "Graph neural networks for friend ranking in large-scale social platforms". In: *Proceedings of the Web Conference 2021*. 2021, pp. 2535–2546.

[Sat20] Ryoma Sato. "A Survey on The Expressive Power of Graph Neural Networks." In: *CoRR* abs/2003.04078 (2020). URL: http://dblp.uni-trier.de/db/journals/corr/corr2003.html#abs-2003-04078.

[SYK21] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. "Random features strengthen graph neural networks". In: *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM. 2021, pp. 333–341.

[SMG13] Andrew M Saxe, James L McClelland, and Surya Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks". In: *arXiv preprint arXiv:1312.6120* (2013).

[Sch+13] Christian M Schneider, Vitaly Belik, Thomas Couronné, Zbigniew Smoreda, and Marta C González. "Unravelling daily human mobility motifs". In: *Journal of The Royal Society Interface* 10.84 (2013), p. 20130246.

[Sch+04] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. "BRENDA, the enzyme database: updates and major new developments". In: *Nucleic acids research* 32.suppl_1 (2004), pp. D431–D433.

[Sen+08] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. "Collective classification in network data". In: *AI magazine* 29.3 (2008), pp. 93–93.

[Seu+17] Mathias Seuret, Michele Alberti, Marcus Liwicki, and Rolf Ingold. "PCA-initialized deep neural networks applied to document image analysis". In: *2017 14th IAPR International Conference on Document Analysis and Recognition*. Vol. 1. IEEE. 2017, pp. 877–882.

[Sha+16] Nauman Shahid, Nathanael Perraudin, Vassilis Kalofolias, Gilles Puy, and Pierre Vandergheynst. "Fast robust PCA on graphs". In: *IEEE Journal of Selected Topics in Signal Processing* 10.4 (2016), pp. 740–756.

[Shc+18] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. "Pitfalls of graph neural network evaluation". In: *arXiv preprint arXiv:1811.05868* (2018).

[She+11] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. "Weisfeiler-lehman graph kernels." In: *Journal of Machine Learning Research* 12.9 (2011).

[Shi+20] Chence Shi*, Minkai Xu*, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. "GraphAF: a Flow-based Autoregressive Model for Molecular Graph Generation". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=S1esMkHYPr.

[Shi+19] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. "Skeleton-based action recognition with directed graph neural networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7912–7921.

[Shu+13] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains". In: *IEEE Signal Processing magazine* 30.3 (2013), pp. 83–98.

[Sig+20] Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis. "GraKeL: A Graph Kernel Library in Python." In: *Journal of Machine Learning Research* 21.54 (2020), pp. 1–5.

[SK18] Martin Simonovsky and Nikos Komodakis. "Graphvae: Towards generation of small graphs using variational autoencoders". In: *International conference on artificial neural networks*. Springer. 2018, pp. 412–422.

[SD+15] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and sunrya Ganguli. "Deep unsunpervised learning using nonequilibrium thermodynamics". In: *International Conference on Machine Learning*. PMLR. 2015, pp. 2256–2265.

[SME21] Jiaming Song, Chenlin Meng, and Stefano Ermon. "Denoising Diffusion Implicit Models". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=St1giarCHLP.

[SE19] Yang Song and Stefano Ermon. "Generative modeling by estimating gradients of the data distribution". In: *Advances in neural information processing systems* 32 (2019).

[Son+21] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. "Score-Based Generative Modeling through Stochastic Differential Equations". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=PxTIG12RRHS.

[Sot+23] Konstantinos Sotiropoulos*, Lingxiao Zhao*, Pierre Jinghong Liang, and Leman Akoglu. "ADAMM: Anomaly Detection of Attributed Multigraphs with Metadata: A Unified Neural Network Approach". In: *2023 IEEE International Conference on Big Data (BigData)*. IEEE. 2023, pp. 865–874.

[SR20] Balasubramaniam Srinivasan and Bruno Ribeiro. "On the Equivalence between Positional Node Embeddings and Structural Graph Representations". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=SJxzFySKwH.

[Sri+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[Sun+23] Haoran Sun, Lijun Yu, Bo Dai, Dale Schuurmans, and Hanjun Dai. "Score-based Continuous-time Discrete Diffusion Models". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=BYWWwSY2G5s.

[Swe+16] Lorne Swersky, Henrique O Marques, Jöerg Sander, Ricardo JGB Campello, and Arthur Zimek. "On the evaluation of outlier detection and one-class classification methods". In: *2016 IEEE international conference on data science and advanced analytics (DSAA)*. IEEE. 2016, pp. 1–10.

[TJ20] Behrooz Tahmasebi and Stefanie Jegelka. “Counting Substructures with Higher-Order Graph Neural Networks: Possibility and Impossibility Results”. In: *arXiv preprint arXiv:2012.03174* (2020).

[TLD09] Wei Tang, Zhengdong Lu, and Inderjit S Dhillon. “Clustering with multiple graphs”. In: *ICDM*. IEEE. 2009, pp. 1016–1021.

[TD04] David MJ Tax and Robert PW Duin. “Support vector data description”. In: *Machine learning* 54 (2004), pp. 45–66.

[TC00] John Shawe Taylor and Nello Cristianini. “Support Vector Machines and other kernel-based learning methods”. In: *Cambridge University* (2000).

[TB07] Ambuj Tewari and Peter L Bartlett. “On the Consistency of Multiclass Classification Methods.” In: *Journal of Machine Learning Research* 8.5 (2007).

[TZK21] Erik Henning Thiede, Wenda Zhou, and Risi Kondor. “Autobahn: Automorphism-based Graph Neural Nets”. In: (2021).

[Ton+21] Xiaochu Tong et al. “Generative models for De Novo drug design”. In: *Journal of Medicinal Chemistry* 64.19 (2021), pp. 14011–14027.

[Top+21] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. “Understanding oversquashing and bottlenecks on graphs via curvature”. In: *arXiv preprint arXiv:2111.14522* (2021).

[Tri+21] Jeanne Trinquier, Guido Uguzzoni, Andrea Pagnani, Francesco Zamponi, and Martin Weigt. “Efficient generative modeling of protein sequences using simple autoregressive models”. In: *Nature communications* 12.1 (2021), p. 5800.

[Vel+18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. “Graph Attention Networks”. In: *International Conference on Learning Representations (ICLR)*. 2018.

[Vel+19] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. “Deep Graph Infomax”. In: *International Conference on Learning Representations (ICLR)*. 2019. URL: https://openreview.net/forum?id=rklz9iAcKQ.

[VZ17] Saurabh Verma and Zhi-Li Zhang. “Hunt for the unique, stable, sparse and fast feature learning on graphs”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 88–98.

[Vig+23] Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. “DiGress: Discrete denoising diffusion for graph generation”. In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=UaAD-Nu86WX.

[VLF20] Clément Vignac, Andreas Loukas, and Pascal Frossard. “Building powerful and equivariant graph neural networks with structural message-passing”. In: *Advances in Neural Information Processing Systems*. 2020.

[Wag+13] Raimar Wagner, Markus Thom, Roland Schweiger, Günther Palm, and Albrecht Rothermel. “Learning convolutional neural networks from few samples”. In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2013, pp. 1–7.

[WWK08] Nikil Wale, Ian A Watson, and George Karypis. “Comparison of descriptor spaces for chemical compound retrieval and classification”. In: *Knowledge and Information Systems* 14.3 (2008), pp. 347–375.

[Wan+23] Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. “Can Language Models Solve Graph Problems in Natural Language?” In: *arXiv preprint arXiv:2305.10037* (2023).

[Wan+20] Xuhong Wang, Ying Du, Ping Cui, and Yupu Yang. “OCGNN: One-class Classification with Graph Neural Networks”. In: *arXiv preprint arXiv:2002.09594* (2020).

[Web+19] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I. Weidele, Claudio Bellei, Tom Robinson, and Charles E. Leiserson. *Anti-Money Laundering in Bitcoin: Experimenting with Graph Convolutional Networks for Financial Forensics*. 2019. arXiv: 1908.02591 [cs.SI].

[WL68] B Weisfeiler and A Leman. “The reduction of a graph to canonical form and the algebgra which appears therein”. In: *NTI, Series* 2 (1968).

[Wei76] Boris Weisfeiler. “On Construction and Identification of Graphs”. In: *LECTURE NOTES IN MATHEMATICS*. Citeseer. 1976.

[Wu+19] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. “Simplifying Graph Convolutional Networks.” In: *ICML*. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 6861–6871.

[Wu+20a] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. “Graph neural networks in recommender systems: a survey”. In: *ACM Computing Surveys (CSUR)* (2020).

[Wu+18] Zhenqin Wu et al. “MoleculeNet: a benchmark for molecular machine learning”. In: *Chemical science* 9.2 (2018), pp. 513–530.

[Wu+20b] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. “A comprehensive survey on graph neural networks”. In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[Xu+19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. “How Powerful are Graph Neural Networks?” In: *International Conference on Learning Representations (ICLR)*. 2019. URL: https://openreview.net/forum?id=ryGs6iA5Km.

[Xu+18] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. “Representation Learning on Graphs with Jumping Knowledge Networks”. In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. 2018, pp. 5453–5462.

[Xu+21] Keyulu Xu, Mozhi Zhang, Jingling Li, Simon Shaolei Du, Ken-Ichi Kawarabayashi, and Stefanie Jegelka. “How Neural Networks Extrapolate: From Feedforward to Graph Neural Networks”. In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=UH-cmocLJC.

[Xu+22] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. "GeoDiff: A Geometric Diffusion Model for Molecular Conformation Generation". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=PzcvxEMzvQC.

[Yan+23] Qi Yan, Zhengyang Liang, Yang Song, Renjie Liao, and Lele Wang. "Swingnn: Rethinking permutation invariance in diffusion models for graph generation". In: *arXiv preprint arXiv:2307.01646* (2023).

[YV15] Pinar Yanardag and SVN Vishwanathan. "Deep graph kernels". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2015, pp. 1365–1374.

[Yin+21] Chengxuan Ying et al. "Do Transformers Really Perform Bad for Graph Representation?" In: (2021).

[You18] Jiaxuan You. *Caveman Dataset*. https://github.com/JiaxuanYou/graph-generation/blob/master/create_graphs.py. [Online; accessed 31-Jan-2024]. 2018.

[You+21] Jiaxuan You, Jonathan M Gomes-Selman, Rex Ying, and Jure Leskovec. "Identity-aware Graph Neural Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 12. 2021, pp. 10737–10745.

[YYL19] Jiaxuan You, Rex Ying, and Jure Leskovec. "Position-aware graph neural networks". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7134–7143.

[You+18] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. "Graphrnn: Generating realistic graphs with deep auto-regressive models". In: *International conference on machine learning*. PMLR. 2018, pp. 5708–5717.

[YHL15] Rose Yu, Xinran He, and Yan Liu. "Glad: group anomaly detection in social media analysis". In: *TKDD* 10.2 (2015), pp. 1–22.

[Yu+18] Wenchao Yu, Wei Cheng, Charu C. Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. "NetWalk: A Flexible Deep Embedding Approach for Anomaly Detection in Dynamic Networks". In: *KDD*. ACM, 2018, pp. 2672–2681.

[Yun+20] Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. "Are Transformers universal approximators of sequence-to-sequence functions?" In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=ByxRM0Ntvr.

[Zah+17] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. "Deep Sets". In: *Advances in Neural Information Processing Systems* 30 (2017).

[Zha+23a] Bohang Zhang, Guhao Feng, Yiheng Du, Di He, and Liwei Wang. "A complete expressiveness hierarchy for subgraph gnns via subgraph weisfeiler-lehman tests". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 41019–41077.

[Zha+24a] Bohang Zhang, Jingchu Gai, Yiheng Du, Qiwei Ye, Di He, and Liwei Wang. "Beyond Weisfeiler-Lehman: A Quantitative Framework for GNN Expressiveness". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=HSKaGOi7Ar.

[Zha+23b] Chenshuang Zhang, Chaoning Zhang, Mengchun Zhang, and In So Kweon. "Text-to-image diffusion model in generative ai: A survey". In: *arXiv preprint arXiv:2303.07909* (2023).

[Zha+22a] Ge Zhang et al. "Dual-discriminative graph neural network for imbalanced graph-level anomaly detection". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24144–24157.

[Zha+18] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. "An End-to-End Deep Learning Architecture for Graph Classification." In: *AAAI*. Vol. 18. 2018, pp. 4438–4445.

[ZL21] Muhan Zhang and Pan Li. "Nested Graph Neural Networks". In: *Advances in Neural Information Processing Systems* 34 (2021).

[Zha+21] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. "Labeling trick: A theory of using graph neural networks for multi-node representation learning". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 9061–9073.

[Zha+20] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. "Revisiting graph neural networks for link prediction". In: (2020).

[ZZ12] Zhenyue Zhang and Keke Zhao. "Low-rank matrix approximation with manifold regularization". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.7 (2012), pp. 1717–1729.

[ZA20a] Lingxiao Zhao and Leman Akoglu. "Connecting graph convolutional networks and graph-regularized pca". In: *arXiv preprint arXiv:2006.12294* (2020).

[ZA23] Lingxiao Zhao and Leman Akoglu. "On using classification datasets to evaluate graph outlier detection: Peculiar observations and new insights". In: *Big Data* 11.3 (2023), pp. 151–180.

[ZA20b] Lingxiao Zhao and Leman Akoglu. "PairNorm Tackling Oversmoothing in {GNN}s". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=rkecl1rtwB.

[ZDA24] Lingxiao Zhao, Xueying Ding, and Leman Akoglu. "Pard: Permutation-Invariant Autoregressive Diffusion for Graph Generation". In: *arXiv preprint arXiv:2402.03687* (2024).

[Zha+24b] Lingxiao Zhao, Xueying Ding, Lijun Yu, and Leman Akoglu. "Improving and Unifying Discrete&Continuous-time Discrete Denoising Diffusion". In: *arXiv preprint arXiv:2402.03701* (2024).

[Zha+22b] Lingxiao Zhao, Louis Härtel, Neil Shah, and Leman Akoglu. "A Practical, Progressively-Expressive GNN". In: *36th Conference on Neural Information Processing Systems*. 2022. URL: https://openreview.net/forum?id=WBv9Z6qpA8x.

[Zha+22c] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. "From Stars to Subgraphs: Uplifting Any GNN with Local Structure Awareness". In: *International Conference on Learning Representations*. 2022. URL: `https://openreview.net/forum?id=Mspk_WYKoEH`.

[Zha+22d] Lingxiao Zhao, Saurabh Sawlani, Arvind Srinivasan, and Leman Akoglu. "Graph anomaly detection with unsupervised GNNs". In: *Preprint arXiv:2210.09535* (2022).

[ZSA22] Lingxiao Zhao, Neil Shah, and Leman Akoglu. "A practical, progressively-expressive GNN". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 34106–34120.

[ZRA20] Yue Zhao, Ryan A Rossi, and Leman Akoglu. "Automating outlier detection via meta-learning". In: *arXiv preprint arXiv:2009.10606* (2020).

[Zhe+23] Lin Zheng, Jianbo Yuan, Lei Yu, and Lingpeng Kong. "A Reparameterized Discrete Diffusion Model for Text Generation". In: *arXiv preprint arXiv:2302.05737* (2023).

[Zhe+20] Chen Zhengdao, Chen Lei, Villar Soledad, and Joan Bruna. "Can Graph Neural Networks Count Substructures?" In: *Advances in neural information processing systems* (2020).

[Zhu+20] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. "Beyond homophily in graph neural networks: Current limitations and effective designs". In: *Advances in neural information processing systems* 33 (2020), pp. 7793–7804.

[Zhu+21] Meiqi Zhu, Xiao Wang, Chuan Shi, Houye Ji, and Peng Cui. "Interpreting and Unifying Graph Neural Networks with An Optimization Framework". In: *arXiv preprint arXiv:2101.11859* (2021).

[ZG02] Xiaojin Zhu and Zoubin Ghahramani. "Learning from Labeled and Unlabeled Data with Label Propagation". In: *None*. 2002.

[ZGL03] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. "Semi-supervised learning using gaussian fields and harmonic functions". In: *Proceedings of the 20th International conference on Machine learning (ICML-03)*. 2003, pp. 912–919.