Innovations in Graph Neural Network Design: Addressing Oversmoothing, Heterophily, and Information Propagation

Kushal Bose

Electronics and Communication Sciences Unit

Indian Statistical Institute

A thesis submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy
in Computer Science

To my Parents

They taught me to think.
They taught me to question.

Acknowledgements

First and foremost, I want to express my deepest appreciation to my advisor, Prof (Dr) Swagatam Das. I am tremendously thankful to him not just for welcoming me as his mentee but also for guiding me through the fundamental principles of conducting independent, high-quality research. From identifying research problems to articulating proposed solutions in well-crafted manuscripts, he consistently provided support by offering his expertise and wisdom. Without his precious guidance and counsel, this dissertation would never have come to fruition.

I wish to use this moment to acknowledge the teaching staff of the Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata, for providing me with their valuable suggestions. I want to convey my appreciation to the personnel of the Electronics and Communication Sciences Unit, the Dean's, and the Director's offices for their dedicated assistance. I also want to acknowledge my colleague researchers, who throughout these years have developed into my cherished companions.

Another individual who has similarly contributed significantly to this dissertation is my sole collaborator, Saptarshi Banerjee, who enthusiastically assisted me in successfully completing the research endeavors detailed in this thesis. I would be delighted to use this moment to wish him the very best in his future pursuits.

I would also like to acknowledge the Indian Statistical Institute for funding my research and the Electronics and Communication Sciences Unit for providing the required infrastructure.

Last but not least, I like to thank my parents and friends; the people part of my immediate and extended family. I am immensely grateful to them for their selfless love, constant support, and encouragement through thick and thin.

Abstract

In an unstructured learning paradigm, Graph Neural Networks (GNNs) adeptly tackle graph data like social networks, molecules, transaction networks, etc. In the primitive stage, GNNs are designed to be shallow, comprising two or three layers. Emulating the success of deep CNNs, deep GNNs are also proposed by stacking multiple layers. Those multi-layered GNNs are pivotal in enabling long-range interactions where multi-hop neighbors carry significant information, like molecular property prediction. Yet, the multi-layered GNNs face challenges of Oversmoothing, where node features become indistinguishable due to the recursive nature of message passing. In the second chapter of the thesis, we propose a non-recursive message passing technique to address oversmoothing. Our method explores random paths and computes path features, and those are subsequently aggregated to update the node features. The multi-hop message passing also depends on the homophily or heterophily settings of the network. GNNs typically perform better in homophilic settings where adjacent nodes share identical class labels. Conversely, the performance of GNNs is exacerbated in the heterophilic networks where adjacent nodes may have different class labels. In the third chapter of the thesis, we address the challenges of graph heterophily by rewiring the graph topology. We learn the similarity scores of the edges obtained from the autoencoder-based class representations. The impressive performances on heterophilic benchmarks reaffirm the superiority of our approach. We also study the effects of rewiring special edges like self-loops and parallel edges. In the fourth chapter of the thesis, we investigate the effects of the addition of self-loops and parallel edges on the eigenvalues of the graph Laplacian. Empirically, we observe that the gradual addition of self-loops or parallel edges generates performance trends (either increasing or decreasing) on the heterophilic graphs. This work offers insights into the graph spectrum based on the observed performance trends, bypassing the need to execute expensive eigenvalue decomposition. The deep GNNs also suffer from Oversquashing, an information bottleneck arises due to the requirement of storing exponentially growing information into fixed capacity channels. In the fifth chapter of the thesis, we propose asynchronous message passing to utilize fixed-capacity channels in a time-dependent access. This prevents the capacity constraints and ultimately overcomes oversquashing. We achieved commendable performances on the REDDIT-BINARY and Peptides-struct datasets. To mitigate both oversmoothing and oversquashing, Graph Transformers (GTs) come into the scenario to enable pair-wise message passing across the network. Precisely, GT incorporates structural information of the underlying graph datasets via positional encodings. In the sixth chapter of the thesis, we designed a novel and efficient positional encoding that is learnable and maps the encodings into hyperbolic spaces. Our positional encodings are expressive and efficiently capture hierarchical structures embedded in the molecular graphs, which is validated by extensive theoretical underpinnings. We further demonstrate that hyperbolic positional encodings, when added with features in final layers, diminish the effects of oversmoothing. We achieved superior performance on MNIST and OGBG-MOLHIV graphs by employing hyperbolic positional encodings. In the seventh chapter of the thesis, we shed light on the potential future research avenues and scope in the domain of GNN.

Contents

List of Figures

List of Tables

List of Notations

$\mathcal{G}$	A graph
$\mathcal{V}$	A set of nodes
$\mathcal{E}$	A set of edges
$\lvert\mathcal{E}\rvert$	Number of edges
n	Number of nodes in the graph
d	Feature dimension of each node
$\mathbf{X}$	Feature matrix of a input graph
$\mathbf{X}^{(k)}$	Node embeddings at k-th layer
$\mathcal{A}$	Adjacency matrix of the graph
$\mathcal{D}$	Degree matrix
L	Unnormalized graph Laplacian
$\tilde{L}$	Symmetrically normalized graph Laplacian
$\tilde{A}$	Self-loop augmented adjacency matrix
$\tilde{D}$	Self-loop augmented degree matrix
A_N	Symmetrically normalized adjacency matrix
$\tilde{D}$	Degree matrix augmented with self-loop
$\tilde{A}_\alpha$	Adjacency matrix augmented with α-time self-loops
$\tilde{D}_\alpha$	Degree matrix augmented with α-time self-loops
$\tilde{A}_\gamma$	Adjacency matrix augmented with γ-times parallel edges with one self-loops
$\tilde{D}_\gamma$	Degree matrix augmented with γ-times parallel edges with one self-loops
$\tilde{A}_N^\alpha$	Symmetrically normalized adjacency matrix augmented with α-times self-loops
$\tilde{A}_N^\gamma$	Symmetrically normalized adjacency matrix augmented with γ-times parallel edges
$\tilde{L}_\alpha$	Symmetrically normalized graph Laplacian augmented with α-times self-loops
$\tilde{L}_\gamma$	Symmetrically normalized graph Laplacian augmented with γ-times parallel edges
$\mathbb{R}^n$	n-dimensional real space
$\mathcal{M}$	Manifold
c	Space curvature

$\mathbb{B}_c^n$	n-dimensional Poincarê Ball with space curvature c
$\mathbb{H}_c^n$	n-dimensional Hyperboloid space with space curvature c
$d_{\mathbb{E}}(x, y)$	Euclidean distance between two points x and y
$d_{\mathbb{B}}(x, y)$	Distance between two points x and y on Poincarê Ball $\mathbb{B}$
$d_{\mathbb{H}}(x, y)$	Distance between two points x and y on Hyperboloid space $\mathbb{H}$
$\mathcal{T}_x\mathcal{M}$	Tangent space at x on $\mathcal{M}$
$g^{\mathcal{M}}(.,.)$	Riemannian metric equipped with manifold $\mathcal{M}$
$\exp_x^{(c)}$	Exponential map at point x with curvature c
$\log_x^{(c)}$	Logarithmic map at point x with curvature c
tanh	Hyperbolic tangent function
sinh	Hyperbolic sin function
cosh	Hyperbolic cosine function
$\|.\|_2$	Euclidean norm
$\|.\|_1$	Absolute norm
$\|P\|_F$	Frobenius norm of matrix P
$\odot$	Elementwise product
$W_l(W^{(l)})$	Learnable weight matrix at l^{th} layer
$\mathcal{D}_i(d_i)$	Degree of node i
$\mathcal{A}_{ij}(W_{ij})$	$(i, j)^{th}$ Element of adjacency matrix
W_Q	Query matrix
W_K	Key matrix
W_V	Value matrix
$M_{ij}(\phi(i, j))$	Computed message between node i and node j
$\mathcal{H}_n$	Node homophily ratio
$\mathcal{H}_e$	Edge homophily ratio
v_i	i^{th} node in the graph
$x_i(X_i)$	Features of i^{th} node
R_{total}	Total Effective Resistance
$h_{\odot}^{(m)}$	Signal propagated with m-layers of GNN

List of Abbreviations and Acronyms

GDL	Geometric Deep Learning
MP	Message Passing
MAE	Mean Absolute Error
AP	Average Precision
AUROC	Area Under the Receiver Operating Characteristics
PE	Positional Encodings
LapPE	Laplacian Positional Encodings
RWPE	Random Walk Positional Encodings
GT	Graph Transformer
GNN	Graph Neural Network
MLP	Multi-layered Perceptron
AE	Autoencoder
DE	Dirichlet Energy
BN	Batch Normalization
LN	Layer Normalization
HyPE	Hyperbolic Positional Encodings
GAT	Graph Attention Network
CAMP	Centrality-guided Asynchronous Message Passing
GCN	Graph Convolutional Networks
MPNN	Message Passing Neural Networks
MSG	Message function
AGG	Aggregate function
UP	Update function
LGR	Label-guided Graph Rewiring
k-NN	k Nearest Neighbor Algorithm
CT	Commute time
HT	Hitting time
RPE	Randomized Path Explorer
ACC	Accuracy
RAMP	Randomized Asynchronous Message Passing
GAT	Graph Attention Network

MSE	Mean Squared Error
CE	Cross-entropy
ReLU	Rectified Linear Unit
Norm	Normalization layer
MHA	Multi-head attention
MSA	Mulit-heas self-attention
FFN	Feed-forward network
GP	Gaussian Process
Enc	Encoder
Dec	Decoder

Chapter 1

Long Range Interactions in Graph Neural Networks

Summary

Graph Neural Networks (GNNs) made remarkable strides in handling graph-structured data over the last decade. Most of the design choices for GNNs are limited to local neighborhoods, devoid of interactions among long-range nodes. As a consequence, shallow GNNs suffer from the problem of underreaching. As an immediate remedy, multi-layered GNNs appear in the landscape to guarantee the long-range interactions. More importantly, the deep GNNs face the challenges of oversmoothing and Oversquashing, creating hindrances to the effective information propagation to the long-distance nodes. A myriad of tasks, like molecular property prediction, communication between social network communities, etc, demand long-range message passing to attain optimal performance. The limitations have garnered widespread attention from researchers for designing GNN architectures that seamlessly propagate messages between multi-hop nodes. In this introductory chapter, we first formally discuss the fundamentals of message passing and neighborhood aggregation methods. Secondly, we offer an interconnected discourse of various problems in GNNs with major research directions and key developments. Finally, we provide an illustrative roadmap of the thesis, highlighting the original contributions for the rest of the chapters.

1.1 Preliminaries on Graph Neural Networks

In a world of complex dynamics, relations between the entities are attributed to the astronomical rise of graph-structured data. The ubiquitous presence of graphs compels the researchers to delve deeper, leading to the development of Geometric Deep Learning (GDL) (Bronstein et al., 2017; Masci et al., 2016). This paradigm predominantly focuses on learning and building AI models that apply to unstructured datasets. The pervasive dominance of graph data like social networks, financial transaction networks, molecular graphs, brain networks, etc, inspires the communities to develop efficient learning algorithms on graphs. The genesis of the graph learning algorithms begins with proposing Graph Neural Networks (GNNs) (Scarselli et al., 2008). GNNs currently emerge as a promising tool for learning graph-structured data by solving a wide array of downstream tasks like node-level tasks, edge-level tasks, graph-level tasks, and graph generative modeling.

The learning on graphs is enabled by employing the Message Passing (MP) framework (Gilmer et al., 2017b). Unlike traditional approaches, the graph comes with both features and structural information. The learning is accomplished by passing messages between the adjacent nodes. The exchanged messages empower GNNs for downstream tasks effectively. The following set of equations in Eq: 1.1 delineates the message passing operation. MP framework is comprised of three pivotal components, like MSG(.,.), AGG({.}), and UP(.,.) functions, respectively, perform message computation between adjacent nodes, aggregate the computed messages, and update the central node features with the aggregated messages.

$$
\begin{aligned}
m_{ij}^{(l+1)} &= \mathrm{MSG}^{(l+1)}(h_i^{(l)}, h_j^{(l)}) \\
h_i^{(l+1)} &= \mathrm{AGG}^{(l+1)}(\{m_{ij}^{(l)} | j \in N(i)\}) \\
x_i^{(l+1)} &= \mathrm{UP}^{(l+1)}(x_i^{l}, h_i^{(l+1)})
\end{aligned}
\tag{1.1}
$$

We represent the layer-wise message passing operations and each function constitutes trainable parameters learned during the training procedure. The equations above update node-level features. To obtain graph-level features, we apply pooling over the node features for the entire graphs which is described in the Eq: 1.2. The READOUT function acts as a pooling function (like sum, mean, or max) applied over the node features to estimate a single

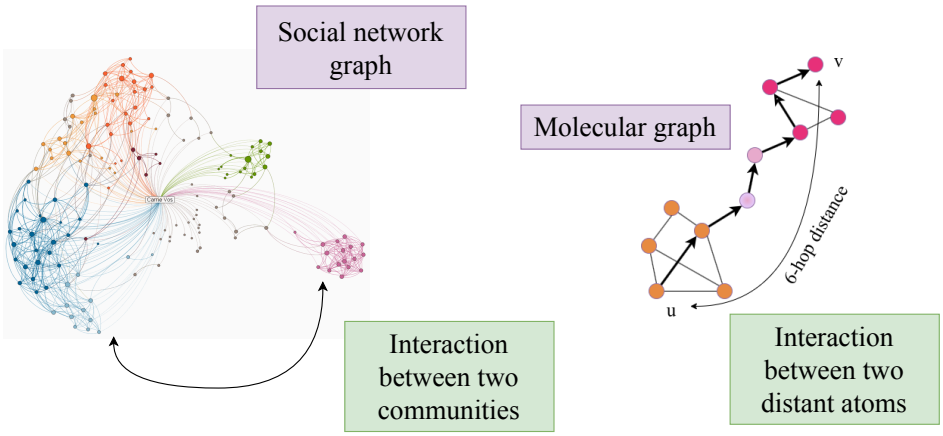


Figure 1.1: An illustrative example where long-range message passing will be necessary to aggregate informative features.

representation.

$$g = \text{READOUT}(\{x_i | i \in [1, n]\}) \tag{1.2}$$

The readout methods are widely used in graph-level tasks like graph classification and graph regression. Notably, the variation of three different functions led to the novel architectures. For instance, if we consider $\text{MSG}(h_i, h_j) = h_j$, $\text{AGG}(\{\})$ as a mean of features, and $\text{UP}(.,.)$ as addition, then the resulting model will be termed as Graph Convolutional Networks (GCN) (Kipf and Welling, 2016). Similarly, several architectures are proposed like GraphSage (Hamilton et al., 2017), GAT (Veličković et al., 2017), PNA (Corso et al., 2020), GCNII (Chen et al., 2020c), etc.

1.2 Challenges of Long-range Interactions: An Overview

The prevailing graph neural networks predominantly aggregate from local neighborhoods. Beyond its localized message passing strategy, numerous tasks require interactions among the multi-hop neighbors, like molecular property predictions (Morris et al., 2020; Dwivedi et al., 2022). Refer to Figure 1.1 to visualize the utility of long-range interactions in the graph data. The straightforward approach to enable long-range message passing is to stack multiple message passing layers. The deep GNN architectures come with their multi-faceted challenges. In the following subsections, we will describe the hindrances that are faced by GNNs when multi-hop message passing is applied. Furthermore, in the subsequent chapters,

we will discuss our attempts to address the problems in multi-layered GNNs.

1.3 Oversmoothing

Definition 1.1 (Oversmoothing (Rusch et al., 2023)). *Let $\mathcal{G}$ be an undirected, connected graph, and $X^{(k)} \in \mathbb{R}^{n\times d}$ denote the k-th layer hidden features of an L-layered GNN defined on $\mathcal{G}$. Moreover, we call $\mu : \mathbb{R}^{n\times d} \to \mathbb{R}_{\geq 0}$ a node-similarity measure if it satisfies the following axioms:*

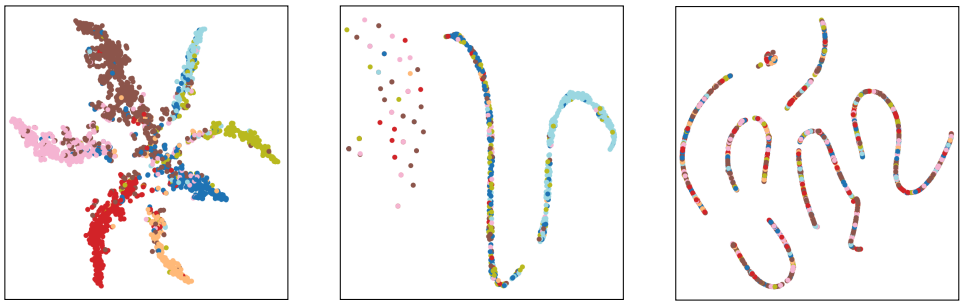
1. *$\exists c \in \mathbb{R}^d$ with $X_i = c$ for all nodes $i \in \mathcal{V} \Leftrightarrow \mu(X) = 0$, for $X \in \mathbb{R}^{n\times d}$.*
2. *$\mu(X + Y) \leq \mu(X) + \mu(Y)$, for all $X, Y \in \mathbb{R}^{n\times d}$.*

   We then define over-smoothing with respect to μ as the layer-wise exponential convergence of the node-similarity measure μ to zero, i.e.,
3. *$\mu(X^{(k)}) \leq C_1 \exp^{-C_2 n}$, for $n = 0, \cdots, N$ with some constants $C_1, C_2 > 0$.*

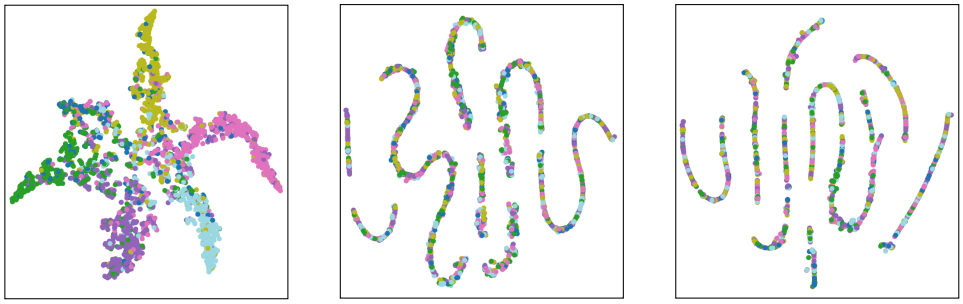
Definition 1.2 (Dirichlet energy). *For an input graph $\mathcal{G}$ with symmetrically normalized Laplacian $\tilde{\Delta}$, then Dirichlet energy for the feature set $X^{(k)}$ is defined as following,*

$$DE(X^{(k)}, \tilde{\Delta}) = tr({X^{(k)}}^\top \tilde{\Delta} X^{(k)}) = \frac{1}{2} \sum a_{ij} \| \frac{x_i^{(k)}}{\sqrt{1+d_i}} - \frac{x_j^{(k)}}{\sqrt{1+d_j}} \|_2^2, \tag{1.3}$$

The concept of "deep learning" naturally comes with the idea of building models by stacking multiple hidden layers and extracting meaningful features from the deeper layers. Naturally, researchers attempted to build GNNs with multiple layers to gain in performance. Unfortunately, the performance of the underlying GNN models deteriorates with the increasing number of convolutional layers. This phenomenon is commonly known as *Oversmoothing*, primarily observed by (Li et al., 2018). The consecutive feature aggregation transforms the node features into indistinguishable ones. We applied a multi-layered GCN on Cora and Citeseer to observe the node embeddings. Refer to Figure 1.2 for a detailed illustration in the backdrop of deeper convolutional architectures.



(a) Cora on 2-layered GCN (b) Cora on 8-layered GCN (c) Cora on 32-layered GCN



(d) Citeseer on 2-layered GCN (e) Citeseer on 8-layered GCN (f) Citeseer on 32-layered GCN

Figure 1.2: The node embeddings of Cora and Citeseer obtained from GCN are presented. The cluster structures tangibly collapsed with the increasing network depths attributed to the occurrence of oversmoothing.

1.3.1 Methods to Counter Oversmoothing

1.3.1.1 Novel Message Passing Techniques

APPNP (Klicpera et al., 2018) employs a costlier PageRank (Page et al., 1999) matrix to perform higher-order neighborhood aggregation while (Klicpera et al., 2019) relies on graph diffusion to learn the same. GPR-GNN (Chien et al., 2020) jointly optimizes node features and topological information irrespective of the node homophily or node heterophily, where Ortho-GConv (Guo et al., 2022) relies on orthogonal feature transformation to minimize the effect of over-smoothing. The over-smoothing problem is also possible in the hypergraphs discussed in (Chen and Zhang, 2022). NRGCN (Chen et al., 2021) proposes a non-recursive aggregation strategy that learns node embedding by extracting the information from the hops of the current set of neighbors. However, the work is not intended to tackle over-smoothing. On the other hand, GeniePath (Liu et al., 2019b) explores the receptive paths of the graph

in both breadth and width direction of the respective neighborhoods where SPAGAN (Yang et al., 2021) aggregates features from the higher-order neighbors by incorporating path-based attention where shortest paths are considered. Another approach, Deeply Supervised GNN (DSGNN) (Yang et al., 2020) learned representations from each layer and also contributed to the layer-wise loss functions. The exploitation of sheaf geometry to understand the interconnectedness between oversmoothing and heterophily (Bodnar et al., 2022). One work also modeled the message-passing operation of GNN through the lens of energy (Di Giovanni et al., 2022b). (Rusch et al., 2022a) proposes a gradient-gating mechanism to enable the multi-rate flow of messages across the nodes. The gating mechanism of such methods reduces the effect of oversmoothing.

1.3.1.2 Normalization and Regularization-based Approaches

Alternatively, normalization-based techniques like PairNorm (Zhao and Akoglu, 2019), and DGN (Zhou et al., 2020) introduce additional normalization layers to distinguish the node features in deeper architectures. DropEdge (Rong et al., 2019), and AdaEdge (Chen et al., 2020a) reduce the rate of over-smoothing by learning to modify the underlying graph topology. On the other hand, Zhou *et al.* (Zhou et al., 2021b) designed Dirichlet energy-constrained graph neural networks, which control over-smoothing in deep models. Adaptive dropping of the edges (Hasanzadeh et al., 2020).

1.3.1.3 Residual Connection-based Approaches

Inspired by the ResNet (He et al., 2016a), various methods were proposed to involve residual connections in designing deeper convolutional architectures. Works like DeepGCN (Li et al., 2020a) implemented residual or skip connections across the hidden layers to tackle oversmoothing. Another notable work is GCNII (Chen et al., 2020c), where a scaled residual connection of initial node features is added to every hidden layer of the GCN. Another approac,h JKNet (Xu et al., 2018b), not only adds initial node features but also features of hidden layers to the final layer of the deep GNN. A major improvement is observed in DAGNN (Liu et al., 2020), which adapts the importance of node features that are aggregated in the final layer. Recently, (Scholkemper et al., 2024) proposed GraphNormv2, poised to mitigate oversmoothing and also reaffirmed by the extensive theoretical analysis.

1.3.1.4 Theoretical Analysis of Oversmoothing

(Cai and Wang, 2020a) discusses the theoretical concepts of over-smoothing in terms of Dirichlet energy. (Oono and Suzuki, 2019) theoretically shows that GNNs lose their expressive power with increasing network depths. (Wu et al., 2023b) offers theoretical analyses of the occurrence of the oversmoothing phenomenon in the deeper graph attention networks. Theory of graph smoothing (Keriven, 2022). Residual connection (Chen et al., 2025). Recently, one study (Epping et al., 2024) proposed the oversmoothing of GCN in terms of the Gaussian process (GP) and showed that initialization of weight matrices with higher variances often leads to resilience to oversmoothing in deeper layers.

1.4 Graph Heterophily

MPNNs are mostly designed to tackle graphs with a homophily assumption where adjacent nodes share similar node labels. Nodes with identical class labels are likely to form well-separated clusters due to the impact of message propagation. Yet, the success of message-passing may face obstacles if the adjacent nodes belong to different classes. In this scenario, recursive message passing does not respect the underlying cluster structures. This scenario is known as heterophily, and the corresponding graphs are called heterophilic. In Figure 1.3, we describe the structural characteristics of the heterophilic graphs,

Definition 1.3. *Node heterophily (Pei et al., 2020) The heterophily from the perspective of nodes is estimated as the ratio between the number of same-class neighbor nodes to all neighbors in the graph. Let us define* $\mathcal{H}_n$ *as*

$$\mathcal{H}_n = \frac{1}{|\mathcal{V}|} \sum_{v_i \in \mathcal{V}} \frac{|\{y_j | y_j \in \mathcal{N}_i, Y_j = Y_i|\}}{|\mathcal{N}_i|}. \tag{1.4}$$

Definition 1.4. *Edge heterophily (Zhu et al., 2020) The heterophily from the edge perspective is measured as the ratio between intra-class edges to the total number of edges in the graph. Let us define* $\mathcal{H}_e$ *as following,*

$$\mathcal{H}_e = \frac{|\{(u,v)|(u,v) \in \mathcal{E}, Y_u = Y_v\}|}{|\mathcal{E}|}. \tag{1.5}$$

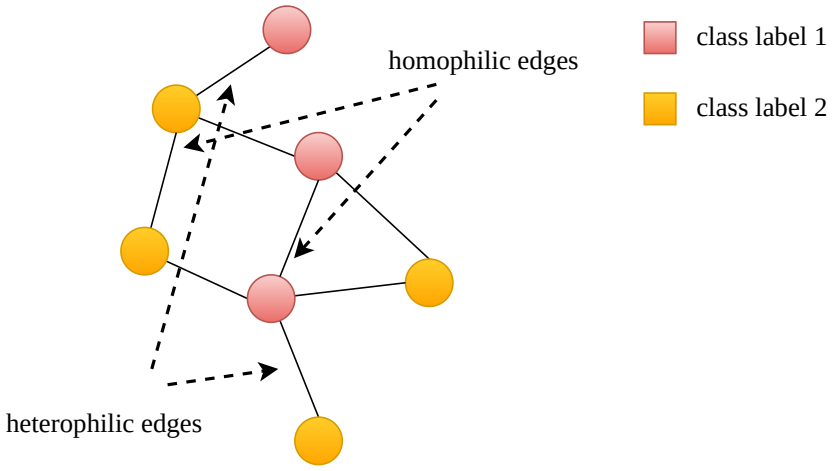


Figure 1.3: The example graph with shown with two different class labels. The corresponding homophilic and heterophilic edges are marked.

1.4.1 Methods to Address Graph Heterophily

The methods to address graph heterophily can be broadly classified into four categories as discussed below.

1.4.1.1 Spectral Graph Filters

The well-known architecture GCN employs a low-pass filter to smooth the features of the similar nodes. To extend the quality of the filters, GCNII (Chen et al., 2020c) is proposed that utilizes initial residual and identity mappings. GCNII exploits multi-hop information which seems to be beneficial for the heterophilic graphs. Beyond the fixed filters, several works like (Chien et al., 2020), (Page et al., 1999) designed variable filters able to capture high-frequency signals. Similarly, a non-deep method, ASGC (Chanpuriya and Musco, 2022) is also designed like GPR-GNN but in a much simpler fashion. Rather than employing a single filter, some works are dedicated to integrating multiple filters to design filterbanks. Prominent works like ACM-GNN (Luan et al., 2022) learns node-specific weights in the filterbank and adaptively uses node-wise localized information to address the graph's heterophily. Unifilter (Huang et al., 2024a) is devised to leverage universal polynomial filters to address different degrees of heterophily across the graph. FAGCN (Bo et al., 2021) proposed low-pass and high-pass filters to balance the information aggregated from homophilic and heterophilic neighbors. On

the other hand, NodeMoE (Han et al., 2024) endows the nodes with multiple experts to select the appropriate high-frequency or low-frequency signals.

1.4.1.2 Exploring Higher-order Neighbors

Another remarkable work, MixHop (Abu-El-Haija et al., 2019) performs long-range feature aggregation from higher-order neighbors by mixing the powers of adjacency matrices. H_2GCN (Zhu et al., 2020) attempts to separate the ego-node features and neighborhood representations. The method used feature concatenation to avoid unnecessarily mixing noisy messages from the neighbors and combined higher-order representations with a tactful design to operate on heterophilic networks. FSGNN (Maurya et al., 2021) selects optimal features generated from multi-hop message passing. The multi-hop neighbors can also be viewed through tree decomposition. TDGNN (Wang and Derr, 2021) leverages high-order neighbor interaction through tree decomposition. But that ignores the order of the tree structures. The problem is circumvented by proposing OrderedGNN (Song et al., 2023), which incorporates inductive bias onto the tree hierarchy and encodes the ordered information of the neighborhood. Apart from the tree-structure view, high-order neighborhoods can be encoded in the form of paths. Currently, methods like PathNet (Sun et al.) randomly sample paths. The paths are encoded while preserving the node ordering and distance information to interact with long-distance neighbors. Similarly, RAW-GNN (Jin et al., 2022) encodes random paths via a recurrent-based aggregator and reweights the importance of paths by computing attention scores between the paths.

1.4.1.3 Exploring Global Homophily

Exploring neighbors beyond local neighborhoods based on feature similarity is a potential avenue to tackle graph heterophily. Firstly, a kNN graph is constructed based on the cosine similarity between the node features. Works like Simp-GCN (Jin et al., 2021b) and UGCN (Jin et al., 2021a) adaptively integrate input graph adjacency and kNN feature graph. These methods show remarkable improvements on the heterophilic graphs. Despite the feature similarity, structural similarity comes into the landscape where potential neighbors are discovered based on similar structural patterns. In this category, methods like Geom-GCN (Pei et al., 2020) incorporate network geometry by mapping the nodes in the continuous latent space and

constructing edges as per the distances in the latent space. Another work, WRGNN (Suresh et al., 2021) computes the similarity between any two ego nodes by comparing ordered degree sequences of their high-order neighborhoods. Then, a multi-relational graph is constructed, and each edge is defined with its structural similarity. Computing global homophily may be computationally expensive. Nodes are mapped in latent space to learn the affinity among them. NLGNN (Liu et al., 2021) employs the GNN itself to learn node embeddings and GPNN (Yang et al., 2022c) applies pointer neural networks to evaluate the global affinity matrix. Later, both of them compute attention and perform long-range interactions. Apart from measuring global homophily, modeling the class compatibility matrix emerges as an effective approach. The compatibility matrix measures the probability of edges of nodes between the pair of classes. CPGNN (Zhu et al., 2021a) first estimates prior beliefs with the feature information, and then prior beliefs are propagated to learn the compatibility matrix, which eventually learns the likelihood of the connections of the nodes among the different classes. Another method, CLP (Zhong et al., 2022a), first learns the compatibility matrix and applies label propagation. Some methods use disentangling node features and graph adjacency to model global homophily. In this class, GloGNN (Li et al., 2022) utilizes MLPs to map the feature matrix and adjacency matrix into feature and topological views. Later, both of them are fused to learn a global affinity matrix. HOG-GCN (Wang et al., 2022b) employs feature information to determine the soft node labels propagated across the graph. In addition, BM-GCN (He et al., 2022) constructs a block similarity matrix representing the probability of having edges between nodes in the different blocks. The block similarity matrix attempts to explore block-guided neighbors and applies classified aggregation.

1.4.1.4 Discriminative Message Passing

Some studies show that edge directionality improves the performance of MPNNs on heterophilic graphs. AMUD (Sun et al., 2024) first identifies the importance of edge directions to understand heterophily from a statistical perspective. DirGNN (Rossi et al., 2024b) proposes separate aggregation functions for the directed edges and demonstrates remarkable improvements on the benchmarks. On the other hand, GGCN (Yan et al., 2022) assigns signed messages to differentiate between similar and dissimilar neighbors. Label-wise GCN (LW-GCN) (Dai et al., 2022) performs soft label-guided graph convolution to avert the effect

of dissimilar nodes in the heterophilic regime. The importance of inter-class or heterophilic edges may improve the performance of GNNs, which was explored by CAGNN (Chen et al., 2023a). Recently, Co-GNN (Finkelshtein et al., 2024) proposes an advanced gating mechanism to empower nodes with four states: Standard, Listen, Broadcast, and Isolate, allowing nodes to choose whether to receive or broadcast messages. This framework possesses greater expressive power of homophily and heterophily.

1.5 Oversquashing

Definition 1.5. *The oversquashing phenomenon was first observed by (Alon and Yahav, 2020) originated from GNNs leveraging long-range dependencies. Compressing aggregated features from a large number of neighbors into a fixed capacity of vectors results in information loss. The effect of oversquashing can be measured by the sensitivity bounds proposed by (Di Giovanni et al., 2023b; Topping et al., 2021a). assuming there exists a l-length path between u and v. The sensitivity bound is estimated as the Jacobian of $h_v^{(l)}$ and $h_u^{(0)}$ which can be represented as,*

$$\left\| \frac{\partial h_v^{(l)}}{\partial h_u^{(0)}} \right\|_1 \leq \underbrace{(cwp)^l}_{model} \underbrace{(\tilde{S}^l)_{uv}}_{topology}. \tag{1.6}$$

The effect of oversquashing depends on the c, the Lipschitz constant of the model parameters, w, the maximum entry of weight matrices, and the width p. Also, oversquashing is affected by the $\tilde{S}^l$ where $\tilde{S}$ can be a normalized adjacency matrix. The lower value of the bound signals diminished the effect on the message propagation of a node by its neighbors.

1.5.1 Methods to Address Oversquashing

Graph rewiring emerged as the most enticing solution to tackle oversquashing. The rewiring methods can be broadly classified into two ways as follows,

1.5.1.1 Spatial Rewiring Methods

Spatial rewiring methods typically deal with surgically modifying the graph structure by adding or removing edges. Alon and Yahav (Alon and Yahav, 2020) suggest converting the graph fully connected (FA) not only in the last layer but also for all intermediate layers.

This enables long-range interactions but also alters the graph structure. In a subsequent work, SDRF (Topping et al., 2021b) proves that oversquashing occurs due to the presence of negatively curved edges. They identify the edges and add edges to ensure the message propagation. GTR (Black et al., 2023a) greedily rewires the graph based on reducing effective resistance between the node pairs. BORF (Nguyen et al., 2023) explores the prowess of Ollivier-Ricci curvature flow to rewire the graph by maintaining the trade-off between oversmoothing and oversquashing. The computation of the edge curvatures for large-scale graphs may be expensive. Therefore, to mitigate the hurdle, AFRC (Fesser and Weber, 2024) is proposed, which computes curvatures in linear time. Apart from curvatures and effective resistance, DRew (Gutteridge et al., 2023) dynamically rewires the graphs with distance-aware skip connections. This method shows impressive improvements for solving long-range tasks on molecular graphs. A similar category of work (Chen et al., 2024a) proposed aggregates information and updates after some fixed delay, which also satisfies the asynchronous property. PR-MPNN (Qian et al., 2023) devised a rewiring strategy by harnessing differentiable k-subset sampling and rewiring the graph optimal for solving the target tasks. Recently, one work, GESN (Tortorella and Micheli, 2022) proposes a training-free GNN to mitigate oversquashing in heterophilic graphs. They suggest using reservoir computing models to avoid the extensive training.

1.5.1.2 Spectral Rewiring Methods

Spectral rewiring methods attempt to enhance the spectral gap of the network. The spectral gap is directly connected to the Cheeger constant of the network. FoSR (Karhadkar et al., 2022) adds edges which increase the spectral gap of the network, reaffirming well-connectedness. The rewiring also introduces the concept of relational rewiring by distinguishing between the pre-existing edges and newly added edges. Inspired by the flip Markov chain, RLEF (Banerjee et al., 2022) is poised to transform the given network into an Expander graph. The greedy version G-RLEF identifies hub edges and flips them such that the change in the number of triangles is minimized. This approach reduces the effective resistance. SJLR (Giraldo et al., 2023), which characterizes the exploration of edge-based curvatures. Another effective way to rewire is by converting the input into another graph by preserving the expander-type properties (Deac et al., 2022). Those methods are non-differentiable and

pursue sequential edge adjustments. A newer direction unraveled with DiffWire (Arnaiz-Rodríguez et al.) proposes an inductive differentiable rewiring strategy based on commute time, spectral gap, and effective resistance. Curvdrop (Liu et al., 2023) claims that positively curved edges are susceptible to oversmoothing and negatively curved edges are prone to oversquashing. They introduce a sampling layer guided by the Ricci curvature that drops or adds edges in the network topology. Another line of work, GOKU (Liang et al., 2025) rewires the graph topology by preserving the spectrum of the input graph.

1.5.1.3 Mixed methods

The spectral rewiring may improve the overall connectivity but may ignore the actual bottlenecks present in the graph. To mitigate the issue, LASER (Barbero et al., 2023) sequentially rewires the graph, improving spatial connectivity and respecting the sparsity and locality of the input graph. LASER enjoys the advantages of both spectral and spatial rewiring.

1.5.1.4 Approaches Beyond Rewiring

PASTEL (Sun et al., 2022) handles underreaching and oversquashing in the context of graph topology. They introduce two quantitative metrics to measure underreaching and oversquashing for further assessment. Some theoretical aspects are studied in (Di Giovanni et al., 2023b,c) to understand the impact of oversquashing on the expressive power of GNNs. DGN (Zhou et al., 2020) shows that the information flow along the direction of eigenvectors reduces the diffusion distance between node pairs and hence reduces overshooting. The most recent work PANDA (Choi et al., 2024a) enhances the channel capacity of the nodes with higher importance to process a larger volume of information effectively to prevent oversquashing.

1.6 Graph Transformer: A Global Attention Perspective

Inspired by Vashwani *et al.* (Vaswani et al., 2017), Dwivedi, and Bresson (Dwivedi and Bresson, 2020) extended the philosophy of Transformer for the graph-structured data. Message-passing Graph Neural Networks (MP-GNNs) implement sparse message passing where GTs assume the fully connected graph structure. Unlike MP-GNNs, Graph Transformers are designed to compute the attention coefficient between the pairs of nodes without considering the

graph structure. A single Transformer layer consists of a self-attention module followed by a feed-forward network. The feature matrix X is transformed into query Q, key K, and values V by multiplying with the projection matrices as $Q = XW_Q$, $K = XW_K$, and $V = XW_V$. The self-attention matrix is computed as follows,

$$X_A = \text{softmax}(\frac{QK^T}{\sqrt{d_{out}}})V, \tag{1.7}$$

where $X_A \in \mathbb{R}^{n \times d_{out}}$ is the self-attention matrix with d_{out} is the output dimension. W_Q, W_K, and W_V are trainable parameters. To boost the impact of the self-attention module, we often employ a multi-head attention (MHA) strategy. The multi-head attention is the result of the concatenation of multiple instances of Eq. 6.1. The multi-head attention can be expressed as:

$$X_A = M \Big\|_{k=1}^{H} \left(\sum_{j \in N(i)} \alpha_{ij}^{(k)} V^{(k)} X_j \right), \tag{1.8}$$

where $\alpha_{ij}^{(k)}$ is the attention coefficient between node i and j from k^{th} head. M and $V^{(k)}$ are the trainable parameters and X_j is the j^{th} node features. The estimated self-attention matrix is followed by a residual connection and then passes through a feed-forward network (FFN) with the normalization layers as follows,

$$\begin{aligned} X' &= \text{Norm}(X + X_A), \\ X'' &= W_2(\text{ReLU}(W_1 X')), \\ X_o &= \text{Norm}(X' + X''), \end{aligned} \tag{1.9}$$

where X_o is the final output of the transformer layer and $W_1 \in \mathbb{R}^{d_{out} \times d}$, $W_2 \in \mathbb{R}^{d \times d}$ are trainable parameters. We can either use Batchnorm (Ioffe and Szegedy, 2015) or Layernorm (Ba et al., 2016) for the feature normalization. Each Transformer layer generates node-level representations, which are permutation equivariant. The absence of positional information on the nodes generates similar outputs. Therefore, it is necessary to incorporate appropriate positional encodings to leverage the learning process in the Transformer.

1.6.1 Positional Encodings for Graph Transformers

The positional encodings serve as the necessary positional encodings provided in the initial node features. The following equation represents the usage of PEs in GT.

$$X \leftarrow X + PE \tag{1.10}$$

In the literature of GT, several novel positional encodings are proposed. The journey begins with proposing eigenvectors of the normalized graph Laplacian (Dwivedi and Bresson, 2020) for each node. Let the i^{th} eigenvector $U_i = [U_{i1}U_{i2}\cdots U_{ik}]$ where $k \leq n$, where k is the dimension up to which we consider the dimension of encodings. As LapPE captures the spectral characteristics of the graphs, but not any structural characteristics. To bridge the gap, (Dwivedi et al., 2021) proposes Random Walk Positional Encoding (RWPE), which constitutes diagonal elements of the higher-order diffusion matrix like AD^{-1}. For i^{th} node, the RWPE is constructed as $[I, AD^{-1}_{ii}, (AD^{-1})^2_{ii}, (AD^{-1})^3_{ii}, \cdots, (AD^{-1})^{k-1}_{ii}]$. Graphormer (Ying et al., 2021b) incorporates degree centrality encodings into the initial node features. Alongside, they further add edge encodings and encodings for shortest path distance (SPD) during the self-attention computation. Structure-aware Transformer (SAT) (Chen et al., 2022a) learns the efficient representation of the rooted subgraph or subtree of each node and feeds the encodings to the initial node features. Another work, the Spectral Attention Network (SAN) (Kreuzer et al., 2021), learns node-wise positional encodings by combining eigenvalues and eigenvectors that pass through the Transformer encoder. Subsequently, the learned encodings are converted into spectral-aware weights that act as the attention during the feature update. Another line of work modifies RWPE to design relative positional encodings, Relative Random Walk Encodings, or RRWP. For each node pair (I, j) the RRWP is defined as $[I, AD^{-1}_{ij}, (AD^{-1})^2_{ij}, (AD^{-1})^3_{ij}, \cdots, (AD^{-1})^{k-1}_{ij}]$. Based on these PEs, the authors proposed a novel architecture named GRIT shows remarkable performances over SOTA methods. They also suggest injecting degree information into every hidden layer. On this connection, GraphGPS (Rampášek et al., 2022) combines MPNN and self-attention mechanisms to design a scalable Graph Transformer.

Apart from designing novel positional encodings, few works are focused on designing stable PEs along with maintaining the property of invariance, symmetry, etc. First of its kind

work like PEG (Wang et al., 2022a) design PEs that update separate channels to update node features and positional features. Node features maintain permutation equivariance, and positional features show rotational and translational equivariance. In a subsequent work, two neural architectures are introduced: BasisNet and SignNet (Lim et al., 2023). Those architectures are invariant to symmetries of sign flips and basis symmetries. In this vein, another notable work (Huang et al., 2024b) identifies the flaw of standard Laplacian eigenvectors and their weaker stability. They found that the hard partition of the eigenspace is highly sensitive to external perturbations. They proposed Stable Positional Encodings or SPE to tackle the underlying issues. Those PEs impart soft partitions of the eigenspace.

Laplacian eigenvectors are widely used as positional encodings, but for large-scale graphs, this will lead to costly eigendecomposition. To avert the glaring shortcomings, PEARL (Kanatsoulis et al., 2025), a novel learnable positional encoding, can be flexibly employed in both GNN and GT. PEARL offers linear time complexity with higher expressive powers and stability. After tackling the problems in large-scale graphs, the design scheme for positional encodings for directed graphs remains obscure. The gap is bridged by the work (Huang et al., 2025). Recently, one framework, Random Feature Propagation (RFP) (Eliasof et al., 2023) initializes PEs as random vectors. Encouraged by the power iteration method, the random vectors are gradually converted to the most dominant eigenvectors.

1.7 Motivation and Objective

At this stage, we have broadly discussed the importance of long-range message passing for the performance improvements in graph neural networks. We extend our discussion by identifying various challenges, including Oversmoothing, Oversquashing, and Heterophily, that persist in multi-layered GNN architectures. A closer scrutiny reveals the interconnectedness of the aforementioned challenges. In this thesis, we primarily focus on the following issues pertaining to the long-range interactions in GNNs.

1. Designing deep GNNs enables the aggregation of multi-hop information from long-distance nodes. Though the iterative message passing causes the oversmoothing of the node features. Various methods are methods to address oversmoothing, but they still fail to explore multiple distant neighbors simultaneously. Most of the methods were

structural modifications (Xu et al., 2018b; Chen et al., 2020c) or additional normalization layers (Zhao and Akoglu, 2019; Zhou et al., 2020). Very few works like (Jin et al., 2022; Liu et al., 2019b) explore multi-hop neighbors via random path sampling. Still, those works offer no solution to oversmoothing. Designing an effective message passing strategy by sampling multi-hop neighbors is a prominent gap that needs to be addressed.

2. Exploring long-distance neighbors can be further complicated if the input graph contains a higher number of heterophilic edges (Zhu et al., 2020; Pei et al., 2020). Those are responsible for the mixing of noisy information from different class labels and resulting in the exacerbating performance of multi-layered GNNs. Various message passing strategies (He et al., 2022; Wang et al., 2022b) are designed to tackle graph heterophily. Perhaps, altering the edge connectivity according to similar node labels may enhance the quality of the aggregated information. In this context, the rewiring techniques are overlooked to address the graph heterophily by tampering with edge connectivity. Therefore, graph rewiring is not fully harnessed in the heterophilic paradigm.

3. Another facet of deep GNNs is the problem of Oversquashing (Alon and Yahav, 2020), an information bottleneck, stemming from the need to compress information from exponentially growing receptive fields into the fixed-dimensional vectors. Mostly, spectral and spatial graph rewiring (Karhadkar et al., 2022; Black et al., 2023a; Topping et al., 2021b) emerges as effective avenues for tackling oversquashing. Altering the graph structure might cause information loss related to the inductive bias of the graph. However, one work (Choi et al., 2024a) increases the width of the features for the bottleneck nodes, mostly those with high centrality values, which does not involve rewiring. Therefore, going beyond graph rewiring should be a promising alternative for addressing oversquashing in GNNs.

4. The long-range interactions can be performed without stacking multiple message passing layers by employing Graph Transformers (Dwivedi and Bresson, 2020). GTs are capable of estimating pair-wise attention or global attention, dismissing the requirement to add additional message passing layers. The performance of GTa heavily depends on the positional encodings (Dwivedi et al., 2021; Chen et al., 2022a). Most of the designed

positional encodings are designed in Euclidean space, unable to capture the hierarchical properties of the underlying graph data. Thus, we need to design positional encodings in non-Euclidean space, or specifically, in hyperbolic spaces.

Motivated by these shortcomings, we list our primary objective in this thesis as follows:

1. Propose a non-recursive message passing algorithm that relies on random path sampling and multi-hop node features to mitigate the effect of oversmoothing. The sampling method should be dynamic and cost-effective rather than being performed in the pre-processing stage.

2. Design a graph rewiring framework to alter edge connectivity that will provide a remedy to graph heterophily. Furthermore, the cause of performance degradation of low-pass filters like GCN on the heterophilic networks requires a deeper scrutiny.

3. Propose a model-agnostic framework to address oversquashing that will go beyond rewiring the graph topology.

4. Design a learnable positional encoding in the hyperbolic space with negative curvature to capture hierarchical components in the input graphs. The positional encodings should be expressive and powerful to represent the nodes in the graph.

1.8 Organisation and chapter-wise contributions

In this thesis, we identified the shortcomings of shallow graph neural networks that aggregate from the local neighborhoods. We further enriched the discussion with the advantages of long-range interaction. In the introductory chapter, we highlighted the challenges that can be confronted when multiple message passing layers are stacked. Notably, GNNs mostly suffer from Oversmoothing, Oversquashing, and Heterophily when deep GNNs are constructed. In chapter 2, we address the effects of Oversmoothing. In chapter 3, we tackle the heterophily issue via graph rewiring. In the following chapter 4, we analyze the effects of graph heterophily on the performance of low-pass filters like GCN. In the chapter 5, we design an asynchronous message passing framework to address oversquashing. In chapter 6, we propose a learnable hyperbolic positional encoding to enhance the performance of Graph Transformers. In the

final chapter 7, we mentioned the future possibilities of our current approaches and listed some open problems in designing the future contours of graph representation learning. We provide an illustrative roadmap of the thesis in Figure 1.4.

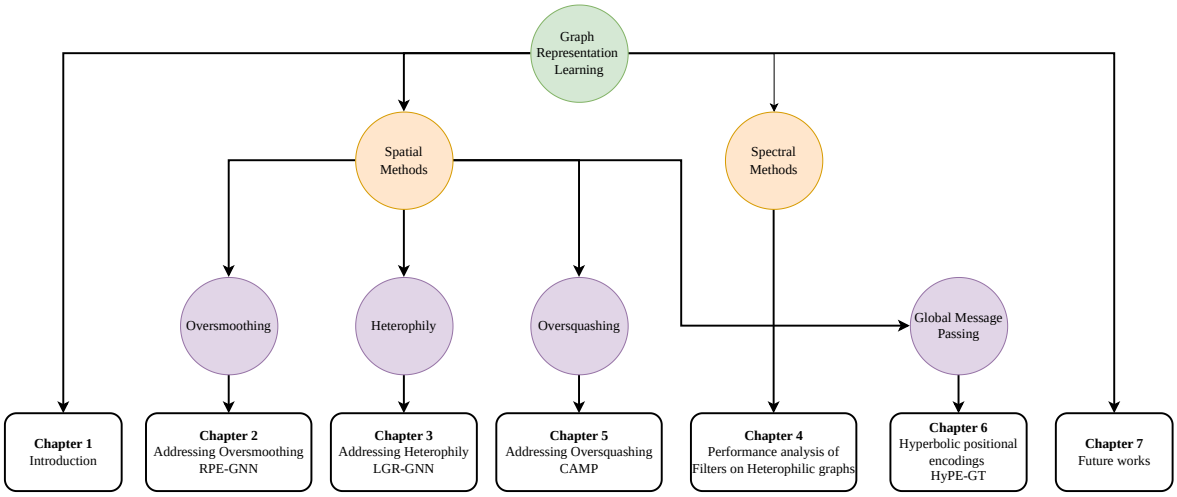


Figure 1.4: The road-map of the thesis.

1.8.1 Contributions of Chapter 2

In this chapter, we identified one of the core reasons for Oversmoothing as the recursive message passing. We analyze the neighborhood aggregation mechanism through the lens of computation graphs. The aggregated messages flow from higher-order neighborhoods to lower-order neighborhoods, which is equivalent to the bottom-to-top message propagation in the computation graphs. The computation graphs grow exponentially when multi-hop neighbors are explored, and the tree height also increases. Such computation graphs contain repeated substructures, leading to the computation of similar node representations due to the recursive nature of message passing. As a solution, we pursue the design of a non-recursive aggregation strategy to mitigate the oversmoothing issue. Our approach randomly samples paths and learns the individual edge features. Subsequently, the edge features are concatenated to generate path features. We only aggregate the features of the nodes when we reach the desired number of hops. The estimated path features act as gates that control the information flow, resulting in the distinct node representations. Our method aggregates messages from lower to higher order neighborhoods, departing from traditional aggregation

mechanisms. Importantly, we only aggregate once, breaking the rule of recursive message passing strategies. We also establish connections between the lazy random walk and our proposed method. Several experiments are performed to validate the efficacy of the proposed method.

1.8.2 Contributions of Chapter 3

In this chapter, we propose a novel method that tackles graph heterophily, which is a situation where connected nodes may share different class labels. Traditional message-passing graph neural networks aggregate the messages from local neighborhoods. Those classes of architectures show promising results on homophilic graphs where adjacent nodes share identical class labels. The higher node feature similarity primarily indicates that connected nodes might share identical class labels. We investigate that standard similarity measures like cosine similarity may not sufficiently capture the intra-class and inter-class similarity. We propose a solution to incorporate the class information into the features by learning auto-encoder-based class embeddings. We reweight the feature similarities with the scores obtained from class representations. The updated scores are utilized to rewire the graph structure, converting it into homophily-prone neighborhoods. The rewiring of the edge connections ensures the diminishing of heterophilic connections and improves the homophily ratio. Our approach is model-agnostic and can be applied to any sort of heterophilic graph. We also offer theoretical justifications of the working mechanism of our methodology. Extensive experiments are carried out on several heterophilic benchmarks to validate the efficacy of our work.

1.8.3 Contributions of Chapter 4

In this chapter, we investigate the utility of graph rewiring to unveil the spectral properties of the graph. We leverage the addition of a pre-determined number of self-loops and parallel edges in the random graphs. The experiment reveals that the addition of self-loops and parallel edges increases and decreases the eigenvalues of the graph Laplacian. To support the outcome, we provide extensive theoretical underpinnings in this work. Consequently, the aforementioned edge additions shift the distribution of the graph spectrum. Furthermore, we empirically observed various performance trends when self-loops or parallel edges are incorpo-

rated in the heterophilic graphs. The gradual increase in the number og self-loops or parallel edges generates different performance trends on the heterophilic graphs. We categorize the heterophilic networks into some pre-defined categories based on the observed performance trends. Furthermore, we establish the connection between performance trends and spectrum shifts. The categorization of graphs entails the spectral properties of the graphs, like parity of lower and higher eigenvalues, presence of weakly connected components. This information may also be evaluated by performing a computationally expensive eigendecomposition algorithm on the graph Laplacian. This seems to be a potential hindrance for real-world large-scale graphs. We offer an innovative pathway to avoid such expensive algorithms to gain insights into the graph spectrum. Our work also bridges the gap between the performance analysis of low-pass filters and structural patterns in the ambit of a heterophilic regime. Our extensive experimental analysis on widely available 17 heterophilic benchmarks validates our claim.

1.8.4 Contributions of Chapter 5

In this chapter, we laid the foundation for asynchronous message passing frameworks, marking a notable departure from the prevailing standard synchronous MP-GNNs. In the beginning, we create mini-batches of nodes selected based on some pre-defined criteria. We update the features of nodes belonging to the mini-batches, and the features of unselected nodes remain unaltered. Our primary target is to alleviate Oversquashing that occurs when information is attempted to be stored in fixed-sized vectors, emanating from exponentially growing neighborhoods. Our proposed framework accesses the channels with fixed capacity at a regular interval of times. This strategy prevents storing a larger volume of information in the channels with constrained capacity, mitigating the effects of oversquashing. In this context, we consider node centrality as our criterion to create node batches. Higher centrality nodes allow a high volume of information flow, acting as key candidates for information bottlenecks. We strategically allow interaction of high centrality nodes to access hidden channels, which eventually leads to the mitigation of oversquashing. We conducted several experiments on several benchmarks and long-range datasets and obtained remarkable improvements on several state-of-the-art methods.

1.8.5 Contributions of Chapter 6

In this chapter, we designed a novel and effective positional encoding for Graph Transformers. Our proposed positional encodings are learnable and embedded in the hyperbolic space, a non-Euclidean space with negative curvature. These two key characteristics are instrumental for enhancing the expressive power of Graph Transformers. We also offer extensive theoretical insights to validate the efficacy of our approach. Pursuing hyperbolic embeddings largely increases the parameter complexity of the underlying model. In this work, we offer an effective pathway to implement hyperbolic embeddings, resolving the issue of parameter complexity. Our proposed positional encodings are predominantly applicable to the molecular graphs that contain hierarchical structures. In this scenario, the hyperbolic positional encodings capture pertinent structural and positional information from the hierarchies present in the datasets. Furthermore, we incorporate hyperbolic positional encodings in the multi-layered GNNs and observe the impressive improvements in the performance. This approach offers a unique pathway to mitigate oversmoothing. Extensive experiments were conducted to establish the efficacy of our proposed learnable hyperbolic positional encodings.

1.8.6 Unifying towards Robust Designs of Graph Neural Networks

In this thesis, we have proposed RPE-GNN, LGR-GNN, CAMP, and HyPE. The designed methods resolve the problem of oversmoothing, heterophily, and oversquashing that primarily occurs in GNNs. Furthermore, HyPE is designed to capture the hierarchical characteristics of the graphs. In this connection, the vanilla Graph Transformers are poised to interact with every pair of nodes, resulting in a complete graph structure. Though pair-wise interaction will increase the computational cost. Employing the sampling scheme of RPE-GNN filters out important neighbors and converts the graph into a more sparse one. Subsequently, we can apply the LGR rewiring framework to increase the homophilic edges. After the rewiring process, the feature updates of nodes can be executed asynchronously based on the CAMP algorithm. The scaled dot-product attention estimation remains unaltered, which is applied across the transformed graphs. Moreover, hyperbolic positional encodings assists to encode the hierarchical information into the node features. The unification of the four methods asserts the individual prowess ans leading to a design of robust graph models.

1.8.7 Contributions of Chapter 7

In this chapter, we discuss the future extensions of our current research work and list some open problems as potential avenues of research.

Chapter 2

Addressing Oversmoothing via Randomized Path Exploration

Summary

Graph Neural Networks (GNNs) have emerged as one of the most powerful approaches for learning on graph-structured data, even though they are mostly restricted to being shallow in nature. This is because node features tend to become indistinguishable when multiple layers are stacked together. This phenomenon is known as over-smoothing. This paper identifies two core properties of the aggregation approaches that may act as primary causes for over-smoothing. These properties are, namely, recursiveness and aggregation from higher to lower-order neighborhoods. Thus, we attempt to address the over-smoothing issue by proposing a novel aggregation strategy that is orthogonal to the other existing approaches. In essence, the proposed aggregation strategy combines features from lower to higher-order neighborhoods in a non-recursive way by employing a randomized path exploration approach. The efficacy of our aggregation method is verified through an extensive comparative study on the benchmark datasets w.r.t. the state-of-the-art techniques on semi-supervised and fully-supervised learning tasks.

2.1 Introduction

Amidst the rise of Geometric Deep Learning (Masci et al., 2016; Bronstein et al., 2017), it is no wonder that the graph structure draws adequate attention for analyzing data beyond the conventional Euclidean domain. The milestone work on Graph Neural Networks (GNNs)

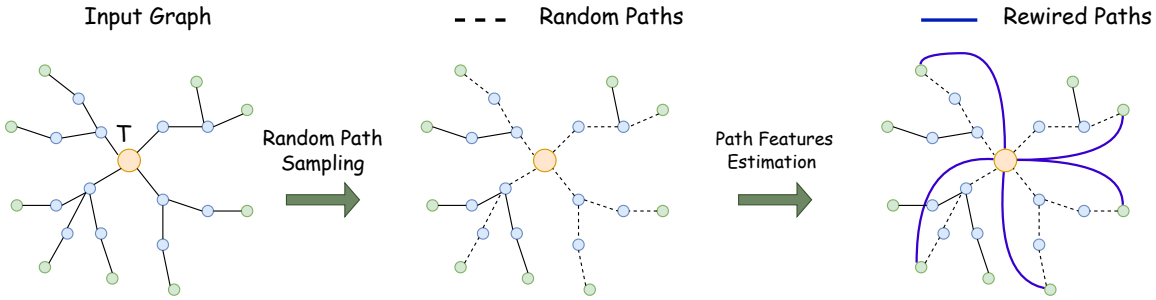


Figure 2.1: The overview of RPE-GNN is represented. The center node T, whose embedding would be evaluated via 3-hop neighborhood aggregation. Initially, random paths are generated gradually by layer-wise node sampling. The selected paths are marked with dotted lines. The path features are evaluated dynamically. Solid blue lines denote the estimated path features. The graph is rewired by adding edges to replace the sampled paths. The edges are incorporated with the respective path features. The newly added edges act as direct connections between the target node and its multi-hop neighbors. The neighborhood aggregation is performed on the newly rewired graph.

(Scarselli et al., 2008) is crowned as one of the go-to tools to facilitate learning on graph-structured data. Consequently, application of deep learning techniques on graphs significantly enhances the performance of various downstream learning tasks such as node classification (Yang et al., 2016; Kipf and Welling, 2016), link prediction (Bojchevski et al., 2018; Grover and Leskovec, 2016), graph classification (Duvenaud et al., 2015; Zhang et al., 2018; Gilmer et al., 2017a), and biological networks (Pavlopoulos et al., 2011; Koutrouli et al., 2020).

However, the commendable performance of GNNs may be considered restrictive to their full potential, given the shallower architecture. Even though models like GCN (Kipf and Welling, 2016), GAT (Veličković et al., 2017), and GraphSage (Hamilton et al., 2017) exhibit impressive performance by using a limited number of hidden layers, they are still not capable of learning from the higher-order neighborhoods. One way to alleviate these restrictions is to aggregate features from the multi-hop neighbors using a deeper architecture.

Unfortunately, stacking multiple layers similar to Convolutional Neural Networks (CNNs), such as ResNet (He et al., 2016b) in GNN, is not so straightforward. This is because the increased number of successive feature aggregations in a deeper architecture may make the combined node features almost indistinguishable. This phenomenon is known as *Over-smoothing* (Li et al., 2018), which has been shown to degrade the performance of node classification tasks as well as the expressive power of the GNN models (Oono and Suzuki, 2019).

The features of the nodes belonging to a connected component become indistinguishable due to the effect of over-smoothing. The primary reason behind the occurrence of over-smoothing is the overlapping sets of neighbors of the concerned nodes. The overlap among the sets of neighbors gradually increases when the size of the neighborhood grows. The overlapping set of neighbors is not only responsible for the over-smoothing, but the aggregation mode also plays a critical role in the feature over-smoothing. The well-known architectures like Graph Convolutional Networks (GCN) (Kipf and Welling, 2016), Graph Attention Networks (GAT) (Veličković et al., 2017), GraphSage (Hamilton et al., 2017), APPNP (Klicpera et al., 2018), Graph Diffusion Convolution(GDC) (Klicpera et al., 2019), etc. aggregate node features from higher to lower order neighborhoods in a recursive manner as depicted in the Figure 2.2. The redundancy involved in the recursive higher-to-lower order computation may lead to repeated consideration of similar neighboring nodes during the feature calculation. This redundancy acts as a primary reason for the over-smoothing. Despite having the problem of over-smoothing, deep models offer a significant advantage over shallow models as the former can learn from the multi-hop neighborhood. Therefore, the drawbacks mentioned earlier and the necessity of deep models motivate us to propose a novel aggregation technique that prevents over-smoothing.

Numerous attempts like DeepGCN (Li et al., 2019), JKNet (Xu et al., 2018b), PairNorm (Zhao and Akoglu, 2019), DGN (Zhou et al., 2020), DropEdge (Rong et al., 2019), AdaEdge (Chen et al., 2020a) etc. were made to mitigate the issue of over-smoothing, but the aforesaid approaches are either different variants of GCN, GAT, or GraphSage or the combinations of the existing architectures where underlying aggregation strategy remains unaltered. The question of tackling over-smoothing by improving the existing neighborhood aggregation techniques still needs to be answered. Motivated by those facts, we attempt to solve the issue by introducing a novel architecture. The architecture enables a message-passing strategy which is entirely different from existing message passing frameworks. The proposed approach consists of two steps: initially, the fixed-length random paths are sampled for a source node to explore its multi-hop neighbors. In the second stage, the path features are estimated by using some pre-defined rule. The sampled paths can be assumed as the connectors or edges between source nodes and randomly selected neighboring nodes. The graph is rewired by using those newly constructed edges, and edges are equipped with the features as corresponding estimated

path features. After that, the message aggregation is performed on the source node by considering the newly rewired graph. The overview of the proposed algorithm is presented in Figure 2.1.

When the number of random paths is increased the computational burden also enhances. To avoid the overhead we propose a dynamic path feature computation technique. The dynamic method allows the model not to store the predetermined random paths as a pre-processing step. The paths are gradually traced out while propagating through the hidden layers. The path features are also estimated alongside. The approach also explores better graph topology.

The aggregation strategy of RPE-GNN is also unique from other aggregation strategies. The existing aggregation techniques updates node features in every hidden layer, whereas we update node features only once which is after the complete estimation of the path features. Specifically, in our case, the hidden layers are utilized to compute path features rather than used for aggregation. In this way, our proposed approach can avoid redundant calculation of neighbors' features, effectively tackling the over-smoothing issue. Finally, we can summarize our contributions as follows,

- Unlike (Kipf and Welling, 2016; Veličković et al., 2017; Hamilton et al., 2017; Xu et al., 2018b), we design a novel strategy that aggregates from lower to higher-order neighborhoods in a non-recursive manner.
- Our approach learns node representations by estimating the features of the randomly explored paths in the graph, which is described in section 6.3. Therefore, we termed it as Randomized Path Explorer-Graph Neural Networks or *RPE-GNN*.
- We design an efficient technique where paths are traced out gradually while propagating through the hidden layers without having prior knowledge of the predetermined paths.
- In section 2.2.3 we show that our method simulates a biased random walk and the over-smoothing issue can be prevented by controlling the bias of the walk.
- We perform diversified experiments to showcase the efficacy of our framework in section 6.4, and it outperforms existing state-of-the-art approaches.

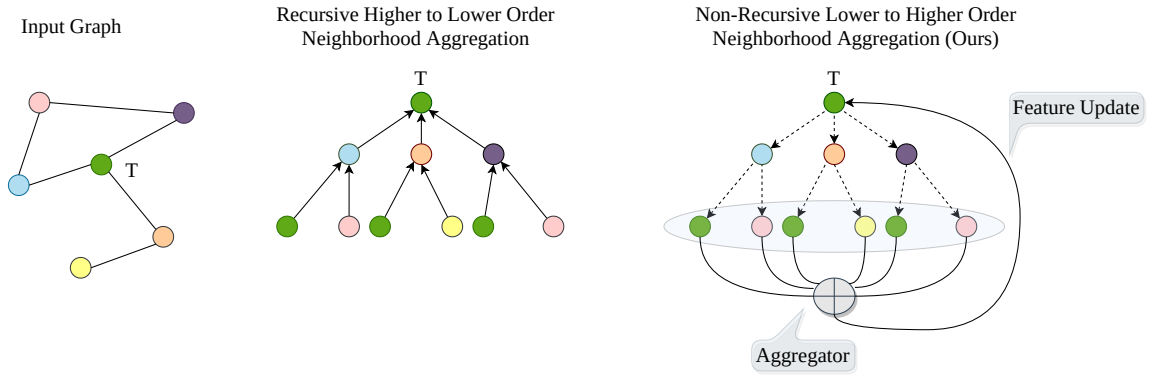


Figure 2.2: Comparative study between the proposed aggregation method and existing aggregation methods is demonstrated. The 2-hop neighborhood aggregation is performed on the node T of the input graph(leftmost). In the case of the recursive aggregation(middle one), the aggregation is performed from higher to lower-order neighborhoods (bold lines with arrows). In this case, the information flow from the leaf nodes to the root node of the computation graph. On the other hand, the non-recursive aggregation method(rightmost) aggregates from lower to higher-order neighborhoods. The non-recursive method initially estimates the path features, and at the final stage, the aggregation is performed with the distant neighbors by employing those path features. Here, the information flows from the root node to the leaf nodes of the computation graph(dotted lines with arrows).

2.2 Proposed Method

2.2.1 Preliminaries

A graph $G = (V, E)$ consists of a set of n nodes V and a set of edges $E \subseteq (V \times V)$. The connections between nodes are represented by an adjacency matrix $A = [a_{ij}]_{n \times n}$ where $a_{ij} \in \{0, 1\}$. Evidently, $a_{ij} = 1$ indicates that the i^{th} and j^{th} nodes are connected by an edge, while $a_{ij} = 0$ denotes otherwise. The feature matrix of G is denoted by $X \in \mathcal{R}^{n \times f}$ where the i^{th} row x_i denotes the f-dimensional feature representation of the i^{th} node. The degree matrix D is a diagonal matrix where the i^{th} element in the diagonal corresponds to the degree of the i^{th} node. The graph Laplacian is defined as $L = D - A$. The symmetrically normalized graph Laplacian is represented as $N = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$.

2.2.2 Randomized Path Exploration

The core idea of our approach is randomly exploring multiple paths in the graph to aggregate multi-hop information. Simultaneously, we also learn features of the corresponding paths via propagating through the hidden layers. The path features are eventually used to evaluate

the node embeddings. The node representations are updated only after the estimation of the path features.

Assume t is the node whose embedding would be evaluated using the k-hop distant neighbors. Consider $N_t^k = \{n_1, n_2, \cdots, n_l\}$ as the set containing all k-hop neighbors. The complete embedding method will be described in a top-down fashion to evaluate the embedding of t.

Path Feature Estimation A path in the graph comprises a sequence of nodes contained in the graph. Assume a k-length path starting from t is denoted as $P_t = \{t = n_0, n_1 \cdots n_k = d\}$. A feature of a path is the sum of the features estimated over the individual edges of the path. The features of a single edge of a path can be estimated as below:

$$\mathbf{M}_{pq} = \sigma((x_p - x_q)W), \tag{2.1}$$

where p and q are the tail and head of the edge. $\mathbf{M}_{pq}$ is the feature of the corresponding edge e_{pq} and $W \in \mathbb{R}^{f \times f'}$ is the trainable weight matrix where f' denotes dimension of newly transformed space. The weight W is shared across all edges in the graph, which makes our architecture parameter-efficient. $\sigma(.)$ represents the non-linear activation function. x_p and x_q are the features associated with the p^{th} and q^{th} node respectively.

As previously mentioned, a path's feature is evaluated as the sum of the features of the individual edges of the respective path. Then the path P_t will have a message as shown below:

$$\mathbf{M}_P = \phi(\sum_{i=0}^{k-1} \mathbf{M}_{n_i n_{i+1}}), \tag{2.2}$$

where $\mathbf{M}_{n_i n_{i+1}}$ is the features of the edge $e_{n_i n_{i+1}}$ and $\phi(.)$ is a trainable function. Specifically, the feature of a path is the sole representation within the graph. The learned features of the paths contain information among the non-adjacent nodes.

Aggregation The estimated path features will be employed to update the node embeddings. The path features encode the node sequences of the respective paths between the target nodes and the multi-hop neighbors. One can redefine path features as the features of the connecting edges between the pair of non-adjacent nodes. Our approach gathers information from the sequence of edges to generate complete path features rather than passing information to distant neighbors like in graph diffusion (Klicpera et al., 2019). Consider the set of all possible

k-length paths starting from t is denoted as $\mathcal{P}$. Therefore, the neighborhood aggregation is formulated as follows:

$$\tilde{x_t} = x_t + \frac{1}{N(\mathcal{P})} \sum_{p \in P, l \in N_t^k} \phi(\mathbf{M}_p) \odot x_l, \tag{2.3}$$

where $\tilde{x_t}$ denotes updated node embedding, $\mathbf{M}_p$ is the estimated features of the path $p \in \mathcal{P}$, x_l is the feature of the $n_l \in N_t^k$ which is the tail node of the path p. $\odot$ denotes the element-wise multiplication between the vectors. $N(\mathcal{P})$ is the number of elements of the set $\mathcal{P}$. The Equation 2.3 aggregates information from the multi-hop node l, which is also a tail node of path p. The equation also acts as the message propagation rule of our proposed approach.

Dynamic Feature Computation In Equation 2.3 the set $\mathcal{P}$ contains all k length paths begin from node t. The number of paths will increase exponentially when k increases, and this will generate a massive computational cost. To prevent the issue, we devise a technique that computes path features dynamically by defining a recurrence relation. Assume a k-length path p whose node sequences are represented as $\{t = n_0, n_1, \cdots, n_k = d\}$. The feature of the path $\mathbf{M}_p$ or $\mathbf{M}_{n_0 n_k}$ is estimated dynamically in the following way:

$$\mathbf{M}_{n_0 n_{l+1}}^{(l+1)} = \mathbf{M}_{n_0 n_l}^{(l)} + \mathbf{M}_{n_l n_{l+1}}^{(l+1)}, \tag{2.4}$$

where l lies in $[1, (k-1)]$ and $\mathbf{M}_{n_0 n_{l+1}}^{(l+1)}$ denotes estimated path features node n_0 to n_{l+1} at the $(l+1)^{th}$ layer. Similarly, $\mathbf{M}_{n_0 n_l}^{(l)}$ indicates the computed features from n_0 to n_l at the l^{th} layer. Again the features of the $(l+1)^{th}$ edge is $\mathbf{M}_{n_l n_{l+1}}^{(l+1)}$ which is estimated by using equation 2.1 as $\mathbf{M}_{n_l n_{l+1}}^{(l+1)} = \sigma((x_{n_l} - x_{n_{l+1}})W^{(l)})$ where $W^{(l)}$ denotes the trainable weight matrix of the l^{th} layer.

The recurrence relation reduces the massive computational overhead during path feature estimation. Prior knowledge of the paths is not required as they would be gradually traced out while propagating through the hidden layers. Therefore, the features of the k-length paths are estimated by consecutively adding the features of k edges. As depicted in Figure 3.2, at l^{th} layer, we have already computed up to l-length features of a k-length path where $l \leq k-1$. Then at the $(l+1)^{th}$ layer we will only estimate the features of the $(l+1)^{th}$ edge of the corresponding path. The features of the $(l+1)^{th}$ edge will be added with the previously computed l-length features to evaluate the $(l+1)$-length path features. Hence, the feature

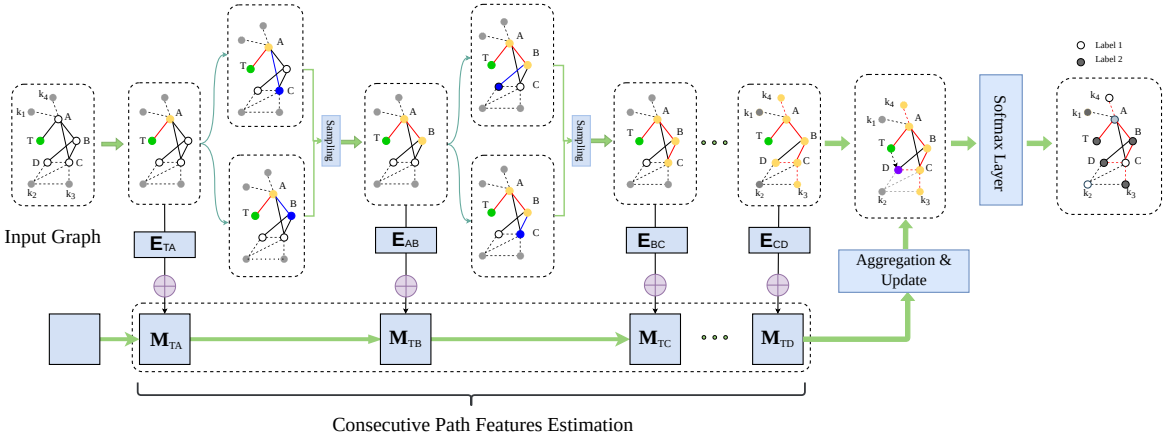


Figure 2.3: The workflow of our proposed aggregation scheme is demonstrated. The node T (in green) is the target node whose embedding would be evaluated. The bold (black) lines denote edges between pairs of nodes, where the dotted lines (black) in the input graph show the existence of multiple intermediate nodes between any pairs of nodes. At each step, a neighbor (marked blue) of the current node (marked yellow) is randomly selected, and the features of the newly selected edge (bold red lines) are calculated. The current edge features are added with the total estimated features. The procedure is continued until the required length of path features is evaluated. The edges with the red color denote the complete traced out path by the random walk. The remaining grey nodes are denoted as unvisited nodes. After the path feature estimation, the aggregation step is performed with the distant neighbor (marked purple), and features of T are updated. The path features are denoted by the dotted line with an arrow directly connecting from T to D. After passing through the softmax layer, the final nodes are classified into the desired number of classes.

computation is implemented by considering only the immediate estimated total features and the current edge features. Therefore, this dynamic approach prevents the algorithm from the unnecessary requirements of the predetermined random paths. The node features are updated only after evaluating the desired length of path features. No update is performed between any two hidden layers like the other prevalent aggregation strategies such as GCN, GAT, GraphSage, JKNet, GCNII, etc. This approach helps the model avoid redundant feature computation during the aggregation procedure.

2.2.3 Relation with Random Walk Statistics

In this section, we will establish a connection between RPE-GNN and random walk on the graph. Our proposed approach inherently simulates a biased random walk in the graph. To aggregate features from k-hop neighborhood we perform a k-length random walk in the graph.

Suppose X_t denotes the state of the walk at the timestamp t and u, v are two connected nodes in the graph. Now, we can define the probability rule of the walk in the following way.

$$Pr[X_{t+1} = v | X_t = u] = \begin{cases} 1-p, & \forall u \neq v, v \in N(u). \\ p, & \text{otherwise.} \end{cases} \tag{2.5}$$

where $Pr[A]$ denotes the probability of occurring of event A, $0 < p < 1$ which controls the bias of the random walk. The $N(u)$ denotes the set of neighbors of the node u. In general, the transition matrix of the random walk is AD^{-1}. But following the lazy random walk (Wang et al., 2019) style, we can formulate the parameterized transition matrix as

$$W_p = pI + (1-p)AD^{-1}. \tag{2.6}$$

Now let us introduce the following lemma

Lemma 2.1. *If (λ, e) denotes the eigenvalue and eigenvector pair of the normalized Laplacian N respectively, then the corresponding eigenvalue and eigenvector pair of W_p is $(1-(1-p)\lambda, D^{\frac{1}{2}}e)$.*

The proof is available in the appendix 2.5.1. The over-smoothing occurs due to the convergence of the biased random walk to the stationary distribution (Chen et al., 2020c). Suppose at time-stamp t the state transition probability vector is denoted as s_t^p, which depends upon the parameter p. The dependence of s_t^p on p can be deduced by proposing the following theorem.

Theorem 2.1. *For any p_1, p_2 such that $0 < p_2 \leq p_1 < 1$, then the following inequality will hold*

$$\left(s_t^{p_1} - \lim_{t\to\infty} s_t^{p_1}\right) \geq \left(s_t^{p_2} - \lim_{t\to\infty} s_t^{p_2}\right). \tag{2.7}$$

The proof is discussed in the appendix 2.5.2. From the above theorem, we can conclude that if p tends to 1, which will eventually slow the convergence of state transition probabilities to the stationary distribution. If $p \to 1$, then from equation 2.5 it can be stated that the probability of moving from the current node to its neighbors will become lesser than the probability of staying in the current node. This fact indicates that after infinitely many steps

of the random walk, it is better to stay at the current vertex than to move to its neighbors. At that moment, exploring more paths does not improve the model's performance. The fact is also experimentally demonstrated in the section 2.3.6.

2.2.4 Connection with existing approaches

RPE-GNN has some significant properties of the propagation rule over the other existing approaches such as GCN (Kipf and Welling, 2016), GAT (Veličković et al., 2017), and GraphSage-GCN (Hamilton et al., 2017). Assume x and d are respectively the features and the degree of any node v, then the updated feature $\tilde{x}$ of GCN can be estimated as:

$$\tilde{x} = \frac{x}{d+1} + \sum_{j \in N(v)} \frac{1}{\sqrt{d+1)(d_j+1)}} x_j, \tag{2.8}$$

where x_j is the features of j^{th} neighbor of node v and $N(v)$ is the set of neighbors of the same. In GCN the aggregated features are symmetrically normalized by the degree of the neighbors of the node v. Similarly, RPE-GNN normalizes the same with the total number of random paths. The random paths may be assumed as the edges incorporated with the estimated path features connecting the source node to its multi-hop neighbors. Therefore, our method tackles the aggregation of the non-adjacent neighbors through the connecting paths.

On the other hand, the feature aggregation of GAT is formulated as:

$$\tilde{x} = x + \sum_{j \in N(v)} \frac{a_{ij}}{d_j} x_j, \tag{2.9}$$

where a_{ij} is the attention coefficient between i^{th} and j^{th} node. The attention coefficient a_{ij} is multiplied by the neighbor's features where the identical attention value weights each dimension. But RPE-GNN weights each dimension of the same by the distinct values. This implies our approach can provide feature-level attention weights.

Another architecture called GraphSage-GCN has the following propagation formula.

$$\tilde{x_i} = x_i + \frac{1}{d_i} \sum_j x_j. \tag{2.10}$$

In this case, the neighborhood features are also simply averaged by the node degree like GCN where we average the weighted features by the number of randomly selected paths. GraphSage-GCN cannot provide feature-level attention whereas RPE-GNN enables feature-level attention which is learned from the random paths. The message propagation technique's highlighted features help prevent over-smoothing in the deeper GNN architectures.

The common objective between RPE-GNN and Graph Diffusion Convolution (GDC) (Klicpera et al., 2019) is to gather multi-hop neighborhood information to update the node features. However, GDC and RPE-GNN have fundamental differences while aggregating features from long-range dependency. Applying GDC, initially, we make the graph denser because the higher powers of the transition matrix lead to interaction with the higher-order neighbors. Then a dense graph is sparsified by removing the edges whose weights are below a pre-defined threshold value. In contrast, RPE-GNN is a novel architecture, which rewires the graphs by randomly exploring multi-hop neighbors via sampling random paths. RPE-GNN does not require any threshold value to rewire the graph and enjoys complete freedom to select any random neighbors.

2.3 Experiments and Results

We perform diversified experiments to showcase the efficacy of our model. The experiments are done on the open benchmark graph datasets.

2.3.1 Details of Datasets

The experiments are carried out on four different categories of datasets.

Citation datasets We select Cora (Sen et al., 2008), Citeseer (Sen et al., 2008), and Pubmed (Sen et al., 2008) as the citation networks where the nodes represent documents, and the edges act as citations between the documents. Each document belongs to one academic topic. The features associated with each node correspond to the bag-of-words representation of the respective document.

Co-authorship datasets Coauthor CS (Shchur et al., 2018) and Coauthor Physics (Shchur et al., 2018) are two co-authorship networks. Nodes represent authors, and edges exist between them if they coauthored a paper. The features of the nodes represent the

keywords related to the author's paper. The label of each node denotes the field of study of the corresponding author.

Co-purchase datasets Amazon Computers (Shchur et al., 2018), and Amazon Photo (Shchur et al., 2018) are two co-purchase networks where each node denotes products, and an edge exists if two products are bought frequently. Node features denote the bag-of-words representation of the product reviews. Node labels indicate the product category.

Wikipedia networks Chameleon (Pei et al., 2020) is the Wikipedia network where nodes denote web pages from the Wikipedia pages, and edges exist if edges have mutual links between them. Node features are the bag-of-words of the nouns on the page. The node labels signify one of the five classes of average monthly web page traffic.

WebKB Datasets We choose Wisconsin and Cornell (Pei et al., 2020) as our webkb datasets where nodes represent web pages, and edges are the hyperlinks between the web pages. The node features are the bag-of-words representation of the web pages. These datasets are heterophilic graphs where connected nodes are more likely to be in different classes.

Table 2.1 summarizes the details of the datasets.

Table 2.1: Dataset Statistics

Dataset	Nodes	Edges	Classes	Features
Cora	2708	5429	7	1433
Citeseer	3327	4732	6	3703
Pubmed	19,717	44,338	3	500
Coauthor CS	18333	81894	15	6805
Coauthor Physics	34493	495924	5	8415
Amazon Photo	13752	491722	10	767
Amazon Computers	7650	238162	8	745
Chameleon	2277	36101	5	2325
Wisconsin	251	499	5	1703
Cornell	183	295	5	1703

The code is implemented by using Pytorch-Geometric (Fey and Lenssen, 2019) framework and is available at https://github.com/gnn-codes/rpe-gnn

2.3.2 Semi-Supervised Node Classification

Dataset Settings. For semi-supervised node classification, we use standard train-validation-test split of the three datasets Cora, Citeseer, and Pubmed as stated in (Kipf and Welling, 2016). For training 20 labeled samples are selected from each of the classes. For validation

Table 2.2: Mean classification accuracy(%) of the model comparing with different state-of-the-art approaches

Method	Cora	Citeseer	Pubmed
GCN	81.1	70.8	79.0
GAT	83.0	71.5	79.0
SGC	81.7	71.3	78.9
JKNet	81.1	69.8	78.1
APPNP	83.1	71.8	80.1
RPE-GNN(Ours)	**83.3±0.02**(8)	**72.1±0.01**(16)	**80.8±0.05**(8)

Table 2.3: Mean classification accuracy(%) of the RPE-GNN comparing with different state-of-the-art approaches where graph diffusion is employed as pre-processing.

Method	Cora	Citeseer	Pubmed
GDC-GCN	82.93	71.95	79.62
GDC-GAT	81.6	70.25	77.78
GDC-JKNet	82.78	71.87	79.95
GDC-APPNP	83.23	71.93	79.78
RPE-GNN	**83.3±0.02**	**72.1±0.01**	**80.8±0.05**
GDC-RPE-GNN	81.6 ±0.33	68.7 ±0.2	80.6 ±0.60

and testing, we select 500 nodes and 1000 nodes, respectively. For Coauthor CS, Coauthor Physics, Amazon Computers, and Amazon Photo, 20 samples are randomly selected from each class for training, 30 samples from each class are randomly selected for validation, and the rest of the samples are chosen for testing as described in (Liu et al., 2020).

Discussions. The performance of our model can be evident by comparing it with the other SOTA approaches. We consider GCN(Kipf and Welling, 2016), GAT (Veličković et al., 2017), SGC (Wu et al., 2019), Jumping Knowledge Networks (Xu et al., 2018b), and APPNP (Klicpera et al., 2018) as our contenders. Table 5.1 provides necessary details regarding the performances of RPE-GNN. The method outperforms all results on three Cora, Citeseer, and Pubmed. The optimal layer numbers are written in the parenthesis. The baselines like GCN, GAT, and JKNet utilize recursive message-passing frameworks. However, we employ non-recursive aggregation without performing redundant feature computations, and still, we achieve better performances over the datasets.

We also consider to pre-process data with graph diffusion convolution(GDC) (Klicpera et al., 2019) to evaluate the performance of RPE-GNN. We consider the variants of GNNs like GCN, GAT, GraphSAGE, and APPNP with the graph diffusion based pre-processing to compare the performances with the GDC-based RPE-GNN. The results are presented in

Table 2.3. From the results, it can be observed that applying graph diffusion does not improve model performance. Graph diffusion modifies the graph connections under some conditions, but RPE-GNN rewires the graph with long-distant nodes via sampling random paths.

To compare performances of RPE-GNN on co-authorship and co-purchase networks we include GCN (Kipf and Welling, 2016), GAT (Veličković et al., 2017), GraphSage (Hamilton et al., 2017), MoNet (Monti et al., 2017), and DAGNN (Liu et al., 2020) as our contenders. RPE-GNN outperforms all methods in the four datasets, as mentioned earlier. All said competitors utilize recursive aggregation in contrast to our aggregation, which is non-recursive and still achieves the best performances, which is demonstrated in Table 2.4. Coauthor CS and Coauthor Physics achieve the best performance when network depth is 2 and 8, respectively, whereas Amazon Computers and Amazon Photo accomplish the same when network depth is 8 and 32, respectively.

Table 2.4: Mean classification accuracy(%) is presented by comparing with the state-of-the-art architectures

Method	Coauthor CS	Coauthor Physics	Amazon Computers	Amazon Photo
GCN	91.1 ±0.5	92.8 ±1.0	82.6 ±2.4	91.2 ±1.2
GAT	90.5 ±0.6	92.5 ±0.9	78.0 ±19.0	85.7 ±20.3
MoNet	90.8 ±0.6	92.5 ±0.9	83.5 ±2.2	91.2 ±1.3
GraphSAGE-mean	91.3 ±2.8	93.0 ±0.8	82.4 ±1.8	91.4 ±1.3
GraphSAGE-maxpool	85.0 ±1.1	90.3 ±1.2	N/A	90.4 ±1.3
GraphSAGE-meanpool	89.6 ±0.9	92.6 ±1.0	79.9 ±2.3	90.7 ±1.6
DAGNN	92.8 ±0.9	94.0 ±0.6	84.5 ±1.2	92.0 ±0.8
RPE-GNN(Ours)	**93.18 ±0.02**	**94.87 ±0.01**	**84.78 ±0.05**	**93.14 ±0.01**

2.3.3 Analysis of Deep Models

We conduct a detailed study on the performance of RPE-GNN for the different network depths. We select three citation networks, Cora, Citeseer, and Pubmed, along with two heterophilic datasets, Wisconsin and Cornell. The number of hidden layers is chosen from the set $\{2, 4, 6, 8, 10, 12, 14, 16\}$. Figure 2.4 illustrates the variation of test accuracy when the network depth is increased gradually. For citation datasets, the model performance gradually improves throughout the experiment while the model depth increases. On the other side, the best performance of Wisconsin and Cornell is achieved in the deeper models.

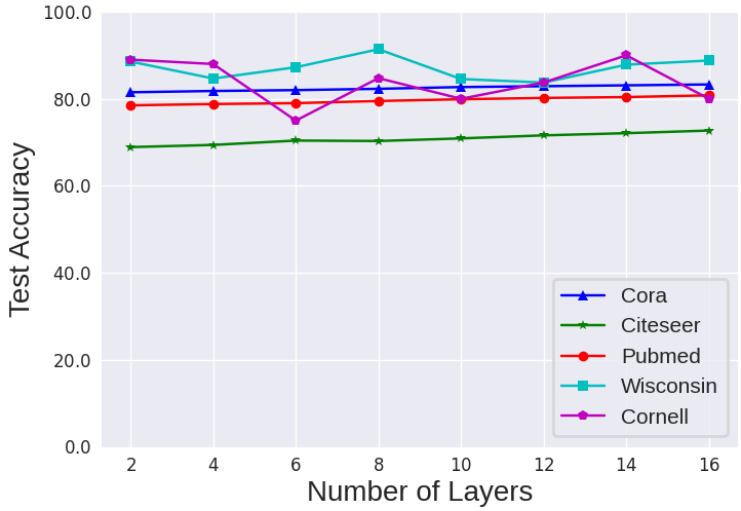


Figure 2.4: The performance of RPE-GNN is observed by varying the network depths in the model. RPE-GNN achieves the best performance on Cora, Citeseer, and Pubmed at 8^{th}, 16^{th}, and 8^{th} layers, respectively. RPE-GNN shows best results on Wisconsin and Cornell at 8^{th} and 14^{th} layers. Better test accuracy may be achieved when multiple layers are stacked together.

Therefore, the study justifies that going deeper into the GNN models is sometimes helpful to achieve optimal model performance. Thus, we can conclude that the proposed non-recursive aggregation strategy may mitigate the over-smoothing issue in the deeper architectures.

Table 2.5: Mean classification accuracy(%) of the model under fully-supervised condition

Method	Cora	Citeseer	Pubmed	Chameleon
GCN	85.77	73.68	88.13	28.18
GAT	86.37	74.32	87.62	42.93
Geom-GCN-I	85.19	77.99	90.05	60.31
Geom-GCN-P	84.93	75.14	88.09	60.90
Geom-GCN-S	85.27	74.71	84.75	59.96
APPNP	87.87	76.53	89.40	54.3
JKNet	85.25	75.85	88.94	60.07
JKNet(Drop)	87.46	75.96	89.45	62.08
Incep(Drop)	86.86	76.83	89.18	61.71
GCNII	88.49	77.08	89.57	60.61
GCNII*	88.01	77.13	**90.30**	62.48
RPE-GNN(Ours)	**89.15**(8)	**78.39**(8)	89.43(8)	**71.48**(16)

Figure 2.5: Test accuracy of RPE-GNN is observed by varying the number of random paths for different depths of the architecture. Test accuracy increases when the number of sampled paths is increased. The model with more depth needs more random paths to produce optimal results.

2.3.4 Full-Supervised Node Classification

Besides semi-supervised settings, we also evaluate RPE-GNN on fully-supervised node classification tasks.

Dataset Settings In this case, we include the web data network Chameleon (Pei et al., 2020) along with the previous three datasets, Cora, Citeseer, and Pubmed. As per (Chen et al., 2020c), the datasets are randomly split into $60\%, 20\%, 20\%$ for train, validation, and testing, respectively, for a fair comparison.

Discussions In addition to previously mentioned baselines we further include Geom-GCN (Pei et al., 2020), APPNP (Klicpera et al., 2018) and SIGN (Frasca et al., 2020) as our new contenders. We present the obtained results in Table 2.5. The best results are mentioned with the optimal layer number within parentheses. RPE-GNN shows almost 1% improvement on Cora and Citeseer datasets compared to all approaches. On the Pubmed dataset, the algorithm produces competitive results ($\leq 1\%$) compared to other approaches. On Pubmed still, RPE-GNN outperforms all other mentioned approaches except GCNII and GCNII*. For Chameleon, we achieve almost 9% improvements over SOTA. The results suggest that model performance can still be improved without having redundant feature estimation.

2.3.5 Architecture & Training

The architecture of RPE-GNN consists of a set of hidden layers, followed by an aggregation layer and, finally, a softmax layer. A k-layered RPE-GNN can perform k-hop neighborhood aggregation. Each hidden layer consists of a trainable weight matrix shared across all

the edges for that layer and is followed by batch normalization layers, non-linear activation function ReLU and dropout layers.

The model is trained by Adam (Kingma and Ba, 2014) optimizer with learning rate 0.2, and other parameters are set to defaults. The weight decay is set to 0.0005. We use 0.5 dropout to prevent potential overfitting. The model is trained for a maximum number of 200 steps, and the results are reported after averaging 10 steps during the test phase. The dimension of the hidden layers is 32. Cross-Entropy loss is used to optimize the model parameters. We also introduce a sampling strategy to generate random paths, which are broadly discussed in appendix 2.6.1

2.3.6 Effect of Sampling of Random Paths

The effect on the model performance with a different number of sampled paths in the graph is demonstrated in Figure 2.5. The experiment suggests that the test accuracy of the model significantly improves when the number of sampled paths is increased. The higher number of random paths enhances the scope of exploring the graph topology. Another observation is the test accuracy may not improve beyond an optimal number of random paths. The optimal number at which test accuracy becomes non-increasing depends on the dataset itself. One should note that for different depths of the model, the required number of random paths is different to achieve the best performance. In fact, for shallow architecture, the optimal number of paths is less than the requirement for the same in the case of deeper architecture.

2.4 Conclusion

We propose RPE-GNN, an architecture that aggregates node features from lower to higher-order neighborhoods in a non-recursive way by considering path features of randomly explored paths in the graph. We design an efficient technique to estimate path features without having prior knowledge of the predetermined random paths. We have theoretically shown that RPE-GNN simulates a biased random walk in the graph. We have demonstrated that RPE-GNN is capable of minimizing the effect of over-smoothing in the deeper GNN models through rigorous experiments. As a future work, the method corresponding to path features estimation can be further improved for better model performance. Also, our work can be

extended to define a new field of study regarding the non-recursive aggregation strategies for graph learning tasks.

2.5 Appendix A

2.5.1 Proof of Lemma III.1

Proof. If λ and e are the eigenvalue and the eigenvector of N respectively, then we have

$$Ne = \lambda e \tag{2.11}$$

By algebraic manipulation, the W_p can be rewritten as

$$W_p = (I - (1-p)D^{\frac{1}{2}}ND^{-\frac{1}{2}}) \tag{2.12}$$

Now, consider the following equation

$$\begin{aligned} W_p D^{\frac{1}{2}}e &= (I - (1-p)D^{\frac{1}{2}}ND^{-\frac{1}{2}})D^{\frac{1}{2}}e \\ &= D^{\frac{1}{2}}e - (1-p)D^{\frac{1}{2}}Ne \\ &= D^{\frac{1}{2}}e - (1-p)D^{\frac{1}{2}}\lambda e [\text{from equation 2.11}] \\ &= (1-(1-p)\lambda)D^{\frac{1}{2}}e \end{aligned} \tag{2.13}$$

Hence, the claim is proved.

□

2.5.2 Proof of Theorem III.2

Proof. The transition matrix of the biased random walk is derived as below

$$W_p = p.I + (1-p)AD^{-1}. \tag{2.14}$$

We know $L = D - A$ as the graph Laplacian. The symmetrically normalized Laplacian is represented as:

$$\begin{aligned} N &= D^{-\frac{1}{2}} L D^{-\frac{1}{2}} \\ &= D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}} \\ &= I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \end{aligned} \tag{2.15}$$

Reformulating the equation 2.14 by using the equation 2.15 we get

$$\begin{aligned} W_p &= p.I + (1-p)D^{\frac{1}{1}} D^{-\frac{1}{2}} A D^{-\frac{1}{2}} D^{-\frac{1}{2}} \\ &= p.I + (1-p)D^{\frac{1}{2}}(I - N)D^{-\frac{1}{2}} \\ &= p.I + (1-p)(I - D^{\frac{1}{2}} N D^{-\frac{1}{2}}) \\ &= I - (1-p)D^{\frac{1}{2}} N D^{-\frac{1}{2}} \end{aligned} \tag{2.16}$$

Suppose the state transition probabilities at time t is s_t them at time $t+1$ the state transition probabilities will be

$$s_{t+1} = W_p s_t \tag{2.17}$$

Substituting $t = 0$ we get $s_1 = W_p s_0$. As W_p is diagonalizable, then a set of eigenvectors forms a basis. Therefore, any vector can be expressed as

$$s_0 = \sum_{i=1}^{n} \alpha_i D^{\frac{1}{2}} e_i \tag{2.18}$$

where e_i is the i^{th} eigen vectors and n is the number of dimension. Now, we can rewrite

$$\begin{aligned} s_1 &= \sum_{i=0}^{n} \alpha_i W_p D^{\frac{1}{2}} e_i \\ &= \sum_{i=0}^{n} \alpha_i (1 - (1-p)\lambda_i) D^{\frac{1}{2}} e_i \end{aligned} \tag{2.19}$$

Iterating t times the expression will be

$$s_t = \sum_{i=0}^{n} \alpha_i (1 - (1-p)\lambda_i)^t D^{\frac{1}{2}} e_i \tag{2.20}$$

It is a well-known fact that eigenvalues of N lie between $[0, 2]$. Therefore, it is easy to observe that the eigenvalues of W_p lie between $[0, 1]$. Assume $\lambda_1 = 1, \lambda_j \leq 1 \forall j \in [2, n]$. Finally, the stationary distribution of the random walk is

$$\begin{aligned}\lim_{t\to\infty} s_t &= \lim_{t\to\infty} \sum_{i=0}^{n} \alpha_i (1-(1-p)\lambda_i)^t D^{\frac{1}{2}} e_i \\ &= \sum_{i=0}^{n} \alpha_i \lim_{t\to\infty} (1-(1-p)\lambda_i)^t D^{\frac{1}{2}} e_i \\ &= \alpha_1 D^{\frac{1}{2}} e_1\end{aligned} \tag{2.21}$$

It may observe that the eigenvalues of W_p other than 1 all lies between $[0, 1)$. Therefore, the rest of the terms vanish as t tends to infinity. Moreover, the expression s_t also depends on the parameter p so that we can rewrite it as s_t^p. WLOG for a fixed λ the following inequality will hold

$$(1-(1-p_1)\lambda) \geq (1-(1-p_2)\lambda), \tag{2.22}$$

where $p_1 \geq p_2$. For $t > 0$ the above inequality can be written as

$$\begin{aligned}&(1-(1-p_1)\lambda)^t \geq (1-(1-p_2)\lambda)^t \\ &\implies \sum_{i=0}^{n} (1-(1-p_1)\lambda_i)^t \geq \sum_{i=0}^{n} (1-(1-p_2)\lambda_i)^t, \\ &\implies \sum_{i=0}^{n} \alpha_i (1-(1-p_1)\lambda_i)^t D^{\frac{1}{2}} e_i \\ &\geq \sum_{i=0}^{n} \alpha_i (1-(1-p_2)\lambda_i)^t D^{\frac{1}{2}} e_i, \\ &\implies s_t^{p_1} \geq s_t^{p_2} \\ &\implies (s_t^{p_1} - \epsilon) \geq (s_t^{p_2} - \epsilon) \\ &\implies \left(s_t^{p_1} - \lim_{t\to\infty} s_t^{p_1}\right) \geq \left(s_t^{p_2} - \lim_{t\to\infty} s_t^{p_2}\right)\end{aligned}$$

From the equation 2.21, it can be deduced that the limiting value of the state transition probabilities is independent of p. Therefore, we can have $\lim_{t\to\infty} s_t^{p_1} = \lim_{t\to\infty} s_t^{p_2} = \epsilon$ for any p_1, p_2 where ϵ denotes the vector of the stationary distribution of the random walk. Hence, the theorem is proved. □

2.6 Appendix B

This section provides the details of the hyper-parameters used to reproduce the results.

2.6.1 Sampling Strategy

We adopt a simple and effective layer-wise sampling technique that prevents increasing the exponential number of random paths. At the beginning of each layer, we initialize a parameter that decides the maximum of how many neighbors of a node can be selected. For every layer, the value of the parameter remains fixed for each node in the graph. The parameter is increased to explore more graph topology and is decreased to restrict the number of random paths.

We also introduce two random variables for layer-wise sampling, namely n_{sample} and l_{range}. The first one decides the maximum number of neighbors that will be selected for every node in the graph. This value will be different for every layer. The second one indicates the number of layers up to which the node-wise sampling will be performed. After that layer, the sampling will be 1, i.e., only one neighbor will be randomly selected. So the default value of n_{sample} is 1.

The hyperparameters of the semi-supervised experiments are summarized in Table B.1. Table B.2 reports the hyperparameters of the fully-supervised experiments. The 'layers' denote the network depth where the algorithm performs best on the corresponding dataset. Here, L_2 represents the weight decay of the optimizer, and 'lr' denotes the learning rate used during the training.

Table B.1: Hyperparameters for semi-supervised node classification experiments

Dataset	Hyper-parameters
Cora	layers: 8, l_{range}: 4, n_{sample}: 3, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32
Citeseer	layers: 16, l_{range}: 4, n_{sample}: 4, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32
Pubmed	layers: 8, l_{range}: 3, n_{sample}: 4, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32
Coauthor CS	layers: 2, l_{range}: 2, n_{sample}: 4, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32
Coauthor Physics	layers: 8, l_{range}: 2, n_{sample}: 4, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32
Amazon Computers	layers: 8, l_{range}: 2, n_{sample}: 4, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32
Amazon Photo	layers: 32, l_{range}: 3, n_{sample}: 4, lr: 0.2, dropout: 0.50, L_2: 0.0005, hidden: 32

Table B.2: Hyperparameters for fully-supervised node classification experiments

Dataset	Hyper-parameters
Cora	layers: 8, l_{range}: 3, n_{sample}: 3, lr: 0.2, dropout: 0.50, L_2: 0.00005, hidden: 64
Citeseer	layers: 8, l_{range}: 2, n_{sample}: 3, lr: 0.2, dropout: 0.50, L_2: 0.00005, hidden: 64
Pubmed	layers: 8, l_{range}: 3, n_{sample}: 3, lr: 0.2, dropout: 0.50, L_2: 0.00005, hidden: 64
Chameleon	layers: 16, l_{range}: 3, n_{sample}: 3, lr: 0.02, dropout: 0.50, L_2: 0.00005, hidden: 64

Chapter 3

Addressing Graph Heterophily via Label-guided Graph Rewiring

Summary

Graph Neural Networks (GNNs) witness impressive performances on homophilic graphs characterized by a higher number of edges connecting nodes of similar class labels. A decline in the performance of GNNs can be experienced when applied to heterophilic graphs where most of the edges connect nodes with different class labels. This study presents a novel and versatile preprocessing framework comprising three fundamental stages. This framework can be seamlessly integrated with various GNN architectures to address heterophily within graphs effectively. In the initial stage, we predict class probabilities for nodes through a dense network. It is widely acknowledged that conventional feature-based similarity measures, such as cosine similarity, might not always accurately capture the correspondence between node pairs. Moving to the second stage, we introduce a re-weighting strategy guided by class embeddings generated from autoencoders to counter this limitation. In the final stage, we utilize the re-weighted similarity coefficients in a two-stage graph rewiring process. This process involves node deletion and subsequent insertion to generate a more homophily-oriented neighborhood. We re-use class embeddings by fusing them with the original node features to enrich the node features with class-level information. The updated node features and the rewired graph structure are ultimately fed into the GNN model. This facilitates effective message passing across neighborhoods. We extensively evaluate our approach on various standard graph datasets encompassing homophilic and heterophilic characteristics. Across these datasets, our framework consistently improves the performance of the established baseline methods.

3.1 Introduction

Message Passing (MP) (Gilmer et al., 2017b) in Graph Neural Networks (GNNs) (Scarselli et al., 2008), (Kipf and Welling, 2016), (Hamilton et al., 2017), (Wu et al., 2019), (Xu et al., 2018a), (Frasca et al., 2020) involves passing information between nodes in a graph, typically through a series of iterative steps, to update each node's representation based on its neighborhood structure and attributes. Message propagation performs best when adjacent nodes share similar class labels, resulting in a homophily network. Thus, Graph Convolutional Network (GCN) (Kipf and Welling, 2016), along with its well-known variants such as GAT (Veličković et al., 2017), GraphSAGE (Hamilton et al., 2017), and SGC (Wu et al., 2019), exhibits strong performance when the input graph is homophilic.

The efficacy of Message Passing-GNN (MP-GNN) is somewhat compromised when the graph is heterophilic, i.e., when adjacent nodes have different class labels. In this scenario, the direct application of message passing is not beneficial because it mixes disparate information from the neighborhood. A potential remedy may come from one of the three directions: **(1)** improving MP to tackle heterophily through the network (H_2GCN (Zhu et al., 2020), BM-GCN (He et al., 2022), HOG-GCN (Wang et al., 2022b), CPGNN (Zhu et al., 2021a), ACM-GCN (Luan et al., 2022) etc.), (**2**) applying learnable similarity measures to control the inflow of the messages from the neighborhoods (GPR-GNN (Chien et al., 2020), GGCN (Yan et al., 2022), etc.), (**3**) altering the graph topology by rewiring with the similar distant nodes (GPNN (Yang et al., 2022c), UGCN (Jin et al., 2021a), DHGR (Bi et al., 2022b) etc.). The first two types of methods operate on the heterophilic graphs without rewiring the graph topology. To some extent, those methods are successful, but the fixed graph topology restricts the direct interaction with the distant neighbors, which may contain crucial information (Pham et al., 2025). Those approaches cannot also provide an alternative view of the input graph structure, leading to significant information loss.

Our aim aligns with the pursuit of optimizing graph rewiring to identify informative nodes located distant from their respective central nodes. To achieve this meaningful rewiring, we introduce a novel and adaptable preprocessing framework named Label-guided Graph Rewiring, abbreviated as **LGR**. This framework is designed to seamlessly accommodate a diverse range of Graph Neural Networks (GNNs). In the initial phase, a dense network is trained to pre-

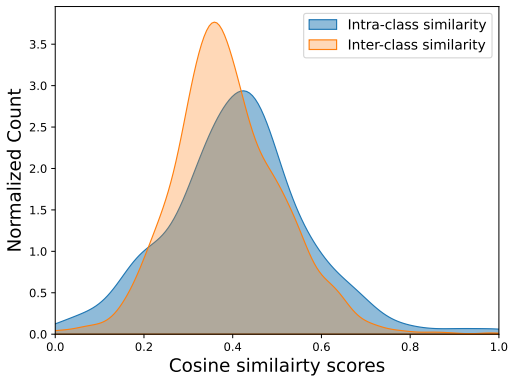


Figure 3.1: A study comparing the histograms between inter-class and intra-class feature similarities for the Texas dataset. In a heterophilic graph, one would typically expect inter-class similarity scores to be lower than intra-class similarity scores. However, the histograms reveal a different pattern, indicating that traditional similarity functions may not accurately capture the true relationships. This suggests a need for more advanced methods to measure true edge similarity in the context of heterophilic graphs.

dict class probabilities for the nodes. Given that the standard similarity measure (like cosine similarity) between the nodes may not be sufficient to decide the actual correspondence between the adjacent nodes. In the absence of the original ground-truth labels, the node class probabilities facilitate interpretation of the edge connections. The justification of the edge correspondence under the lens of cosine similarity culminates in the scenario demonstrated in Figure 3.1. The experiment is conducted on the Texas dataset, where we categorize the edges as (1) edges with both endpoints of the class label, say 2, and (2) edges with one endpoint having class label 2 and another endpoint having any class other than 2. The edges from the second category should have lower feature similarity values because the endpoints have different class labels. However, the figure illustrates the reverse scenario where the histogram of inter-class edges is almost identical to that of the intra-class edges. This supports the fact that the feature similarity measure does not represent the relationship between the nodes. On the other hand, the class embeddings are extracted from the autoencoders and combined with the predicted class probabilities obtained from MLP to generate weighted class embeddings. The updated class representations are utilized to evaluate label-guided similarity scores, which entail the information from both class probabilities and class embeddings. We employ the label-guided scores to re-weight the standard similarity measures between the nodes. The updated node similarity scores are utilized to modify the graph topology. We in-

troduce a two-stage rewiring process where several lower similar nodes will be initially deleted after including some higher similar nodes from the multi-hop neighborhoods. This two-stage rewiring process generates an informative and homophily-prone neighborhood for effective feature aggregation. Additionally, the weighted class embeddings are incorporated with the original node features to provide class-specific knowledge. To the best of our knowledge, we are the first to incorporate predicted class probabilities and autoencoder-based class embeddings to generate similarity scores in the context of heterophilic graphs. Also, our framework offers provisions for both the inclusion and exclusion of nodes according to the similarity scores, leveraging a more congenial network structure for message-passing. In the end-to-end trainable architectures like (Wang et al., 2022b; He et al., 2022) where multiple losses are combined to optimize, often requires proper balancing for faster convergence. In LGR, we pursue the independent MLP, Autoencoder, and GNN training processes, obliterating the requirement for balancing multiple loss functions. Additionally, we augment node features by enriching them with class-level information. After the completion of the preprocessing steps, the modified node features and rewired graph topology are fed into the MPNNs to estimate the class labels of the nodes. Intuitively, the preprocessing stage converts the heterophilic or homophilic input network to a more homophilic one, and GNN models can be easily applied to the modified graph.

Contribution Our contributions can be succinctly summarized as follows:

- We introduce an efficient preprocessing framework, Label-guided Graph Rewiring (**LGR**), adept at addressing homophilic and heterophilic graphs. Recognizing the limitations of the conventional similarity measures in accurately identifying genuine node-pair relationships, our framework addresses this challenge by assigning weights derived from the combination of the predicted class probabilities from a dense network and class representations from the autoencoder. To the best of our knowledge, we are the first to incorporate predicted class probabilities and autoencoder-based class embeddings to generate similarity scores in the context of heterophilic graphs.

- LGR incorporates a comprehensive two-stage graph rewiring process. It eliminates redundant or less informative nodes and strategically includes pertinent long-distance nodes, thereby imbuing the neighborhood with a heightened sense of homophily.

- In end-to-end trainable architectures like (Wang et al., 2022b; He et al., 2022) where multiple losses are combined to optimize, often proper balancing is required for faster convergence. In LGR, we pursue the independent training processes of MLP, Autoencoder, and GNN, which obliterate the consolidation of multiple loss functions. We also augment node features with class embeddings to provide rich class-level information.
- We offer extensive theoretical insights to understand the working mechanism of the LGR framework.
- Our framework seamlessly lends itself to integration with a diverse array of GNN models, including prominent ones such as GAT (Veličković et al., 2017), GCNII (Chen et al., 2020c), H_2GCN (Zhu et al., 2020) etc. This integration allows for the efficient transmission of messages within the graph, thereby preserving the inherent flexibility of our proposed framework.

3.2 Proposed Method

3.2.1 Preliminaries

Consider an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ contains $n = |\mathcal{V}|$ is number of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The feature matrix $X \in \mathbb{R}^{n \times d}$ contains the node attributes of the graph where X_i denotes the d-dimensional node features of i^{th} node. The Y denotes the set of node labels of the graph, and $\mathcal{T}_{\mathcal{V}}$ denotes the set of training nodes.

Node Homophily. It estimates the ratio between the number of same-class neighbor nodes to all neighbors in the graph. Let us define $\mathcal{H}_n$ as

$$\mathcal{H}_n = \frac{1}{|\mathcal{V}|} \sum_{v_i \in \mathcal{V}} \frac{|\{y_j | y_j \in \mathcal{N}_i, Y_j = Y_i|\}}{|\mathcal{N}_i|}. \tag{3.1}$$

Edge Homophily. It measures the ratio between intra-class edges to the total number of edges in the graph. Let us define $\mathcal{H}_e$ as following,

$$\mathcal{H}_e = \frac{|\{(u, v) | (u, v) \in \mathcal{E}, Y_u = Y_v\}|}{|\mathcal{E}|}. \tag{3.2}$$

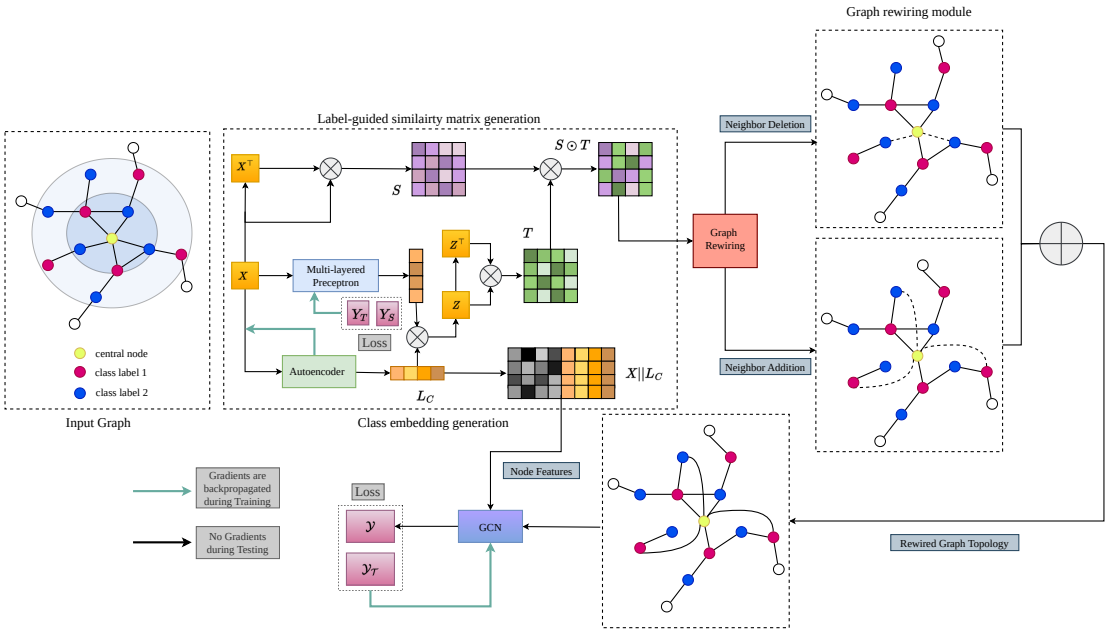


Figure 3.2: The complete workflow of LGR-GNN is elucidated, encompassing three preprocessing steps: (1) the generation of a similarity matrix guided by class probabilities, (2) the class embedding generator, and (3) the graph rewiring module. The feature matrix X is multiplied by its transpose $X^\top$ to compute the similarity matrix. The class embeddings (L_c) are generated from the autoencoder, and the predicted class probabilities are generated from the MLP. The class embeddings are combined to produce weighted class embeddings (Z) and later utilized to produce a label-guided similarity matrix T, which is subsequently employed within the graph rewiring module. Furthermore, L_c's are combined with the original node features X. Finally, the modified node features and the rewired graph are plugged into the Graph Convolutional Network (GCN) to predict the node labels.

From the above formulae, the higher value indicates the high homophily of the graph and vice-versa. In this paper, we will consider the edge homophily ratio to measure the homophilic or heterophilic properties of the graphs.

3.2.2 Problem Statement

We aim to perform node classification on the input $\mathcal{G}$ in the ambit of transductive and fully supervised settings. The $\mathcal{G}$ can be either homophilic or heterophilic and contains a set of nodes with class labels, which are tagged as the training set $\mathcal{T}_\mathcal{V}$ to streamline the training process effectively. Solving the node classification is akin to designing a $F_\theta : (X, A) \rightarrow \tilde{Y}$ where $\tilde{Y}$ denotes the predicted class labels. Our primary target is to propose a rewiring strategy that tackles the graph heterophily. Suppose, a rewiring strategy is $\mathcal{R}_\mathcal{G}$ can be expressed

as $\mathcal{R}_{\mathcal{G}} : (X, A) \rightarrow (X, A_{\mathcal{R}})$. Therefore, our final target is to devise a strategy such that $F_\theta \circ \mathcal{R}_{\mathcal{G}} : (X, A) \rightarrow \tilde{Y}$.

3.2.3 Generation of Class Probabilities from MLP

This is the initial step of the proposed framework. We train a Multi-Layered Perceptron (MLP) to evaluate the class probabilities for the nodes. The feature matrix X is provided as input to the MLP without any information regarding the graph topology. The following equation represents the softmax output of the nodes.

$$\overline{\mathcal{Y}} = \text{softmax}(\sigma(\text{MLP}(X \mid \Theta))) \in \mathbb{R}^{n \times C}, \tag{3.3}$$

where σ denotes the non-linear activation function, Θ denotes the learnable parameters of the MLP, C is the distinct classes, and $\overline{\mathcal{Y}}$ is the predicted class probabilities. The parameters of the MLP are optimized by minimizing the following loss function computed over labels of available training nodes.

$$\mathcal{L}_{\text{MLP}} = \sum_{v_i \in \mathcal{T}_{\mathcal{V}}} \mathcal{F}(\overline{\mathcal{Y}}_i, Y_i), \tag{3.4}$$

where $\overline{\mathcal{Y}}_i$ and Y_i denote the predicted label vector and the available ground-truth label for the training node $v_i \in \mathcal{T}_{\mathcal{V}}$. $\mathcal{F}$ is the cross-entropy loss, and the loss is computed over all nodes in the training set. In this context, we have both predicted class probabilities and ground truth labels for nodes belonging to the training set $\mathcal{T}_{\mathcal{V}}$ and only class probabilities for the nodes not in the training set. To maximize the utility of available ground-truth information, we replaced the class probabilities of the training nodes with the respective one-hot encodings and class probabilities remain unchanged for other nodes. Finally, we construct a label matrix $\hat{\mathcal{Y}}$ as follows,

$$\hat{\mathcal{Y}}_i = \begin{cases} \overline{\mathcal{Y}}_i & i \notin \mathcal{T}_{\mathcal{V}} \\ \overline{Y_i} & i \in \mathcal{T}_{\mathcal{V}}, \end{cases} \tag{3.5}$$

where $\overline{Y_i}$ denotes the one-hot representation for i^{th} node.

3.2.4 Class Embeddings by Autoencoder

We generate class embeddings using a shared autoencoder across all available class labels in $\mathcal{G}$. Class embeddings are the essence of the information regarding the classes. If n_c is the number of samples belonging to the class label $c \in C$, then the corresponding feature matrix is denoted by $X_c \in \mathbb{R}^{n_c \times d}$. The class-specific feature matrices are passed through an Autoencoder to the latent space representation. The following equation represents the output from the model

$$L_c = \text{Enc}(X_c \,|\, \Phi_e) \qquad X'_c = \text{Dec}(L_c \,|\, \Phi_d), \tag{3.6}$$

where X'_c is the reconstructed output from the decoder part of the autoencoder. $L_c \in \mathbb{R}^{C \times l}$ is the l-dimensional latent class representation generated from the encoder part from the autoencoder. Φ_e and Φ_d denote the trainable parameters of the encoder and decoder part, respectively. The parameters of the autoencoder are shared across all the class labels. The autoencoder is trained to learn the corresponding class embeddings for each class label. The training is completed by minimizing the loss function $\mathcal{L}_{\text{AE}}$ as

$$\mathcal{L}_{\text{AE}} = \text{MSE}(X_c, X'_c), \tag{3.7}$$

where MSE denotes the mean square error. The learned class embeddings are augmented with the node features to produce an augmented feature matrix $\hat{X} \in \mathbb{R}^{n \times (d+l)}$. For the node v_i, the class embeddings are concatenated with the input node features. The following equation represents the procedure.

$$\hat{X}_i = [X_i \,||\, L_c] \in \mathbb{R}^{(d+l)}, \tag{3.8}$$

where X'_i and L_c denote the augmented features and corresponding latent class representation where c represents the class label for node v_i.

3.2.5 Class Embeddings-guided Similarity Matrix

The class labels of the adjacent nodes are likely to depend on the similarity of the corresponding node features. A common practice is to employ a cosine similarity measure to capture node feature similarity. The feature similarity matrix, where S_{ij} represents the similarity of pair-wise node features, can be represented as,

$$S_{ij} = \frac{x_i^\top x_j}{||x_i||||x_j||}, \tag{3.9}$$

where x_i, x_j are the node features of node v_i and v_j respectively and $||.||$ denotes the Euclidean norm. The interplay between the node labels and feature similarities can be further improved by involving information on the predicted class probabilities obtained from the MLP. Initially, we will estimate weighted class embeddings by the latent representation of classes generated and the predicted class probabilities as,

$$Z = (\tilde{\mathcal{Y}} L_c) \in \mathbb{R}^{n \times l}, \tag{3.10}$$

where L_c denotes the class embeddings corresponding to class c. Z represents the weighted combination of the class probabilities and class embeddings. If a node $i \in \mathcal{T}_\mathcal{V}$, we will replace Z_i with its true class embeddings L_c obtained from the autoencoder where c is the class label of node i. Conversely, if the node is out of the training set, then Z_i will be kept unaltered, which is the weighted version of the L_c, which is more informative because it captures the weights learned from the MLP. Finally, we introduce the label-guided weight matrix $T \in \mathbb{R}^{n \times n}$ which is constructed in the following way,

$$T = softmax(ZZ^\top) \in \mathbb{R}^{n \times n}, \tag{3.11}$$

where T contains weights for every pair of nodes, which depends not merely on the predicted class labels but also on the latent class vectors. Thereby, the similarities of the node-pairs are more enriched with information. We update the feature similarity matrix by performing element-wise multiplication between S and T

$$\tilde{S} = S \odot T. \tag{3.12}$$

The class embeddings-guided matrix $\tilde{S}$ contains the information of both feature similarity along with the ground truth labels and predicted class labels. This approach mitigates the sole dependence on the feature similarity measures. The $\tilde{S}$ will be considered to modify the graph topology, vividly discussed in the next section.

3.2.6 Rewiring of the Graph Topology

Heterophilic graphs exhibit a higher count of heterophilic edges compared to homophilic edges. To address this disparity, we modify the graph's topology, thereby simultaneously reducing the number of heterophilic connections and augmenting the homophilic edges. Our approach involves a meticulous two-stage rewiring process encompassing edge deletion and edge insertion, creating a fresh neighborhood around the nodes. The determinations guiding this process stem from the class embeddings-enhanced similarity matrix $\tilde{S}$. To illustrate, let's consider a node v_i and its associated neighborhood $\mathcal{N}(v_i)$. Within this context, $\tilde{S}_{v_i} \in \mathbb{R}^n$ signifies the array of similarity scores pertaining to n nodes within the graph of node v_i. Furthermore, we introduce two key variables: k_t, which delineates the maximum number of nodes permissible for addition to the neighborhood, and k_b, designating the upper limit on the number of nodes that can be excised from the neighborhood.

Edge Deletion: The nodes are deleted from the existing neighborhood depending on the scores generated from $\tilde{S_{v_i}}$. The maximum k_b number of nodes with the lowest scores are selected. The set can be represented as $\mathcal{N}_{\mathrm{d}} = \{v_{j_1}, v_{j_2}, \cdot, v_{j_{k_b}}\}$. The updated neighborhood $\mathcal{N}_{\mathrm{deleted}}(v_i)$ is represented as:

$$\mathcal{N}_{\mathrm{deleted}}(v_i) = \mathcal{N}(v_i) \setminus \mathcal{N}_d. \tag{3.13}$$

If $k_b \geq |\mathcal{N}(v_i)|$, then the $\mathcal{N}_{\mathrm{deleted}}(v_i) = \phi$, that is, the node will be isolated in the graph. The node insertion method tackles the extreme case, which we discuss in the next section.

Edge Insertion Similar nodes are inserted in the respective neighborhoods in this stage to make the graph more homophilic. Similar and informative nodes are identified from the high score values from $\tilde{S_{v_i}}$. The maximum k_t number of nodes with higher scores are selected for the insertion in the neighborhood. The set of newly added nodes are presented

as $\mathcal{N}_{\text{a}} = \{v_{m_1}, v_{m_2}, \cdot, v_{m_{k_t}}\}$. After node addition the updated neighborhood will $\mathcal{N}_{\text{added}}(v_i)$ be

$$\mathcal{N}_{\text{added}}(v_i) = \mathcal{N}_{\text{deleted}}(v_i) \ \cup \ \mathcal{N}_a. \tag{3.14}$$

Finally, we define the updated neighborhood as $\overline{\mathcal{N}}(v_i) = \mathcal{N}_{\text{added}}(v_i)$. The rewired neighborhoods contain more similar nodes than the dissimilar ones, which makes the graph more homophilic. It is noteworthy that our two-stage rewiring process produces neighborhoods that respect the original ones. Assume a node similar to the centering node might be present in the set of similar nodes retained during the inclusion process. Concurrently, a dissimilar node of the centering node might be discarded if it is present in the set of dissimilar nodes. This mode of inclusion and exclusion facilitates the generation of neighborhoods derived from the original graph topology, which is prone to be more homeopathic than the previous ones.

Theorem 3.1. *For any vertex v in the $\mathcal{G}$ with neighborhood is $\mathcal{N}_v$ with edge homophily ratio h and $|\mathcal{N}_v| = d$. Define m as the number of neighbors with identical class labels as the label of v. After applying the LGR framework. the updated neighborhood became $\mathcal{N}'_v$ with modified edge homophily ratio h'. Assume h_t and h_b are the respective probabilities for selecting nodes of the class label of v according to the similarity scores obtained from MLP and autoencoder. Consider k_t and k_b as the number of nodes chosen for inclusion and exclusion, respectively. Therefore, if $\frac{k_t}{k_b} > \left(\frac{dh_b - m}{dh_t - m}\right)$, then we have $h' > h$ implies the homophily ratio will increase for the rewired neighborhood in $\mathcal{G}$.*

Proof. See in Appendix 3.5 □

3.2.7 Integrating with MP-GNNs

After the completion of the pre-processing stage, we obtained a rewired graph $\hat{\mathcal{G}} = (\hat{X}, \hat{A})$ where $\hat{X}$ and $\hat{A}$ denote the class embedding-augmented feature matrix and rewired adjacency matrix. Assume $\hat{D}$ is the degree matrix of the rewired graph. We introduce a multi-layered Graph Neural Network (GNN) where layer-wise transformation is defined as follows

$$H^{(m+1)} = \sigma(\text{GNN}(\hat{A}, H^{(m)}, W^{(m)})), \tag{3.15}$$

where $H^{(m+1)}$ denotes the transformed node features at $(m+1)^{th}$ layer, $H^{(0)} = \hat{X}$, $W^{(m)}$ is the parameterized transformation at the m^{th} layer where l varies from $[0, M-1]$, and σ denotes a non-linear activation function. Notably, the equation applies to any GNN architecture where our framework can easily fit.

Table 3.1: Details of the standard homophilic and heterophilic graph datasets are presented.

	Homophilic graphs			**Heterophilic graphs**					
Datasets	Cora	Citeseer	Pubmed	Cornell	Texas	Wisconsin	Actor	Squirrel	Chameleon
Nodes	2808	3327	19717	183	183	251	7600	5201	2277
Edges	5429	4732	44338	295	309	499	33544	198493	36101
Features	1433	3703	500	1703	1703	1703	931	2089	2325
Classes	7	6	3	5	5	5	5	5	5
$\mathcal{H}_n$	0.82	0.71	0.79	0.11	0.10	0.13	0.21	0.22	0.25
$\mathcal{H}_e$	0.81	0.74	0.80	0.13	0.11	0.20	0.22	0.22	0.23

3.2.8 Training and Loss optimization

Our proposed framework has three key stages: (1) training of MLP, (2) training of autoencoder, and (3) training of GNN. The architectures mentioned above, MLP, autoencoder, and GNN, are trained independently with the available node labels in the training set. The loss terms are not summed up but optimized independently, removing the overhead of balancing them during training. This is in stark contrast with the end-to-end trainable architectures. Finally, the loss of the GNN can be represented as follows

$$\mathcal{L}_{\text{GCN}} = \sum_{i \in \mathcal{T}_V} f(H_i^M, Y_i), \tag{3.16}$$

where H^M is the output of the last layer of the GNN, Y_i is the ground-truth label of the i^{th} training instance and f is the CrossEntropy loss function which is widely used for the training of the GCN architecture. The independent optimization of the different components ensures the flexibility of the framework to incorporate with any MP-GNNs.

3.3 Experiments

3.3.1 Details of the Datasets

We consider 3 standard homophilic graphs from (**?**) and 6 standard heterophilic graphs (Pei et al., 2020) to evaluate the performance of our proposed framework. Additionally, we also include 5 newly proposed heterophilic graphs from (Platonov et al., 2023) to conduct the experiments. The details of the standard datasets are provided in Table 3.1. The details of the new datasets can be found in Table 3.9 and refer to Section 3.6.1 in the Appendix 3.6.

WebKB Datasets This dataset comprises three networks - Cornell, Texas, and Wisconsin. Each node in these networks represents a web page, and its features consist of bag-of-words representations. The edges between nodes indicate hyperlinks connecting the web pages. Class labels are assigned to each node, representing categories such as students, staff, faculty, project, and course.

Actor Dataset : The Actor dataset is obtained from film-director-actor-writer networks. Nodes in this network represent actors, and edges exist between two actors if there is a co-occurrence of them on the same Wikipedia page. Node feature vectors are created using bag-of-word representations of the keywords found on the actors' web pages. Each node is assigned a class label based on information obtained from the actor's Wikipedia page, categorizing them into five classes.

Wikipedia Datasets: In this dataset, we only focused on the Squirrel network, which represents Wikipedia pages. Nodes in this network correspond to individual web pages, and edges denote connections between them. The node features are derived from the bag-of-words representations of nouns on the respective pages. Each node is assigned a class label based on the average monthly traffic received by its corresponding web page.

Citation Datasets: We included three citation networks from the Planetoid dataset - Cora, Citeseer, and Pubmed. In these networks, nodes represent papers, and edges indicate citation relationships between them. Node features are derived from the bag-of-word representations of paper titles and abstracts. Each node is assigned a class label representing a specific academic topic. These citation networks are homophilic in nature, whereas the former three types of datasets exhibit a highly heterophilic nature. For a more detailed description of the datasets, please refer to Table 3.1, which provides additional information regarding their

specific characteristics.

Table 3.2: The mean test accuracy and standard deviations of the various GNN models preprocessed by the LGR framework on heterophilic graphs are presented. The best results are boldfaced, and average gains are highlighted in green.

Methods	Variant	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Actor
GCN	vanilla	46.48 ± 7.72	62.70 ± 5.51	56.07 ± 5.49	42.14 ± 2.07	29.75 ± 0.84	29.03 ± 1.03
	SDRF	48.10 ± 8.52	61.89 ± 4.59	62.15 ± 6.32	56.66 ± 1.41	35.36 ± 1.09	31.02 ± 0.70
	FoSR	45.40 ± 6.48	63.51 ± 5.30	54.11 ± 6.57	38.94 ± 2.41	29.63 ± 0.68	29.12 ± 1.27
	Drew	50.00 ± 6.76	59.73 ± 3.51	54.31 ± 9.49	43.79 ± 2.02	32.43 ± 0.77	27.98 ± 1.01
	LGR (Ours)	**85.40 ± 5.43**	**85.13 ±6.53**	**89.21 ± 4.22**	**69.51 ± 6.97**	**61.14 ± 4.47**	**61.05 ± 3.26**
GAT	vanilla	48.37 ± 5.59	58.91 ± 4.32	55.68 ± 5.89	34.14 ± 5.61	25.91 ± 2.47	27.41 ± 0.98
	SDRF	50.81 ± 7.13	63.24 ± 6.85	55.88 ± 4.82	60.76 ± 11.00	59.99 ± 2.01	26.70 ± 2.12
	FoSR	46.75 ± 5.54	59.45 ± 4.68	53.92 ± 7.08	28.90 ± 2.23	25.36 ± 3.35	27.31 ± 1.06
	Drew	45.68 ± 8.84	59.19 ± 4.59	60.39 ± 11.22	42.24 ± 1.26	30.61 ± 1.50	28.61 ± 0.84
	LGR (Ours)	**85.67 ± 5.67**	**83.78 ± 5.12**	**88.03 ± 3.44**	**69.82 ± 7.15**	**60.73 ± 6.36**	**61.25 ± 7.55**
APPNP	vanilla	39.45 ± 9.06	50.81 ± 19.48	50.39 ± 7.49	25.28 ± 4.60	21.39 ± 1.35	25.74 ± 1.31
	SDRF	46.48 ± 6.25	55.13 ± 13.73	46.86 ± 9.17	35.87 ± 3.42	23.95 ± 3.27	25.38 ± 1.08
	FoSR	40.00 ± 7.23	58.10 ± 7.27	50.39 ± 6.68	22.58 ± 2.59	21.70 ± 1.48	27.00 ± 0.98
	Drew	73.51 ± 5.38	70.00 ± 11.42	86.08 ± 3.87	47.08 ± 2.29	29.87 ± 1.40	35.99 ± 1.02
	LGR (Ours)	**82.97 ± 3.83**	**79.72 ± 17.85**	**87.25 ± 3.42**	**69.21 ± 6.86**	**57.57 ± 1.80**	**52.64 ± 2.33**
GCNII	vanilla	52.43 ± 9.29	51.35 ± 22.41	44.47 ± 8.72	40.87 ± 1.54	30.90 ± 1.65	34.49 ± 0.91
	SDRF	42.70 ± 10.99	51.62 ± 17.28	59.99 ± 18.20	60.92 ± 2.53	42.54 ± 2.55	33.41 ± 1.02
	FoSR	44.32 ± 4.39	59.18 ± 4.43	51.37 ± 7.00	48.17 ± 1.72	30.46 ± 1.18	34.25 ± 0.77
	Drew	61.08 ± 12.10	62.70 ± 6.14	74.12 ± 4.54	43.38 ± 3.09	31.98 ± 1.23	34.66 ± 1.12
	LGR (Ours)	**89.54 ± 5.89**	**86.48 ± 5.79**	**89.01 ± 3.43**	**69.18 ± 6.96**	**60.86 ± 3.38**	**61.26 ± 2.55**
GPRGNN	vanilla	73.24 ± 10.63	72.97 ± 13.45	79.41 ± 9.08	24.25 ± 1.88	30.11 ± 1.90	28.76 ± 2.71
	SDRF	63.78 ± 10.89	72.16 ± 8.72	81.17 ± 6.08	53.64 ± 9.23	38.01 ± 3.70	27.10 ± 2.38
	FoSR	73.51 ± 4.49	72.97 ± 10.74	77.25 ± 7.75	30.21 ± 7.22	30.10 ± 2.08	28.63 ± 2.95
	Drew	66.76 ± 5.55	69.19 ± 7.37	76.86 ± 5.25	43.29 ± 1.78	32.56 ± 1.82	34.72 ± 0.42
	LGR (Ours)	**85.94 ± 6.01**	**84.32 ± 7.63**	**87.84 ± 5.02**	**69.42 ± 6.91**	**50.67 ± 3.59**	**55.84 ± 4.08**
H_2GCN	vanilla	64.86 ± 4.68	71.08 ± 3.83	71.17 ± 5.68	49.14 ± 1.93	33.31 ± 1.18	32.14 ± 0.52
	SDRF	60.54 ± 5.56	65.67 ± 6.62	60.78 ± 10.63	**72.06 ± 2.32**	**62.98 ± 1.53**	27.81 ± 1.89
	FoSR	65.40 ± 4.32	67.83 ± 3.90	71.76 ± 5.76	47.47 ± 2.65	33.26 ± 0.67	30.83 ± 1.89
	Drew	52.70 ± 7.28	62.16 ± 4.19	54.90 ± 6.74	40.24 ± 2.04	32.58 ± 1.18	33.28 ± 0.99
	LGR (Ours)	**85.13 ± 5.82**	**86.75 ± 6.67**	**88.03 ± 3.21**	69.67 ± 7.02	60.82 ± 1.45	**61.42 ± 2.69**
Avg Gain	-	**23.39**	**17.11**	**15.62**	**10.95**	**13.84**	**25.86**

3.3.2 Experimental Setup

We run experiments on 9 standard datasets, including 6 heterophilic and 3 homophilic graphs. Each dataset has 10 different splits, which are curated by Pei *et al* (Pei et al., 2020) where the ratio of the train/validation/test sets are $60\%, 20\%, 20\%$ respectively. For new 5 heterophilic datasets (Platonov et al., 2023), there are 10 splits with $50\%, 25\%, 25\%$ for train/validation/test respectively. The whole pipeline is comprised of three key neural networks: (1) MLP, (2) Autoencoder, and (3) GNN model. Each of them is trained separately, and the results are utilized sequentially. In each module, we employ ReLU activation between the hidden layers. We also use Dropout to facilitate the faster convergence of the training.

The model parameters are optimized by Adam optimizer. We report the mean test accuracy after averaging the results obtained from all splits across the standard datasets. For the new datasets, we report mean test accuracy for Roman-empire and Amazon-ratings and mean ROC-AUC for Minesweeper, Tolokers, and Questions after averaging results obtained from all splits. Further details on the hyperparameters are vividly discussed in Supplementary Section I. We implement the framework by using Pytorch and Pytorch-Geometric packages, and the code is available at https://github.com/kushalbose92/LGR-GCN/tree/main.

3.3.3 Discussion on Results

We conduct comprehensive experiments on a set of 3 homophilic networks and 6 heterophilic networks. We also include more 5 heterophilic graphs curated by (Platonov et al., 2023) to validate the efficacy of LGR. In conjunction with LGR, we integrate six models as GCN (Kipf and Welling, 2016), GAT (Veličković et al., 2017), APPNP (Klicpera et al., 2018), GCNII (Chen et al., 2020c), GPRGNN (Chien et al., 2020), and H_2GCN (Zhu et al., 2020) to showcase the versatile adaptability of our framework. Detailed quantitative assessments are compiled in Tables 3.2, 3.3, and 3.5. We compare the performance of LGR with three widely adopted rewiring strategies, namely SDRF (Topping et al., 2021b), FoSR (Karhadkar et al., 2022), and Drew (Gutteridge et al., 2023), to demonstrate the prowess of the proposed rewiring framework. The GNN models are grouped into two distinct categories: the first category, tailored for homophilic graphs, encompasses models such as GCN, GAT, GCNII, and APPNP. The second category, dedicated to well-adopted heterophilic networks, features H_2GCN and GPRGNN.

The results show that the performance of the base models improved with commendable margins when input graphs were preprocessed with the LGR framework. We employ average gain (Bi et al., 2022b) as the yardstick to measure the improvements on multiple datasets, comparing with the SDRF and FoSR. The proposed framework shows higher average gains across different heterophilic datasets, especially for Actor and Cornell datasets. LGR performs excellently on dense graphs like Chameleon, Squirrel, and Actor networks. The dense graphs often suffer from oversquashing (Alon and Yahav, 2020), which seems to be alleviated by the LGR framework by applying efficient rewiring of the graph topology. The framework becomes efficacious even for the homophilic networks where average gains show an uptick.

Specifically, methods like GCN, GAT, APPNP, and GCNII, designed for homophilic networks, enjoy the benefits of rewiring by the LGR framework, showing a significant outcome. We also observed performance improvements on the datasets obtained from (Platonov et al., 2023). Platonov identified issues in the standard datasets like Chameleon or Squirrel; the new heterophilic graphs are free from such issues. We also observed performance gains in those datasets and outperformed other contenders across every dataset. The results suggest that our proposed framework is readily adaptable to any preferred GNN model, tailored for both homophilic and heterophilic graphs.

Table 3.3: The mean test accuracy and standard deviations of the various GNN models preprocessed by the LGR framework on homophilic graphs are presented. The best results are boldfaced, and the average gains per model are highlighted in green.

Methods	Variant	Cora	Citeseer	Pubmed
GCN	vanilla	86.47 ± 1.20	76.25 ± 1.49	87.77 ± 0.47
	SDRF	86.90 ± 1.13	76.32 ± 1.64	86.82 ± 0.57
	FoSR	86.78 ± 1.43	76.22 ± 1.69	86.81 ± 0.55
	Drew	76.42 ± 9.00	71.91 ± 2.62	OOM
	LGR (Ours)	**87.74 ± 1.29**	**76.78 ± 1.36**	**88.07 ± 0.34**
GAT	vanilla	81.01 ± 8.13	69.59 ± 6.53	77.40 ± 5.61
	SDRF	33.03 ± 6.71	55.60 ± 7.28	63.91 ± 3.31
	FoSR	31.34 ± 2.99	54.90 ± 8.65	64.02 ± 3.36
	Drew	74.25 ± 11.03	68.53 ± 5.45	OOM
	LGR (Ours)	**86.01 ± 1.42**	**75.79 ± 1.84**	**87.33 ± 0.51**
APPNP	vanilla	83.59 ± 6.69	70.77 ± 11.02	75.07 ± 5.16
	SDRF	35.01 ± 7.48	38.31 ± 7.48	73.32 ± 5.14
	FoSR	37.24 ± 9.92	35.04 ± 8.80	75.74 ± 6.42
	Drew	74.25 ± 1.75	72.07 ± 2.01	OOM
	LGR (Ours)	**86.80 ± 1.37**	**76.74 ± 1.51**	**87.56 ± 0.42**
GCNII	vanilla	83.58 ± 1.58	71.07 ± 21.02	88.32 ± 0.35
	SDRF	**86.46 ± 0.88**	73.68 ± 1.71	**89.48 ± 0.53**
	FoSR	86.31 ± 1.02	73.37 ± 1.79	89.45 ± 0.43
	Drew	83.58 ± 2.84	72.56 ± 4.64	OOM
	LGR (Ours)	84.38 ± 1.57	**76.08 ± 1.76**	88.52 ± 0.26
GPRGNN	vanilla	80.56 ± 3.86	74.75 ± 13.67	78.05 ± 8.04
	SDRF	32.27 ± 5.39	33.81 ± 14.05	77.94 ± 9.66
	FoSR	35.51 ± 7.28	33.92 ± 9.76	81.38 ± 6.35
	Drew	84.53 ± 1.40	72.59 ± 1.60	88.07 ± 0.52
	LGR (Ours)	**86.23 ± 1.08**	**76.61 ± 1.69**	**88.21 ± 0.28**
H_2GCN	vanilla	80.66 ± 3.13	73.62 ± 6.07	86.62 ± 0.39
	SDRF	36.53 ± 14.81	35.25 ± 11.78	85.80 ± 0.54
	FoSR	37.60 ± 15.47	33.68 ± 9.66	85.91 ± 0.70
	Drew	84.63 ± 1.31	73.19 ± 1.78	OOM
	LGR (Ours)	**81.99 ± 2.71**	**75.22 ± 2.24**	**88.31 ± 0.47**
Avg Gain	-	**1.005**	**2.865**	**3.820**

Table 3.4: Results of the ablation study performed on the Squirrel dataset for various components of LGR-GCN in terms of mean and standard deviation of the test accuracy

Dataset	Class Embedding	Graph Rewiring	Label-guided Similarity Matrix	Test Accuracy
Squirrel	✓			24.25±3.94
		✓		30.77±1.57
		✓	✓	53.73±1.61
	✓	✓		60.15±10.71
	✓	✓	✓	**61.14±10.47**

3.3.4 Effect of k_t and k_b on LGR framework

The effectiveness of the rewired neighborhood in the graph hinges on two key variables k_t and k_b. We experiment to study their effect on the graph rewiring and performance of the underlying GCN. We consider one homophilic graph, Citeseer, and one heterophilic graph, Squirrel, to conduct the experiment. Refer to Figure 3.3 for the detailed portrayal of the effect on the performance by choosing different sets of k_t and k_b on both homophilic and heterophilic networks. In the case of Citeseer, when k_b increases, akin to deleting nodes from the neighborhood, it degrades performance. Conversely, when k_t increases, the addition of non-adjacent nodes improves the performance. Citeseer is a homophilic network where deleting nodes from a 1-hop neighborhood incurs information loss, and the addition of similar non-adjacent nodes enriches the neighborhood with more homophilous nodes. Conversely, deletion and insertion favor the model performance in heterophilic networks like Squirrel. An increasing value of k_b removes the non-informative nodes from the neighborhood and an increase of k_t yields the addition of the higher-order neighbors, improving the overall model performance. The experiment underscores the necessity of graph rewiring while dealing with heterophily networks.

3.3.5 Ablation Study

We perform an ablation study on the Squirrel dataset to justify the importance of individual components in the proposed framework. The study is executed by employing LGR with GCN backbone. LGR consists of three key components, namely (1) class embeddings-guided similarity coefficients, (2) generation of learnable class embeddings, and (3) two-stage graph rewiring. The component-wise ablation is presented in Table 3.4, where we report results for every meaningful possible combination of components in the framework. The effect of indi-

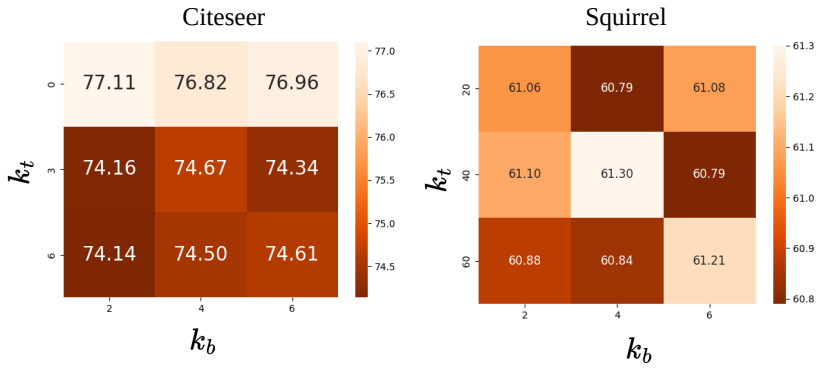


Figure 3.3: The effect of k_b and k_t on the performance of the LGR framework on Citeseer and Squirrel datasets.

Table 3.5: The performance of the various GNN models preprocessed by the LGR framework on heterophilic graphs from (Platonov et al., 2023) are presented. For Roman-empire and Amazon-ratings average test accuracy and for Minesweeper, Tolokers, and Questions average ROC-AUC are reported with the standard deviation. The best results are boldfaced and also average gains are highlighted in green. OOM represents out of memory.

Methods	Variant	Roman-empire	Amazon-ratings	Minesweeper	Tolokers	Questions
GCN	vanilla	77.42 ± 0.67	40.44 ± 1.80	89.29 ± 0.51	79.27 ± 0.89	75.09 ± 1.01
	SDRF	73.25 ± 0.89	38.91 ± 1.92	89.32 ± 0.57	77.87 ± 0.13	75.12 ± 3.45
	FoSR	77.45 ± 0.65	40.26 ± 1.69	89.39 ± 1.34	74.06 ± 2.31	65.72 ± 4.85
	LGR (Ours)	**79.46 ± 0.65**	**42.88 ± 1.35**	**90.45 ± 0.49**	**83.17 ± 1.11**	**76.99 ± 0.90**
GAT	vanilla	54.31 ± 1.31	42.18 ± 4.36	78.32 ± 2.30	76.53 ± 1.74	67.48 ± 5.31
	SDRF	53.65 ± 1.73	42.08 ± 0.54	77.34 ± 1.61	77.81 ± 1.37	67.23 ± 3.67
	FoSR	54.23 ± 1.99	42.20 ± 4.32	78.17 ± 2.22	76.75 ± 1.48	69.59 ± 6.08
	LGR (Ours)	**57.14 ± 1.23**	**42.33 ± 4.34**	**80.43 ± 2.58**	**79.76 ± 2.22**	**70.10 ± 6.55**
APPNP	vanilla	61.09 ± 0.35	41.95 ± 3.11	67.33 ± 1.03	69.07 ± 3.12	52.25 ± 2.18
	SDRF	63.05 ± 0.82	41.24 ± 2.34	69.41 ± 1.31	69.77 ± 3.43	52.21 ± 2.04
	FoSR	61.07 ± 0.42	41.32 ± 3.12	67.37 ± 1.02	69.08 ± 2.52	52.11 ± 2.69
	LGR (Ours)	**66.10 ± 1.51**	**43.99 ± 3.31**	**71.32 ± 1.03**	**70.26 ± 2.33**	**53.39 ± 1.40**
GCNII	vanilla	67.97 ± 17.98	34.31 ± 8.76	77.37 ± 1.45	79.25 ± 1.17	77.79 ± 0.99
	SDRF	66.71 ± 1.31	34.12 ± 7.59	77.14 ± 0.96	79.17 ± 1.52	77.78 ± 0.56
	FoSR	67.92 ± 1.96	34.36 ± 8.99	75.40 ± 1.58	79.27 ± 1.71	77.81 ± 1.05
	LGR (Ours)	**68.11 ± 1.05**	**34.63 ± 8.95**	**79.10 ± 1.77**	**79.58 ± 0.79**	**77.93 ± 0.82**
GPRGNN	vanilla	73.82 ± 0.58	36.93 ± 0.25	86.00 ± 8.63	75.06 ± 1.83	73.21 ± 1.61
	SDRF	75.98 ± 0.75	36.56 ± 2.16	88.18 ± 0.44	77.67 ± 2.07	73.47 ± 1.18
	FoSR	75.70 ± 0.50	37.51 ± 2.07	88.75 ± 1.21	73.51 ± 2.73	73.34 ± 1.71
	LGR (Ours)	**78.44 ± 1.18**	**39.72 ± 2.24**	**89.43 ± 1.29**	**78.03 ± 1.51**	**73.61 ± 1.57**
H_2GCN	vanilla	81.07 ± 0.33	37.10 ± 0.38	90.59 ± 0.41	82.26 ± 1.11	OOM
	SDRF	81.14 ± 0.57	37.91 ± 0.82	90.56 ± 0.33	82.35 ± 1.87	OOM
	FoSR	80.97 ± 0.40	37.12 ± 0.43	90.51 ± 0.38	82.09 ± 1.07	OOM
	LGR (Ours)	**83.88 ± 0.52**	**38.83 ± 2.68**	**90.64 ± 0.35**	**82.81 ± 1.11**	OOM
Avg Gain	-	**2.21**	**1.34**	**1.26**	**1.25**	**0.76**

vidual components yields poor performance. For example, the augmenting class embeddings record degrading performance. The performance gradually improves when multiple components are involved in the framework. The graph rewiring and the class embeddings-guided similarity matrix boost the performance with a good margin. The incorporation of the class embeddings also plays a crucial role in the superior performance of the proposed method. Finally, the best result is obtained when all components are involved in the framework. Therefore, we show the utility of individual components in the proposed framework.

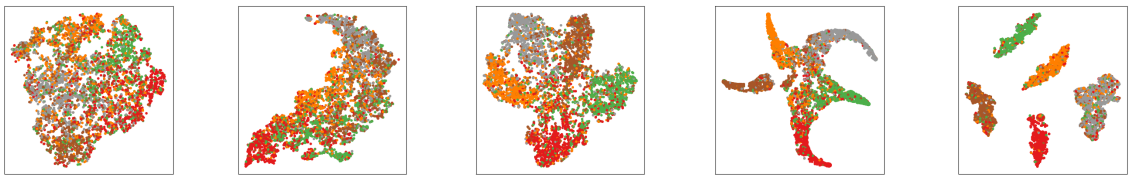


Figure 3.4: Node embeddings of the Squirrel dataset generated by different popular baselines are presented. Among all, the LGR-GCN generates more well-formed clusters with better class boundaries compared to others.

3.3.6 Comparative Study of the Node Embeddings

We perform a comparative study on the node embeddings of the Squirrel dataset with our proposed approach, LGR-GCN, along with four other state-of-the-art methods: GCN, Simp-GCN, RAW-GNN, and BM-GCN. The node embeddings are generated using t-SNE with the provided hyperparameters. Figure 3.4 denotes the node embeddings where individual colors denote the ground-truth labels. As GCN is well-suited to homophilic datasets, the node embedding contains nodes with mixed colors. Thus, GCN forms a poor cluster structure and cluster boundaries are not visible. More advanced methods like Simp-GCN perform better than GCN, but they are still incapable of capturing cluster structure. In the case of RAW-GNN or BM-GCN, both exhibit good cluster formation without visible clustering boundaries. In our method, LGR-GCN preserves the cluster structure with a high separation among the clusters, which guarantees the effectiveness of our approach.

3.3.7 Effect on Homophily Ratio after Graph Rewiring

The graph rewiring is the key module of our proposed framework, which makes the neighborhood more homophily-dominant. We conduct experiments on four heterophilic datasets, Squirrel, Actor, Texas, and Cornell, and three homophilic datasets, Cora, Citesser, and Pubmed, respectively. After applying the rewiring procedure, the edge homophily ratios of all four heterophilic datasets increased, which is evident in Figure 3.5. The corresponding homophily ratio remains almost unaltered with the rewiring process. The two-stage rewiring strategy enables the inclusion of similar nodes and the exclusion of dissimilar nodes, increasing the homophily ratio. For the homophilic networks, the LGR framework introduces more similar nodes and discards dissimilar nodes, which makes the homophily ratio higher. Therefore, the proposed rewiring technique successfully transforms the neighborhoods into more homophily-oriented. The rewired networks become more suitable for applying the GNNs.

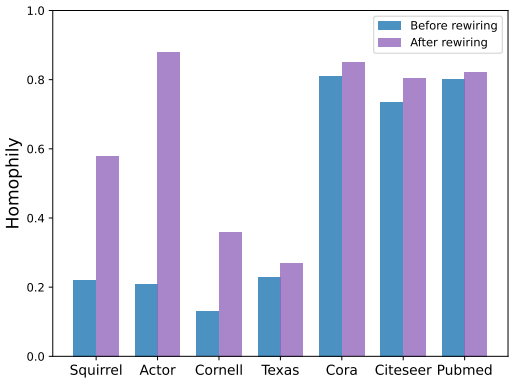


Figure 3.5: The effect on homophily ratio after applying LGR on four heterophilic and three homophilic graphs are presented. LGR framework effectively converts the heterophilic neighborhoods into more homophilic ones and also maintains the homophily neighborhoods for the respective homophilic graphs.

3.3.8 Visualization of Class Compatibility Matrix

We presented class compatibility matrices of both datasets, Citeseer and Squirrel in Figure 3.6. The class compatibility matrix denotes the connection patterns among the different sets of classes in the graph. The heatmap for Citeseer suggests that the higher values along the diagonal elements are compared to other non-diagonal entries. This signifies the rewired graph contains a higher number of intra-edge connections. On the contrary, a different trend can

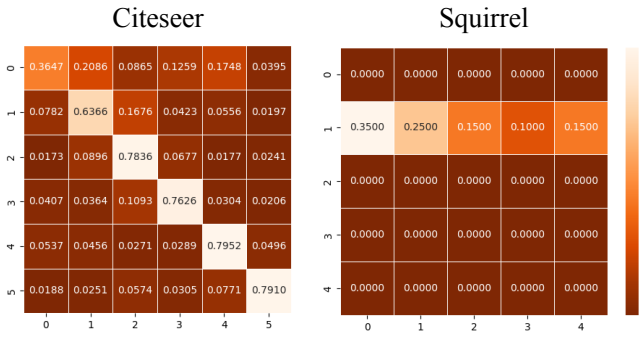


Figure 3.6: The heatmaps for the Citeseer and Squirrel datasets are demonstrated. In both cases, the number of inter-class edges reduces and intra-class edges increase, which highlights the purpose of the LGR framework.

be observed in the compatibility matrix of Squirrel. The LGR framework removes the major portion of inter-class edges and efficiently rewires the graph to emphasize on the intra-class edges. Additionally, almost all non-diagonal elements are zero which portrays the declining numbers of inter-class edges in the rewired graph. However, the framework retains some of the inter-class edges which seems to be informative. Hence, the overall increasing number of intra-class edges assists in ensuring effective message propagation across the graph.

3.3.9 Robustness of LGR framework

We verify the robustness of the model by infusing two types of noise described as the following

Feature-level Noise A random Gaussian noise with mean 0 and standard deviation 0.01 is added with the node features of the input graph. The following equation represents the feature-level noise addition

$$X_N = X + n_f N(0, 0.01), \tag{3.17}$$

where n_f denotes the weight of the noise added to the node features.

Structure-level Noise There are two different ways to incorporate structure-level noise in the graph such as (1) the insertion of edges within the non-adjacent nodes and (2) the deletion of the edges from the adjacent nodes. Define two fractions namely f_i and f_d indicate the fraction of the total edges to be deleted or inserted respectively. Let, $|\mathcal{E}|$ be the total number of edges. Therefore, after edge insertion (deletion) the new set of edges is $\mathcal{E}_N = \mathcal{E} \cup \mathcal{E}_i$ or $\mathcal{E}\ \mathcal{E}_d$ where $\mathcal{E}_i, \mathcal{E}_d$ denotes the set of edges added and deleted respectively. Furthermore, we have

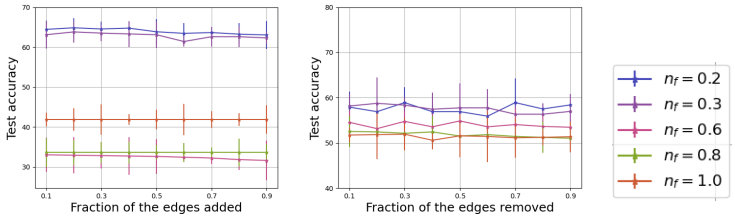


Figure 3.7: Performances of LGR on Squirrel paired with GCN, with the variation of both feature-level and structure-level noise, are presented. The results suggest that LGR is robust to structural noise rather than feature-level noise.

$|\mathcal{E}_i| = f_i\mathcal{E}$ and $|\mathcal{E}_d| = f_d\mathcal{E}$.

Experimental Setup: We applied two types of noise simultaneously on the Squirrel dataset to validate the resilience of the LGR framework paired with GCN. For the feature-level noise, we vary n_f, and for every level of feature-level noise, we perform experiments on edge insertion and deletion separately with varying f_i, f_d. Refer to Figure 3.7 for a detailed view of the response of LGR to different noise levels. LGR exhibited resilience to the structural noise, but it showed some weaker performances for the feature-level noise. As our primary objective is to rewire the graph, it is justified for such performance with noisy features.

Table 3.6: The performance of various GNN models coupled with the LGR framework applied to six heterophilic graphs and performances are analyzed in terms of F1-score and AUROC. The best results are boldfaced.

Model	Chameleon		Squirrel		Actor		Cornell		Texas		Wisconsin	
	F1-score	AUROC	F1-score	AUROC	F1-score	AUROC	F1-score	AUROC	F1-score	AUROC	F1-score	AUROC
GCN	0.44±0.02	0.72±0.01	0.29±0.01	0.58±0.01	0.28±0.01	0.56±0..01	0.47±0.05	0.63±0.03	0.61±0.03		0.55±0.06	0.60±0.06
GCN + LGR	**0.70±0.07**	**0.82±0.04**	**0.61±0.09**	**0.77±0.06**	**0.60±0.01**	**0.78±0.05**	**0.84±0.07**	**0.90±0.04**	**0.84±0.06**	-	**0.87±0.05**	**0.91±0.05**
GAT	0.30±0.02	0.75±0.01	0.23±0.03	0.59±0.08	0.28±0.01	0.59±0.09	0.45±0.05	0.69±0.05	0.59±0.05		0.57±0.05	0.74±0.04
GAT + LGR	**0.70±0.07**	**0.83±0.04**	**0.61±0.09**	**0.78±0.06**	**0.60±0.10**	**0.80±0.05**	**0.85±0.05**	**0.92±0.04**	**0.85±0.06**	-	**0.88±0.03**	**0.92±0.03**
APPNP	0.28±0.05	0.61±0.02	0.22±0.01	0.54±0.03	0.25±0.01	0.53±0.01	0.34±0.11	0.62±0.05	0.56±0.11		0.50±0.09	0.60±0.05
APPNP + LGR	**0.69±0.07**	**0.83±0.04**	**0.61±0.09**	**0.78±0.06**	**0.60±0.10**	**0.80±0.06**	**0.80±0.05**	**0.89±0.04**	**0.84±0.07**	-	**0.85±0.03**	**0.91±0.05**
GPRGNN	0.35±0.08	0.62±0.03	0.27±0.02	0.56±0.03	0.34±0.07	0.65±0.04	0.65±0.12	0.83±0.09	0.75±0.09		0.75±0.11	0.88±0.09
GPRGNN + LGR	**0.68±0.07**	**0.82±0.05**	**0.58±0.12**	**0.78±0.06**	**0.61±0.1**	**0.79±0.06**	**0.85±0.06**	**0.90±0.04**	**0.87±0.06**	-	**0.86±0.03**	**0.91±0.04**
GCNII	0.41±0.02	0.69±0.01	0.31±0.01	0.61±0.01	0.34±0.06	0.64±0.01	0.53±0.11	0.74±0.06	0.62±0.05		0.56±0.11	0.69±0.11
GCNII + LGR	**0.70±0.07**	**0.83±0.05**	**0.61±0.09**	**0.78±0.05**	**0.61±0.1**	**0.79±0.05**	**0.85±0.06**	**0.91±0.03**	**0.85 ±0.05**	-	**0.89±0.03**	**0.94±0.03**
H2GCN	0.42±0.04	0.71±0.01	0.31±0.01	0.58±0.01	0.30±0.02	0.58±0.02	0.59±0.11	0.74±0.05	0.60±0.04		0.61±0.09	0.71±0.09
H2GCN + LGR	**0.70±0.07**	**0.84±0.04**	**0.60±0.09**	**0.78±0.06**	**0.61±0.09**	**0.80±0.05**	**0.85±0.05**	**0.91±0.03**	**0.84±0.07**	-	**0.89±0.04**	**0.93±0.03**

3.3.10 Sensitivity Analysis on Various GNN Architectures

LGR framework is applied to the Squirrel dataset coupled with different configurations of GNN architectures as presented in Table 3.7. We include an improved version of GCN and GAT with different attention heads to showcase the performance variations. Increasing self-

loops in GCN (A+2I) deteriorated the performance. On the contrary, increasing the number of attention heads yields a performance gain.

Table 3.7: Performance of LGR paired with different GNNs configurations on Squirrel is presented.

GNN Configurations	Accuracy
GCN (A + I)	60.73 ± 6.36
GCN (A+2I)	55.47 ± 5.94
GAT (h=4)	61.24 ± 4.89
GAT (h=8)	61.00 ± 3.82
GAT (h=16)	61.12 ± 4.85

3.3.11 Comparative Study on Multiple Evaluation Metrics

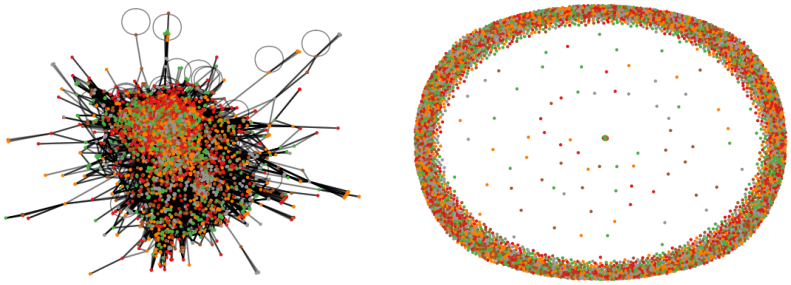
We conducted a study to assess the performance of the LGR framework by considering two other metrics, namely F1-score and AUROC, apart from test accuracy. LGR, coupled with six GNN models, was applied to six standard heterophilic graphs. Refer to Table 3.6 to visualize the performance analyses of LGR. The underlying GNN models performed better for both metrics when the input graph was rewired with the LGR framework. Note that the standard test mask of Texas does not contain nodes that cover all class labels; therefore, AUROC estimation is not feasible for the Texas dataset.

3.3.12 Visualization of the Rewired Graph

An experiment is performed to study the effects of applying the two-stage framework on Squirrel. Refer to Figure 3.8 to visualize the difference in the edge connectivity between the input graph and the rewired graph. Squirrel is a dense graph with a lower homophily ratio. The rewiring framework effectively drops all irrelevant edges and rewires the network topology, which seems conducive to message passing.

3.3.13 Visualization of Degree Distribution and Clustering Coefficients

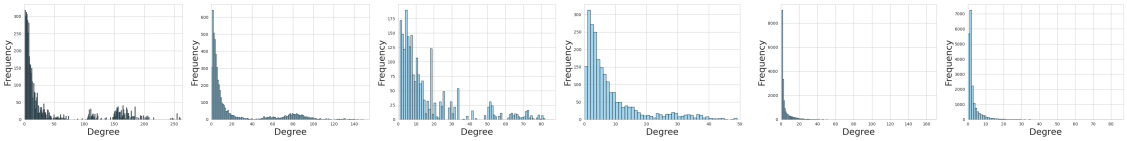
We studied the alteration of the degree distribution of the input graph and the rewired graph after applying LGR on three datasets Squirrel, Chameleon, and Pubmed. Refer to Figure 3.9



(a) Input graph (b) Rewired graph

Figure 3.8: The network structures of Squirrel before and after rewiring, respectively, are presented. The dense graph is converted into a more sparse graph, dropping unimportant edges.

to visualize the impact on the node degrees after the rewiring process. The direct comparisons with the input degree distribution suggest the sparsification of the graph, underlining the removal of heterophilic or uninformative edges across the network. Similar trends are also evident in the case of average clustering coefficients of the graphs. The detailed clustering coefficients for nine graphs are provided in Table 3.8. Lastly, a decrease of both degree distribution and clustering coefficients highlights the dropping of irrelevant edges from the heterophilic networks and rewires them to be suitable for effective message-passing.



(a) Squirrel w/o LGR (b) Squirrel with LGR (c) Chameleon w/o LGR (d) Chameleon with LGR (e) Pubmed w/o LGR (f) Pubmed with LGR

Figure 3.9: The comparative study of degree distributions before and after applying the LGR framework is presented for Squirrel, Chameleon, and Pubmed, respectively.

3.4 Conclusion

In this work, we introduce an adaptive pre-processing framework named LGR (Label-guided Graph Rewiring) to rewire both homophilic and heterophilic graphs, converting them into more homophilic. The resultant rewired graph can then be effortlessly integrated as input for various GNN models, thus ensuring the inherent flexibility of our proposed framework. The architecture of LGR is structured around three fundamental steps. Firstly, it involves the

Table 3.8: The average clustering coefficients before and after rewiring with LGR are demonstrated for six heterophilic and three homophilic graphs, respectively.

Dataset	Avg. Clustering Coeff.	
	w/o LGR	with LGR
Squirrel	0.337	0.158
Chameleon	0.376	0.172
Actor	0.0542	0.019
Cornell	0.097	0.029
Texas	0.111	0.038
Wisconsin	0.113	0.044
Cora	0.241	0.094
Citeseer	0.141	0.050
Pubmed	0.060	0.022

generation of pseudo-node labels as well as class representations by harnessing the autoencoders to acquire class representations. Both are combined to produce class representations to guide the re-weighting of conventional similarity scores computed between node pairs. These recalibrated scores then drive a two-stage process of graph rewiring involving both node deletion and node insertion. Concurrently, the class embeddings are augmented with the raw node features to enrich with class-specific insights. Subsequently, the updated node features and the rewired graph collectively serve as inputs for GNN models, seamlessly facilitating downstream tasks. We experimented on 11 heterophilic graphs and 3 homophilic graphs. We further offer theoretical analyses and study the robustness of the LGR framework. Looking ahead, the potential of this framework can be extended to explore its effectiveness in addressing diverse downstream tasks that necessitate graph rewiring. Furthermore, future endeavors should focus on refining the graph topology rewiring process to enhance the precision and efficacy of the framework.

3.5 Appendix A

Theorem 3.1 For any vertex v in the $\mathcal{G}$ with neighborhood is $\mathcal{N}_v$ with edge homophily ratio h and $|\mathcal{N}_v| = d$. Define m as the number of neighbors with identical class labels as the label of v. After applying the LGR framework. the updated neighborhood became $\mathcal{N}'_v$ with modified edge homophily ratio h'. Assume h_t and h_b are the respective probabilities for selecting nodes of the class label of v according to the similarity scores obtained from the MLP and autoencoder. Consider k_t and k_b are the number of nodes chosen for inclusion

and exclusion, respectively. Therefore, if $\frac{k_t}{k_b} > \left(\frac{dh_b - m}{dh_t - m}\right)$, then we have $h' > h$ implies the homophily ratio will increase for the rewired neighborhood in $\mathcal{G}$.

Proof. Consider the neighborhood of a node v as $\mathcal{N}_v$ with the degree of d and we have $m = |\{u : y_u = y_v | u \in \mathcal{N}_v\}|$. The edge homophily of node v can be defined as

$$h = \Pr[y_u = i | y_v = j, \forall u \in \mathcal{N}_v], \tag{3.18}$$

where y_v denotes the class label of node v. After applying LGR on $\mathcal{G}$, the two-stage rewiring will update the neighborhood as $\mathcal{N}'_v$. Assume, the two-stage rewiring framework will remove the k_b number of nodes and will add the k_t number of nodes in the neighborhood. The current degree of v is $d' = |\mathcal{N}'_v| = d - k_b + k_t$. The updated neighborhood can be expressed as

$$\mathcal{N}'_v = \mathcal{N}_v \setminus \mathcal{N}_b \cup \mathcal{N}_t, \tag{3.19}$$

where $\mathcal{N}_b$ and $\mathcal{N}_t$ contain dissimilar and similar set of nodes with $|\mathcal{N}_b| = k_b$ and $|\mathcal{N}_t| = k_t$ respectively. The nodes are selected based on the similarity scores obtained by combining the feature similarity scores from MLP and the autoencoder. Again the homophily of v can be written as

$$h = \Pr[y_u = i | y_v = j, \forall u \in \mathcal{N}_v] = \frac{m}{d}. \tag{3.20}$$

Assume that the probability of sampling a node with an identical label as v from the set $\mathcal{N}_t$ is h_t. Similarly, the probability of sampling of the same from the set $\mathcal{N}_b$ is h_b. The statements can be represented by the following equations.

$$\begin{aligned} \mathrm{p}^s_\psi(y_u = y_v | u \in \mathcal{N}_t) &= h_t \\ \mathrm{p}^d_\psi(y_u = y_v | u \in \mathcal{N}_b) &= h_b, \end{aligned} \tag{3.21}$$

where ψ is the learnable parameters of the distribution depending on the MLP and autoencoder. Let's call $H_v = \{u : y_u = y_v | u \in \mathcal{N}'_v\}$ denotes the set of nodes with identical class labels of v. The total number of nodes of the class label y_v in the newly constructed neighborhood will be $|H_v|$. Therefore, the expected number of nodes of similar label as v in $\mathcal{N}'_v$

is:

$$\begin{aligned}\mathbb{E}[|H_v|] &= m + \mathbb{E}_{X_s \sim \mathrm{p}^s_\psi}[X_s] - \mathbb{E}_{X_d \sim \mathrm{p}^d_\psi}[X_d] \\ &= m + \sum_{i \in \mathcal{N}_t} h_t - \sum_{j \in \mathcal{N}_b} h_b \\ &= m + h_t k_t - h_b k_b,\end{aligned} \tag{3.22}$$

where X_s, X_d are the random variables for selecting similar and dissimilar nodes respectively. The total number of nodes in the newly formed neighborhood will be:

$$d' = |\mathcal{N}'_v| = d + k_t - k_b. \tag{3.23}$$

The new homophily ratio h' can be presented as follows:

$$\begin{aligned}h' &= \frac{\mathbb{E}[|H_v|]}{|\mathcal{N}'_v|} \\ &= \frac{m + h_t k_t - h_b k_b}{d + k_t - k_b}\end{aligned} \tag{3.24}$$

The change in the edge homophily is

$$\begin{aligned}\Delta h &= h' - h \\ &= \frac{m + h_t k_t - h_b k_b}{d + k_t - k_b} - \frac{m}{d} \\ &= \frac{d(h_t k_t - h_b k_b) - m(k_t - k_b)}{d(d + k_t - k_b)}\end{aligned} \tag{3.25}$$

Assuming $d + k_t > k_b$ results in a positive denominator, we will be concerned with the numerator. Consider the condition $\frac{k_t}{k_b} > \left(\frac{dh_b - m}{dh_t - m}\right)$, the following algebraic manipulation can be applied as following,

$$\begin{aligned}&\frac{k_t}{k_b} > \left(\frac{dh_b - m}{dh_t - m}\right) \\ &k_t(dh_t - m) > k_b(dh_b - m) \\ &dh_t k_t - mk_t > dh_b k_b - mk_b \\ &dh_t k_t - dh_b k_b > mk_t - mk_b \\ &d(h_t k_t - h_b k_b) - m(k_t - k_b) > 0\end{aligned} \tag{3.26}$$

The above inequality confirms the positive numerator in Δh implies $\Delta h > 0$, affirming the increase in the homophily ratio in $\mathcal{N}'_v$. The mentioned condition signifies the impact of selecting the proper number of similar nodes and discarding dissimilar nodes, ensuring the increasing homophily in the rewired graph. □

Remark 3.1. *As per Eq 3.20, for the homophily ratio h, we can substitute $m = dh$ in the condition $\frac{k_t}{k_b} > \left(\frac{dh_b - m}{dh_t - m}\right)$ will resulting as following,*

$$\frac{k_t}{k_b} > \left(\frac{dh_b - m}{dh_t - m}\right) = \frac{h_b - h}{h_t - h} \tag{3.27}$$

if the degree of heterophily of the network is $h' = 1 - h$, then the condition will be as follows,

$$\begin{aligned} \frac{k_t}{k_b} &> \frac{h_b - h}{h_t - h} \\ &= \frac{h_b + h' - 1}{h_t + h' - 1} \end{aligned} \tag{3.28}$$

For the optimal performance of the LGR framework, we should have $h_t \neq 1 - h'$. Furthermore, if $h_b > h_t$, then we $k_t > k_b$ indicate the increase in the neighborhood size, resulting in a more dense graph. Otherwise, the graph became sparser when $h_t > h_b$ holds. The sampling probabilities p_t and p_b predominantly depend on learning the parameters of both MLP and the autoencoder. For a sparser graph containing a relatively lesser number of heterophilic edges, the condition $h_t > h_b, h_t \neq 1 - h'$ should hold inevitably. Therefore, the aforementioned condition presents the optimality of MLP and autoencoder when the LGR framework seems to perform efficient rewiring, irrespective of the degree of heterophily of the input graph.

3.6 Appendix B

3.6.1 Details of the New Heterophilic Graphs

There are five heterophilic datasets proposed by (Platonov et al., 2023) that are discussed as follows.

Roman-empire The dataset is prepared from the longest article, Roman Empire, from English Wikipedia. The node of the graph represents each word (non-unique) in the article. Nodes are connected if either two words are adjacent or connected in the dependency tree

of the sentence. Therefore, the graph is chained and shortcuts exist depending don't the syntactic dependencies between the words. The node labels denote the syntactic roles, and a total of 18 classes are there. The node features are generated from the fastText word embeddings.

Table 3.9: The details of the five heterophilic graphs proposed by (Platonov et al., 2023) are presented.

Datasets	Roman-empire	Amazon-ratings	Minesweeper	Tolokers	Questions
# Nodes	22662	24492	10000	11758	48921
# Edges	32927	93050	39402	519000	153540
# Classes	18	5	2	2	2
# Features	300	300	7	10	301
Average degree	2.91	7.6	7.88	88.28	6.28
# Isolated nodes	0	3495	1	936	17761
Homophily ratio	0.05	0.38	0.68	0.59	0.84

Amazon-ratings This dataset is based on the Amazon product co-purchasing network metadata dataset from SNAP datasets. Nodes are products (books, music CDs, DVDs, VHS video tapes), and edges connect products that are frequently bought together. The task is to predict the average rating given to a product by reviewers. We grouped possible rating values into five classes. For node features, we use the mean of fastText embeddings for words in the product description. To reduce the size of the graph, the curators consider only the largest connected component of the 5-core of the graph.

Minesweeper This dataset is inspired by the Minesweeper game, and it is the only synthetic dataset in this benchmark. The graph is a regular 100x100 grid where each node (cell) is connected to eight neighboring nodes (with the exception of nodes at the edge of the grid, which have fewer neighbors). 20% of the nodes are randomly selected as mines. The task is to predict which nodes are mines. The node features are one-hot-encoded numbers of neighboring mines. However, for randomly selected 50% of the nodes, the features are unknown, which is indicated by a separate binary feature. The structure of this graph is significantly different from the other datasets due to its regularity. The average degree is 7.88 since almost all the nodes have exactly eight neighbors.

Tolokers This dataset is based on data from the Toloka crowdsourcing platform. The nodes represent tolokers (workers) who have participated in at least one of 13 selected projects. An edge connects two tolokers if they have worked on the same task. The goal is to predict which tolokers have been banned in one of the projects. Node features are based on the worker's

profile information and task performance statistics. This graph has 11.8K nodes, with an average degree of 88.28. Thus, the graph is significantly denser than all the other graphs. About 22% of the tolokers in this dataset have been banned.

Questions This dataset is based on data from the question-answering website Yandex Q. Nodes are users, and an edge connects two nodes if one user answered the other user's question during a one-year time interval (from September 2021 to August 2022). To restrict the size of the dataset, curators considered only users interested in the topic 'medicine'. The task is to predict which users remained active on the website (were not deleted or blocked) at the end of the period. For node features, the mean of fastText embeddings (Grave et al., 2018) for words is used in the user description. Since some users (15%) do not have descriptions, an additional binary feature is used that indicates such users.

Chapter 4

Performance Analysis of Low-pass Filters on Heterophilic Networks: A Deeper Insights

Summary

Graph heterophily poses a formidable challenge to the performance of Message-passing Graph Neural Networks (MP-GNNs). The familiar low-pass filters like Graph Convolutional Networks (GCNs) face performance degradation, which can be attributed to the blending of the messages from dissimilar neighboring nodes. The performance of the low-pass filters on heterophilic graphs still requires an in-depth analysis. In this context, we update the heterophilic graphs by adding a number of self-loops and parallel edges. We observe that the eigenvalues of the graph Laplacian decrease and increase, respectively, by increasing the number of self-loops and parallel edges. We conduct several studies regarding the performance of GCN on various benchmark heterophilic networks by adding either self-loops or parallel edges. The studies reveal that the GCN exhibited either increasing or decreasing performance trends on adding self-loops and parallel edges. In light of the studies, we established connections between the graph spectra and the performance trends of the low-pass filters on the heterophilic graphs. The graph spectra characterize the essential intrinsic properties of the input graph, like the presence of connected components, sparsity, average degree, cluster structures, etc. Our work is adept at seamlessly evaluating graph spectrum and properties by observing the performance trends of the low-pass filters without pursuing the costly eigenvalue decomposition. The theoretical foundations are also discussed to validate the impact of adding self-loops and parallel

edges on the graph spectrum.

4.1 Introduction

Graph Neural Networks (Scarselli et al., 2008) made remarkable strides by achieving impeccable performance in graph-structured data. The key reason behind the immense performance superiority is the message passing (MP) framework, which enables the exchange of messages between the adjacent nodes. Before judging the narrative of the success story of the MP framework, let us first mention that graphs can be broadly categorized into two classes such as (1) *homophilic* graphs where adjacent nodes share identical class labels, and (2) *heterophilic* graphs where adjacent node labels are different from each other. The prowess of MP is mostly observed in homophilic graphs because of the tendency to blend messages from similar types of neighbors. In contrast, several studies (Zhu et al., 2020), (Zhu et al., 2021a), (He et al., 2022), (Suresh et al., 2021), (Wang et al., 2022b) suggest that the MP framework shows exacerbating performances on heterophilic graphs due to the influence of dissimilar messages received from neighbors.

A well-known study (Nt and Maehara, 2019) reveals that all MP-GNNs such as GCN (Kipf and Welling, 2016), GraphSage (Hamilton et al., 2017), GAT (Veličković et al., 2017), SGC (Wu et al., 2019), etc, which smooth the features of adjacent nodes, are low-pass filters. The low-pass filters successfully convert the features of the connected nodes into more similar ones compared to the features of other non-adjacent nodes. This narrates the key reason behind the successful application of low-pass filters on homophilic graphs. Applying low-pass filters on the heterophilic graphs often leads to a degradation in performance as a result of smoothing the features of the dissimilar adjacent nodes. Therefore, analyzing the performance of low-pass filters on heterophilic graphs requires more in-depth scrutiny. The low-pass filters are designed to amplify the coefficients of lower frequencies in the graph spectrum, representing the eigenvalues of the symmetrically normalized graph Laplacian. In homophilic graphs, the smoothing of node features occurs due to the amplification of the lower frequencies of the graph spectrum. In the case of heterophilic graphs, high-pass filters sharpen the node features by amplifying the higher frequencies of the graph spectrum.

The graph spectrum entails significant information regarding structural patterns such as

connected components, community structures, isolated nodes, sparsity, etc. For instance, if the set of eigenvalues contains sufficient zeros, then the network will contain more connected components or isolated nodes. The network will contain weakly (strongly) connected components if the spectrum has a higher number of low frequencies (high frequencies). Therefore, the dissection of the spectrum yields profound information relating to the spatial properties of the graphs. In this work, we investigate the dependency of the graph structure on the performance of the low-pass filters applied to heterophilic networks. We also attempt to uncover the structural properties of the existing heterophilic graphs from their spectrum. The graph spectrum is typically obtained with expensive eigenvalue decomposition, which imposes an unnecessary computational burden or can be infeasible in some real-world scenarios. Therefore, we seek to devise an efficient avenue that significantly addresses the computational overhead of evaluating the graph spectrum.

Table 4.1: Four possible categories A, B, C, and D are presented. Each category depends on the performance trends of a low-pass filter when either self-loops or parallel edges are added to the heterophilic graph.

	Rewiring		
	Self-loop	Parallel edge	Category
Performance of LPF	Increasing (↑)	Increasing (↑)	A
	Increasing (↑)	Decreasing (↓)	B
	Decreasing (↓)	Increasing (↑)	C
	Decreasing (↓)	Decreasing (↓)	D

We aim to bridge the gap by offering two simple strategies that update the graph topology by incorporating self-loops and parallel edges. After the alteration of edge connections, Graph Convolution Network (GCN), a recognized and well-adopted low-pass filter, is applied to the updated graph. We observe some interesting patterns in the performance trends of GCN when the number of self-loops or parallel edges gradually increases. The performance either monotonically improves or degrades by adding either self-loops or parallel edges. Therefore, we can have four distinct combinations of performance trends. Each combination is tagged with a category name, which is mentioned in the Table 4.1. The category assignment is purely dependent on the combination of performance trends in both self-loops and parallel edge addition. For instance, if GCN shows an increasing trend on self-loop addition and a decreasing trend on parallel edge addition, then the underlying dataset is categorized as

"B". Furthermore, for parallel edge addition, if the trend changes to increasing, then the category will change to "A". Importantly, every category delineates a particular set of characteristics regarding the spectrum of the input graph. The performance trend of GCN can be attributed to the underlying parity between the lower and higher frequencies present in the graph spectrum. Each performance trend offers a unique insight into the parity of lower and higher frequencies, which directly links with the spatial edge connectivity of the network. We also observe that the eigenvalues of the normalized graph Laplacian decrease with the addition of self-loops in the network. Conversely, the addition of parallel edges enhances the eigenvalues of the graph Laplacian. The shrinking or expansion of the graph frequencies leads to a specific performance trend, reflecting the distribution of eigenvalues (or frequencies) in the graph spectrum. In this context, we consider 17 benchmark heterophilic graphs to predict their spectrum characteristics by observing the performance trends of GCN after adding either self-loops or parallel edges. We also offer the theoretical underpinnings of the shrinking or expansion of the eigenvalues. The phenomena are also observed empirically on the random Erdős-Rény graphs.

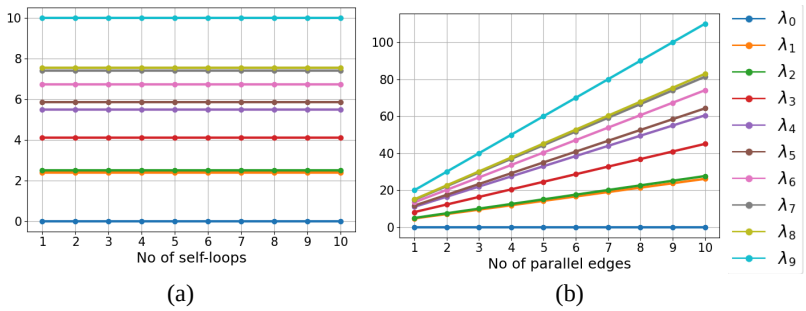
Contribution Our contributions are briefly outlined as follows,

- We provide deeper analyses pertaining to the performance of GCN, a low-pass filter, applied to 17 benchmark heterophilic graphs. We modify the graph structure with the addition of self-loops and parallel edges separately. GCN is applied to the updated graphs and observes the performance trends. We categorize the performance trends into four categories and each graph lies in one of the four categories.

- The categorization of performance trends leads to the identification of characteristics of the graph spectrum. Different performance trends underscore the various patterns of the spectrum which reveals the properties of the networks like connected components, community structure, sparsity, etc.

- We also observe that the frequencies in the graph spectrum decrease with the addition of self-loops and frequencies increase with the addition of parallel edges. We also establish a connection between the performance trends of GCN and the shrinking or expansion of the frequencies of the graph spectrum.

- We offer extensive theoretical underpinnings for the shrinking or expansion of the frequencies of the graph spectrum with the addition of self-loops and parallel edges in the network. The detailed proofs and derivations are discussed in the Appendix.



(a) (b)

Figure 4.1: The changes in the eigenvalues of the unnormalized graph Laplacian are presented with the addition of (a) self-loops and (b) parallel edges, respectively. The eigenvalues remain unaltered with the addition of self-loops. On the contrary, the unconstrained growth of eigenvalues is observed with the addition of parallel edges.

4.2 Background

The spectral analysis on the graphs gains traction due to its ability to unravel the relationship between frequencies and the spatial connectivity of the networks. Work like (Ortega, 2022) introduces the key ingredients of signal processing like Graph Fourier transforms, frequencies, and the design of the filters for the graph-structured data. Another line of work (Ortega et al., 2018) deals with the intricate details of graph signal processing by shedding light on the spectrum analysis from the perspective of the graph Laplacian, extensive real-world applications, and the underlying challenges. The exploitation of spectral analysis in the discrete domain is rigorously harnessed by (Sandryhaila and Moura, 2014). Another mode of work (Tremblay et al., 2018) offers the prospect of designing versatile filter banks and spectral wavelets on graph-structured data. The design of efficient and localized convolutional filters becomes an inevitable area of research which is initiated by (Defferrard et al., 2016).

A large pool of well-adopted GNNs like GCN, GraphSage, GAT, SGC, etc are recognized as potential low-pass filters. The fact is first asserted by (Nt and Maehara, 2019). Chen *et al.* (Chen et al., 2023b) established the bridge to fill the gap between the spatial and spectral

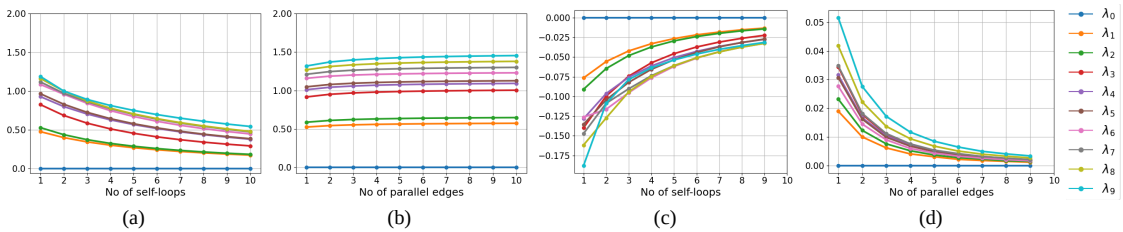


Figure 4.2: (a) Eigenvalues of $\tilde{L}$ exhibit a decreasing trend with the increase in addition of self-loops, and (b) eigenvalues show an increasing trend with the addition of parallel edges. The corresponding changes in the eigenvalues with the addition of self-loops and parallel edges are demonstrated in (c) and (d) respectively.

properties of the prevalent graph neural networks. AutoGCN (Wu et al., 2022) proposes a variant of GCN equipped with low-pass, high-pass, and band-pass filters which automatically adjusts magnitude depending on the homophily or heterophily of the input graph. In this connection, another prominent work AdaGNN (Dong et al., 2021) learns an adaptive filter that spans across multiple layers, capturing the varying node frequencies to improve node embeddings. Taking the cue from above, FAGCN (Bo et al., 2021) firstly proposed one low-pass and one high-pass filter. The proposed filters automatically maintain the balance of collecting information from both homophilic and heterophilic graphs. Designing novel convolutional filters became an enticing avenue which is evident by (Bianchi et al., 2021) which employed an auto-regressive moving average filter (ARMA) filter by replacing polynomial filters to achieve a more robust, flexible frequency response. Another work EGC (Tailor et al., 2021) employed the effectiveness of isotropic message passing, where the message function only depends on the source nodes, over the anisotropic message passing. EGC allows the involvement of multiple learnable filters to achieve a spatially evolving frequency response. (Zhu and Koniusz, 2021) leverages the use of Markov Diffusion Kernel to obtain a filter that maintains a delicate balance between harnessing information of local and global context for each node across the network. On the other side, (**?**) identified the gap for spectral analysis on signed graphs. To mitigate the gap, they proposed two signed GNNs that retain low-pass and high-pass information respectively. Compatible Label Propagation (CLP) (**?**) combines a class compatibility matrix and label propagation to improve node classification on heterophilic graphs. Very recently, an inductive spectral filter SLOG (**?**) was proposed which considers real-valued order polynomial filters. SLOG also combines subgraph sampling

in the spatial domain and signal processing in the spectral domain. Another orthogonal work (**?**) studied the effect on the eigenvalues of the adjacency matrix but no theoretical analysis was provided on the graph Laplacian or graph spectrum. Moreover, they offered analyses on the eigenvalues of the adjacency matrix for the addition of any edges. In contrast, our work predominantly presents systematic addition of self-loops and parallel edges which offers insights into the graph spectra in the context of heterophilic graphs.

4.3 A Deeper Investigation

4.3.1 Notations

Consider an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$ where $\mathcal{V}$ denotes the set of vertices with $|\mathcal{V}| = n$, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $X \in \mathbb{R}^{n \times d}$ is the feature matrix contains d-dimensional feature vectors. We define graph Laplacian $L = D - A$ and symmetrically normalized Laplacian as $\tilde{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = \mathrm{I} - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. Also, augmenting self-loops the normalized Laplacian will be $\tilde{L} = \mathrm{I} - \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ where $\tilde{A} = A + \mathrm{I}$ and $\tilde{D} = D + \mathrm{I}$.

4.3.2 Preliminaries on Spectral Graph Theory

The spectral analysis of graphs (**?**) revolves around understanding the characteristics of eigenvalues and eigenvectors of the symmetrically normalized graph Laplacian $\tilde{L}$. The analysis entails that the eigenvalues and eigenvectors of $\tilde{L}$ represent the Fourier frequency and Fourier modes. Suppose the eigendecomposition on Laplacian yields us $\tilde{L} = U \Sigma U^\top$ where columns of U represent the eigenvectors and Σ is a diagonal matrix containing the eigenvalues. Let a signal $x \in \mathbb{R}^n$ act on the nodes in the graph, then the Fourier transformation of x is presented as $\hat{x} = U^\top x$. The inverse Fourier transform can be formulated as $x = U\tilde{x}$. Thus, for any filter g, the graph convolution between g and x is estimated as:

$$g * x = U((U^\top g) \odot (U^\top x)) = U \tilde{G} U^\top x, \tag{4.1}$$

where $\odot$ denotes the element-wise vector multiplication and $\tilde{G} = \mathrm{diag}\{\tilde{g}_1, \cdots, \tilde{g}_n\}$. Each $\tilde{g}_i$ denotes the spectral filter coefficient. A well-known fact is that $\tilde{L}$ has the eigenvalues lie in $[0, 2]$. Let us categorize the set of eigenvalues as $\lambda_{<1}$ or *lower frequencies* which are strictly

smaller than 1 and $\lambda_{\geq 1}$ or *higher frequencies*, which are greater than or equal to 1. If a filter amplifies the coefficients of $\lambda_{<1}$, then it acts as a low-pass filter, and on the contrary high-pass filter amplifies the coefficients of $\lambda_{\geq 1}$. For instance, the filter function of GCN is $\tilde{G} = \mathrm{I} - \Sigma$ where the coefficients of $\lambda_{<1}$ increases. Therefore, GCN acts as a low-pass filter.

4.3.3 Addition of Self-loops

We conduct experiments by adding the self-loops corresponding to each node in the graph. The number of self-loops is denoted by α. After the addition of self-loops α-times, the adjacency matrix will be $\tilde{A}_\alpha = A + \alpha\mathrm{I}$ and the corresponding degree matrix will look like $\tilde{D}_\alpha = D + \alpha\mathrm{I}$. Therefore, the symmetrically normalized graph Laplacian can be presented as $\tilde{L}_\alpha = \mathrm{I} - \tilde{D}_\alpha^{-\frac{1}{2}} \tilde{A}_\alpha \tilde{D}_\alpha^{-\frac{1}{2}}$.

4.3.4 Addition of Parallel edges

We also perform experiments by adding parallel edges corresponding to every edge in the graph. Assume γ denotes the number of parallel edges to be added in the network. Therefore, adding γ-times parallel edges, the updated adjacency matrix will be $A_\gamma = (\gamma + 1)A$. Thus, the corresponding degree matrix will be $D_\gamma = (\gamma + 1)D$. Adding self-loops the further modified adjacency and degree matrices will be $\tilde{A}_\gamma = (\gamma + 1)A + \mathrm{I}$ and $\tilde{D}_\gamma = (\gamma + 1)D + \mathrm{I}$. Therefore, we can define the corresponding symmetrically normalized graph Laplacian as $\tilde{L}_\gamma = \mathrm{I} - \tilde{D}_\gamma^{-\frac{1}{2}} \tilde{A}_\gamma \tilde{D}_\gamma^{-\frac{1}{2}}$.

4.3.5 Empirical Evidence on Random Graphs

We conducted experiments on randomly generated Erdős-Rényi graphs to study the effects on the eigenvalues with the addition of self-loops and parallel edges in the graph. A random graph $\mathcal{G}_{er}$ is generated with 10 vertices and having edge probability 0.50. Two individual experiments are performed with (1) the addition of self-loops, and (2) the addition of parallel edges. We varied the number of self-loops or parallel edges, ranging from 1 to 10. The eigendecomposition is performed for every stage of addition to monitor the changes that occurred in the corresponding eigenvalues. Refer to Figure 4.2 for the portrayal of the effect on the eigenvalues and the other corresponding changes in the spectrum. The plots (a) and

Table 4.2: Total number of nodes, isolated nodes, edge density ($\text{sp}_{\mathcal{G}}$), and average degree (d_{avg}) for 17 heterophilic graphs are presented. We also mentioned the log-scaled versions of edge density ($\overline{\text{sp}}_{\mathcal{G}}$) and average degrees ($\overline{d}_{\text{avg}}$) with lower values indicate higher density and average degree.

Properties/Datasets	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Actor
# nodes	183	183	251	2277	5201	7600
# isolated nodes (%)	87(47.5%)	73(39.8%)	81(32.3%)	0(0%)	0(0%)	636(8.36%)
density ($\text{sp}_{\mathcal{G}}$/$\overline{\text{sp}}_{\mathcal{G}}$)	0.017/4.02	0.019/3.93	0.016/4.10	0.013/4.27	0.016/4.13	0.001/6.86
avg. degree (d_{avg}/$\overline{d}_{\text{avg}}$)	1.62/3.82	1.77/3.82	2.05/4.13	15.85/6.34	41.73/7.17	3.94/7.54
Properties/Datasets	arxiv-year	snap-patents	Penn94	pokec	twitch-gamers	genius
# nodes	169343	2923922	41554	1632803	168114	421961
# isolated nodes (%)	17440(10.29%)	881754(30.15%)	0(0%)	200110(12.25%)	44596(26.52%)	371870(88.12%)
density ($\text{sp}_{\mathcal{G}}$/$\overline{\text{sp}}_{\mathcal{G}}$)	8.1e-5/9.41	3.2e-6/12.63	0.003/5.75	3.3e-5/10.68	9.6e-4/7.63	2e-5/11.41
avg. degree (d_{avg}/$\overline{d}_{\text{avg}}$)	6.88/10.65	4.77/13.50	65.56/9.24	27.31/12.91	80.86/10.64	4.37/11.56
Properties/Datasets	Roman-empire	Amazon-ratings	Minesweeper	Tolokers	Questions	-
# nodes	22662	24492	10000	11758	48921	
# isolated nodes (%)	0(0%)	3495(14.26%)	1(0.01%)	936(7.96%)	17761(36.3%)	
density ($\text{sp}_{\mathcal{G}}$/$\overline{\text{sp}}_{\mathcal{G}}$)	2.5e-4/8.26	3.1e-4/8.07	7.8e-4/7.14	0.007/4.89	1.2e-4/8.96	
avg. degree (d_{avg}/$\overline{d}_{\text{avg}}$)	2.91/8.64	3.79/8.71	3.94/7.82	44.14/7.98	3.14/9.41	

(b) demonstrate the effect of 10 eigenvalues of $\mathcal{G}_{er}$ with the addition of self-loops or parallel edges in the graph. On the other side, plots (c) and (d) depict the changes in the eigenvalues. As we have added 10 self-loops or parallel edges, thus 9 differences are recorded in the plots.

Observation The plots reaffirm that the addition of self-loops leads to the shrinking of the eigenvalues in the spectrum except for the eigenvalue 0 (Refer Figure 4.2(a)). On the contrary, eigenvalues increase with the addition of parallel edges (Refer Figure 4.2(b)). Additionally, we also present the change in the eigenvalues in Figures 4.2(c) and 4.2(d). Notably, the monotone increase (decrease) of the curves underlines the slower rate with the addition of self-loops (parallel edges).

4.3.6 Theoretical Analysis: A Spectral Perspective

We perform a deeper theoretical analysis of the effect on the graph spectrum when adding self-loops or parallel edges. The detailed study is provided as follows,

Lemma 4.1. *Consider a graph G with A and D as the adjacency and degree matrix. Now α-times self-loops are added in G with $\alpha_1 \in \mathbb{Z}^+$. Assume λ_α^{max} is the maximum eigenvalue of symmetrically normalized graph Laplacian $\tilde{L}_\alpha$ of the updated graph. If β_1 is the smallest eigenvalue of $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ and $\max_i d_i$ is the maximum degree of G, then $\lambda_\alpha^{max} \leq \frac{\max_i d_i(1-\beta_1)}{\alpha+\max_i d_i}$.*

Lemma 4.2. *Consider a graph G with A and D as the adjacency and degree matrix. Now*

Table 4.3: GCN (a low-pass filter) is applied on the 6 standard heterophilic datasets curated by Pie *et al.* (Pei et al., 2020). Performance analyses are demonstrated after adding multiple self-loops and parallel edges respectively in the graphs. The performance trends are also highlighted and corresponding categories are marked for each dataset.

Method	Input adjacency	Chameleon	Squirrel	Actor	Cornell	Texas	Wisconsin
GCN	A + I	71.68 ± 1.92	62.78 ± 1.99	27.40 ± 1.12	40.27 ± 6.44	54.86 ± 5.27	45.29 ± 6.10
	A + 2I	67.69 ± 2.25	58.64 ± 2.27	29.20 ± 1.13	43.78 ± 5.09	56.21 ± 4.95	50.19 ± 6.39
	A + 3I	65.15 ± 1.56	55.35 ± 1.76	30.67 ± 1.25	47.83 ± 8.11	54.05 ± 4.18	56.47 ± 3.80
	A + 4I	63.22 ± 1.22	53.43 ± 1.40	32.08 ± 1.20	46.48 ± 7.81	58.64 ± 6.16	60.98 ± 4.37
	A + 5I	61.90 ± 2.51	51.44 ± 1.51	32.94 ± 0.85	50.27 ± 7.47	57.02 ± 7.49	61.96 ± 5.42
Trend	→	Decreasing (↓)	Decreasing (↓)	Increasing (↑)	Increasing (↑)	Increasing (↑)	Increasing (↑)
GCN	A + I	71.68 ± 1.92	62.70 ± 1.99	27.40 ± 1.12	40.27 ± 6.44	54.86 ± 5.27	45.29 ± 6.10
	2A + I	75.06 ± 1.24	66.43 ± 2.40	25.54 ± 1.30	40.54 ± 6.39	54.59 ± 6.13	47.45 ± 4.94
	3A + I	76.31 ± 0.99	67.79 ± 1.96	25.63 ± 0.69	44.05 ± 7.83	55.67 ± 5.43	47.25 ± 5.22
	4A + I	76.60 ± 0.77	67.92 ± 1.66	24.76 ± 1.19	45.67 ± 8.58	51.89 ± 7.43	46.86 ± 4.24
	5A + I	77.32 ± 1.07	68.59 ± 1.71	24.82 ± 1.34	41.08 ± 5.64	51.08 ± 10.0	46.86 ± 4.75
Trend	→	Increasing (↑)	Increasing (↑)	Decreasing (↓)	Increasing (↑)	Increasing (↑)	Increasing (↑)
Category	→	C	C	B	A	A	A

γ-times self-loops are added in G with $\gamma \in \mathbb{Z}^+$. Assume λ_γ^{max} is the maximum eigenvalue of symmetrically normalized graph Laplacian $\tilde{L}_\gamma$ of the updated graph. If β_1 is the smallest eigenvalue of $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ and $\max_i d_i$ is the maximum degree of G, then $\lambda_\gamma^{max} \leq \frac{(1+\gamma)\max_i d_i(1-\beta_1)}{1+(1+\gamma)\max_i d_i}$.

Remark 4.1. *The maximum eigenvalue of $\tilde{L}_\alpha$ decreases with the increasing number of self-loops in the network, indicating the shrinking of the graph spectrum. On the contrary, the maximum eigenvalue of $\tilde{L}_\gamma$ increases with the increasing number of parallel edges in the network, illustrating the expansion of the graph spectrum.*

Lemma 4.3. *Given a k-regular graph $\mathcal{G}$, the eigenvalues of $\tilde{A}_N^\alpha$ will lie in $[-1, 1]$ $\forall \alpha \geq 1$.*

Lemma 4.4. *Given a k-regular graph $\mathcal{G}$, the eigenvalues of $\tilde{A}_N^\gamma$ will lie in $[-1, 1]$ $\forall \gamma \geq 1$.*

Remark 4.2. *The eigenvalues will lie in $[-1, 1]$ whether the self-loops or parallel edges are added in a regular graph. The range of eigenvalues will remain unaffected with the addition of self-loops or parallel edges.*

Theorem 4.1. *Consider a k-regular graph with $\alpha_1, \alpha_2 \in \mathbb{R}^+$ where $\alpha_1 \leq \alpha_2$, then the inequality will hold $\lambda_{\alpha_1}^i \geq \lambda_{\alpha_2}^i, \forall\ 1 \leq i \leq n$ where $\lambda_{\alpha_1}^i$ and $\lambda_{\alpha_2}^i$ are the i^{th} eigenvalues of $\tilde{L}_{\alpha_1}$ and $\tilde{L}_{\alpha_2}$ respectively.*

Theorem 4.2. *Consider a k-regular graph with $\gamma_1, \gamma_2 \in \mathbb{R}^+$ with $\gamma_1 \leq \gamma_2$, then the inequality will hold $\lambda_{\gamma_1}^i \leq \lambda_{\gamma_2}^i, \forall\ 1 \leq i \leq n$ where $\lambda_{\gamma_1}^i$ and $\lambda_{\gamma_2}^i$ are the i^{th} eigenvalues of $\tilde{L}_{\gamma_1}$ and $\tilde{L}_{\gamma_2}$ respectively.*

Remark 4.3. *We have shown that for the regular graphs, the eigenvalues of the symmetrically normalized graph Laplacian decrease concurrently when self-loops are added. Adding self-loops attenuates the graph spectrum frequencies, thereby shifting the graph spectrum towards zero eigenvalue. This will also increase the number of lower frequencies and decrease the number of higher frequencies.*

The eigenvalues of the symmetrically normalized graph Laplacian increase when parallel edges are added. Adding parallel edges amplifies the frequencies of the graph spectrum, which shifts the spectrum toward the value 2*. Consequently, the number of lower frequencies decreases and the number of higher frequencies increases.*

Corollary 4.1. *The increase in the eigenvalues of L_γ is independent of the number of self-loop additions in $\mathcal{G}$. On the contrary, the eigenvalues of $\tilde{L}_\gamma$ will increase if at least one self-loop is added per node in $\mathcal{G}$.*

In the following two theorems, we will demonstrate the alteration of the eigenvalues of A_N with the addition of self-loops or parallel edges by the perturbation of the adjacency matrix.

Theorem 4.3. *Consider a connected graph $\mathcal{G}$ with $A_N = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. Assuming the diagonal of A of G is perturbed by a significantly small $\alpha > 0$, then the updated normalized adjacency matrix will be A_N^α. The change in the eigenvalues of A_N^α with respect to the eigenvalues of A_N will increase when α increases.*

Theorem 4.4. *Consider a connected graph $\mathcal{G}$ with normalized adjacency matrix $A_N = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. Assuming each element of A except the diagonal is multiplied by $1+\gamma$ where $\gamma > 0$ is a significantly small quantity, then the updated normalized adjacency matrix will be A_N^γ. The change in the eigenvalues of A_N^γ with respect to the eigenvalues of A_N will decrease when γ increases.*

Remark 4.4. *Theorem 4.3 suggests that the change in the eigenvalues of the normalized adjacency matrix increases an increasing number of self-loops which also signify the decrease in the change in the eigenvalues of $\tilde{L}_\alpha$. The small perturbation in the adjacency matrix leads to a shrinking of the frequencies in the graph spectrum. The spectrum will shift toward the zero eigenvalue. Spectrum shrinking also highlights the alteration in the parity of frequencies. The number of lower frequencies increases and the number of higher frequencies decreases.*

Conversely, Theorem 4.4 states that the change of eigenvalues of the normalized adjacency matrix decreases with the increasing number of parallel edges. This indicates an increase in the change in the eigenvalues of $\tilde{L}_\gamma$, highlighting the spectrum expansion. To this effect, the graph spectrum will shift toward the value 2. Furthermore, the parity of frequencies changes, specifically, the number of lower frequencies decreases and the number of higher frequencies increases.

4.3.7 Effect on the Performance of GCN for Spectral Shifts

The theoretical analyses offer insights into the effects on graph spectra when self-loops or parallel edges are added. We also observed that spectrum shifts either to zero eigenvalue or to the value 2 and also alter the parity of the frequencies or eigenvalues. The higher number of low frequencies in a network indicates the presence of well-separated communities, mostly sharing identical node labels. GCN or any low-pass filter is supposed to smooth out the features of the connected nodes, assuming that they share similar class labels. The addition of self-loops increases the number of lower frequencies which appears to be beneficial for smoothing out features, improving the performance of GCN.

Conversely, the graph spectra shift toward the value 2, signifying the increase in the higher frequencies. A network with overlapped communities contains many high frequencies in the spectrum. GCN is poised to smooth out the features of nodes that are probably having different class labels. This will lead to the degradation of the performance of GCN.

4.3.8 Connection between Theoretical Analysis and Performance Trends

The various performance trends depend on the presence of parity between lower and higher frequencies in the spectrum of the input graph. As discussed in the theoretical analyses, the addition of self-loops and parallel edges respectively decreases and increases the eigenvalues or frequencies in the spectrum. If GCN witnessed an increasing trend on a heterophilic graph with the gradual addition of self-loops, then we can conclude that initially the graph spectrum is aligned toward zero eigenvalue and the addition of self-loops further tilts the balance to lower frequencies. The similar effects are evident in datasets like Actor, Cornell, pokec, etc. In contrast, if the performance of GCN exacerbates with the gradual addition

of parallel edges, then we can infer initial graph spectrum is skewed towards the maximum value 2, indicating a greater number of higher frequencies. Further addition of parallel edges enhances the high frequencies and the decreasing trend observed in the performance of GCN. The impact is evident in datasets like Actor, arxiv-year, snap-patents, Tolokers, etc.

4.4 Experiments

4.4.1 Datasets

Our experiments encompass three categories of datasets (1) Pie *et al.* (Pei et al., 2020) proposed 6 standard heterophilic networks consisting of Cornell, Texas, Wisconsin, Chameleon, Squirrel, and Actor, (2) Lim *et al.* (Lim et al., 2021) curated 6 large-scale heterophilic networks namely arxiv-year, snap-patents, Penn94, pokec, twitch-gamers, and genius, and (3) Platonov *et al.* (Platonov et al., 2023) identified prevailing shortcomings on the existing datasets and developed a set of 5 heterophilic networks viz Roman-empire, Amazon-ratings, Minesweeper, Tolokers, and Questions. The details of all datasets are vividly available in Table 4.2.

4.4.2 Experimental Settings

For all graphs from Pie *et al.*, we have considered 10 standard train/valid/test splits with 60%/20%/20% samples. Graphs from Lim *et al.* and Platonov *et al.* train/valid/test splits are fixed as 50%/25%/25% for 5 and 10 splits respectively. We applied a two-layered GCN architecture across all 17 graphs to carry out entire experiments. The dropout rate is fixed at 0.50 and LayerNorm is employed to make training convergence faster. The model parameters are optimized by Adam optimizer. We evaluated the best model on every split and finally reported mean and standard deviations across all splits for each of the datasets. The performance metric is test accuracy and for Minesweeper, Tolokers, and Questions the ROC-AUC is reported. Our Pytorch and Pytorch-geoemtric based implementation is available at https://anonymous.4open.science/r/DIHG-5212/README.md.

4.4.3 Analysis of Isolated Nodes, Sparsity, and Average Degree in the Heterophilic Networks

We offer an in-depth analysis of the number of isolated nodes, edge density, and the average degree in the context of the heterophilic networks. Refer to Table 4.2 for illustrating the comparative statistics of the various networks. As per the existing formula, the edge density can be defined as $\text{sp}_{\mathcal{G}} = \frac{2|\mathcal{E}|}{|\mathcal{V}|(|\mathcal{V}|-1)}$. If the input graph is too sparse, the estimated value will be too little to comprehend. Therefore, we devise an alternative solution with the assistance of a logarithmic scale which is defined as $\overline{\text{sp}}_{\mathcal{G}} = -\log(\text{sp} + \epsilon)$ where ϵ is tiny real number is added to avert the numerical instability. The scaling suggests that the lower the $\overline{\text{sp}}_{\mathcal{G}}$, the more dense the graph is. A similar issue is confronted in the estimation of average degree as $d_{\text{avg}} = \frac{2|\mathcal{E}|}{|\mathcal{V}|}$. Therefore, we also pursue similar tricks to tackle the issue. The scaled average degree is defined as $\overline{d}_{\text{avg}} = -\log(d_{\text{avg}} + \epsilon)$ where ϵ is same as defined as earlier. A lower $\overline{d}_{\text{avg}}$ signifies the higher average degree of the underlying network.

4.4.4 Category of Distribution of Eigenvalues

Suppose we attempt to apply a low-pass filter (like GCN) on any input graph. The graph will be pre-processed either by adding a fixed number of self-loops or parallel edges. Considering all possibilities, the low-pass filter can exhibit four distinct types of performance trends. The various possible trends are vividly portrayed in Table 4.1. We marked the categories respectively as A, B, C, and D. The different performance trends may be the manifestations of the underlying edge connectivity of the network. The structure of the networks has an inherent connection with the eigenvalues derived from the normalized graph Laplacian. Therefore, the defined categories may help to comprehend the parity of lower and higher frequencies (eigenvalues) of the graph spectrum.

4.4.5 Initial Distribution of Eigenvalues

We estimate the distribution of eigenvalues from the normalized graph Laplacian for six standard heterophilic datasets Cornell, Texas, Wisconsin, Chameleon, Squirrel, and Actor, Refer to Figure 4.3 for the detailed illustration. The histograms of Cornell, Texas, and Wisconsin are identical. On the other side, the corresponding histograms of Chameleon and

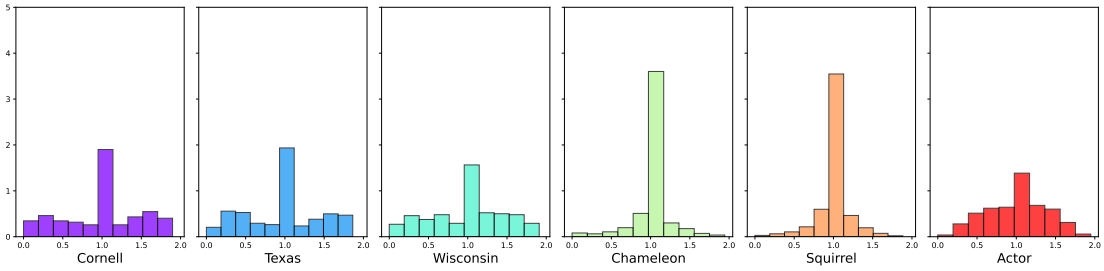


Figure 4.3: The distribution of eigenvalues of the normalized graph Laplacian presented for six standard heterophilic datasets obtained from (Pei et al., 2020).

Table 4.4: GCN (a low-pass filter) is applied on the 6 large-scale heterophilic datasets curated by Lim *et al.* (Lim et al., 2021). Performance analyses are demonstrated after adding multiple self-loops and parallel edges respectively in the graphs. The performance trends are also highlighted and corresponding categories are marked for each dataset.

Method	Input Adjacency	arxiv-year	snap-patents	Penn94	pokec	twitch-gamers	genius
GCN	A + I	48.75 ± 0.43	34.96 ± 0.15	77.12 ± 0.43	59.53 ± 0.18	60.83 ± 0.29	80.03 ± 0.66
	A + 2I	48.59 ± 0.53	34.77 ± 0.14	75.82 ± 0.45	59.17 ± 0.20	61.00 ± 0.34	79.81 ± 1.38
	A + 3I	47.34 ± 0.20	34.60 ± 0.04	$74,32 \pm 0.49$	59.77 ± 0.15	61.07 ± 0.33	78.77 ± 0.99
	A + 4I	46.26 ± 0.21	34.57 ± 0.11	72.79 ± 0.55	60.43 ± 0.14	61.18 ± 0.26	77.80 ± 0.51
	A + 5I	45.73 ± 0.48	34.43 ± 0.17	71.39 ± 0.40	60.94 ± 0.17	61.23 ± 0.30	77.35 ± 0.37
Trend	→	Decreasing (↓)	Decreasing (↓)	Decreasing (↓)	Increasing (↑)	Increasing (↑)	Decreasing (↓)
GCN	A + I	48.69 ± 0.37	35.00 ± 0.15	77.13 ± 0.39	59.54 ± 0.23	60.86 ± 0.30	80.09 ± 0.70
	2A + I	48.31 ± 0.54	34.89 ± 0.12	77.64 ± 0.41	61.32 ± 0.14	60.72 ± 0.21	74.41 ± 2.07
	3A + I	47.93 ± 0.64	34.64 ± 0.23	77.81 ± 0.38	62.22 ± 0.10	60.70 ± 0.24	69.88 ± 0.48
	4A + I	47.74 ± 0.52	34.46 ± 0.18	77.88 ± 0.37	62.77 ± 0.09	60.70 ± 0.21	70.17 ± 0.89
	5A + I	47.78 ± 0.38	34.28 ± 0.13	77.81 ± 0.35	63.10 ± 0.10	60.69 ± 0.21	71.13 ± 1.26
Trend	→	Decreasing (↓)	Decreasing (↓)	Increasing (↑)	Increasing (↑)	Decreasing (↓)	Decreasing (↓)
Category	→	D	D	C	A	B	D

Squirrel carry similar patterns, The histogram of the Actor dataset is completely different compared to the rest of the others. Later we will observe that the datasets have histograms of similar patterns that will yield identical trends in the performances.

4.4.6 Performance Categorisation of Low-pass Filter

We performed semi-supervised node classification on a diverse array of heterophilic graphs to analyze the different performance trends of the low-pass filter. For each graph, separate experiments were conducted to study the effects of the addition of self-loops and parallel edges respectively. We applied GCN, a well-adopted low-pass filter, on 17 heterophilic graphs, and the respective performance trends are demonstrated in Tables 4.3, 4.4, and 4.5 for respectively standard heterophilic graphs, large-scale datasets, and currently proposed heterophilic graphs. The study reveals that each graph exhibits a steady pattern of either increasing or

decreasing while adding either the self-loops or the parallel edges. We further marked the categories considering the performance trends for the increasing number of self-loops and parallel edges. For instance, GCN on Chameleon showed a decreasing trend while increasing the number of parallel edges, and on the contrary, performance increases on adding the parallel edges. Therefore, Chameleon was assumed to be in the category of A as per the rules from Table 4.1. Note that, the steady patterns are persistent across every dataset considered for the experimentation.

4.4.7 Observation from Performance Trends

The rationale against the backdrop of the performance trend can be explained through the lens of analyzing the spectrum of the graph. Since GCN performs the low-pass filtering, the coefficients of the lower frequencies will be enhanced, and the coefficients of the higher frequencies will be shrunk. Based on this, our discussion will revolve around analyzing 6 standard heterophilic graphs.

Table 4.5: GCN (a low-pass filter) is applied on the 5 newly-proposed heterophilic datasets curated by Platonov *et al.* (Platonov et al., 2023). Performance analyses are demonstrated after adding multiple self-loops and parallel edges respectively in the graphs. The performance trends are also highlighted and corresponding categories are marked for each dataset.

Method	Input Adjacency	Roman-empire	Amazon-ratings	Minesweeper	Tolokers	Questions
GCN	A + I	76.70 ± 0.63	42.01 ± 0.58	89.51 ± 0.54	80.10 ± 1.11	73.96 ± 1.44
	A + 2I	76.28 ± 0.58	41.58 ± 0.56	89.46 ± 0.53	80.13 ± 1.06	73.69 ± 0.67
	A + 3I	75.99 ± 0.76	41.56 ± 0.53	89.41 ± 0.55	80.03 ± 1.02	72.54 ± 1.64
	A + 4I	75.72 ± 0.70	41.49 ± 0.40	89.33 ± 0.54	79.76 ± 1.02	72.56 ± 1.23
	A + 5I	75.26 ± 0.55	41.21 ± 0.60	89.22 ± 0.53	79.70 ± 0.98	72.53 ± 1.15
Trend	→	Decreasing (↓)	Decreasing (↓)	Decreasing (↓)	Decreasing (↓)	Decreasing (↓)
GCN	A + I	76.71 ± 0.62	42.00 ± 0.58	89.51 ± 0.54	80.10 ± 1.12	73.30 ± 2.04
	2A + I	76.75 ± 0.80	41.93 ± 0.50	89.57 ± 0.51	79.95 ± 1.08	74.51 ± 1.23
	3A + I	76.99 ± 0.67	41.93 ± 0.90	89.57 ± 0.52	79.87 ± 0.93	75.05 ± 0.99
	4A + I	76.90 ± 0.59	42.02 ± 0.53	89.57 ± 0.51	79.87 ± 0.99	74.93 ± 1.18
	5A + I	76.95 ± 0.65	42.18 ± 0.77	89.60 ± 0.49	79.75 ± 0.94	74.73 ± 1.58
Trend	→	Increasing (↑)	Increasing (↑)	Increasing (↑)	Decreasing (↓)	Increasing (↑)
Category	→	C	C	C	D	C

Addition of Self-loops We observed that the addition of self-loops improves the performance of GCN on Cornell, Texas, Wisconsin, and Actor (Refer Table 4.3). Aligning to the theoretical analyses, adding self-loops will create a decreasing trend of the eigenvalues of $\tilde{L}$. Consequently, the number of lower frequencies will increase in the graph spectrum, which

is beneficial for the performance boost of the GCN. Refer to Figure 4.4((a), (b), (c), (f)) to visualize the change of distribution of eigenvalues of $\tilde{L}$ with the addition of self-loops. A deep observation reveals that the number of lower frequencies gradually increases while increasing the number of self-loops in the graph. This transformation offers a conducive environment for the operation of the low-pass filter (here GCN). Finally, the shape of the eigenvalue distribution resembles when $\alpha = 5$ for all four datasets.

A stark contrast was observed in the performance trends of the Chameleon and Squirrel datasets. Both of the datasets demonstrated degrading performance with the addition of an increasing number of self-loops. As mentioned earlier, the addition of self-loops will shrink the spectrum of the graph, leading to an increase in the lower frequencies. Refer 4.4((d), (e)) for the nuanced view of the alteration of eigenvalue distribution of $\tilde{L}$ for both datasets. Both histograms depict that the lower frequencies increase but eigenvalues are mostly centered towards 1 which might not be beneficial for the operation of GCN. Therefore, in this scenario, GCN witnessed deteriorating performance with the increasing number of self-loops.

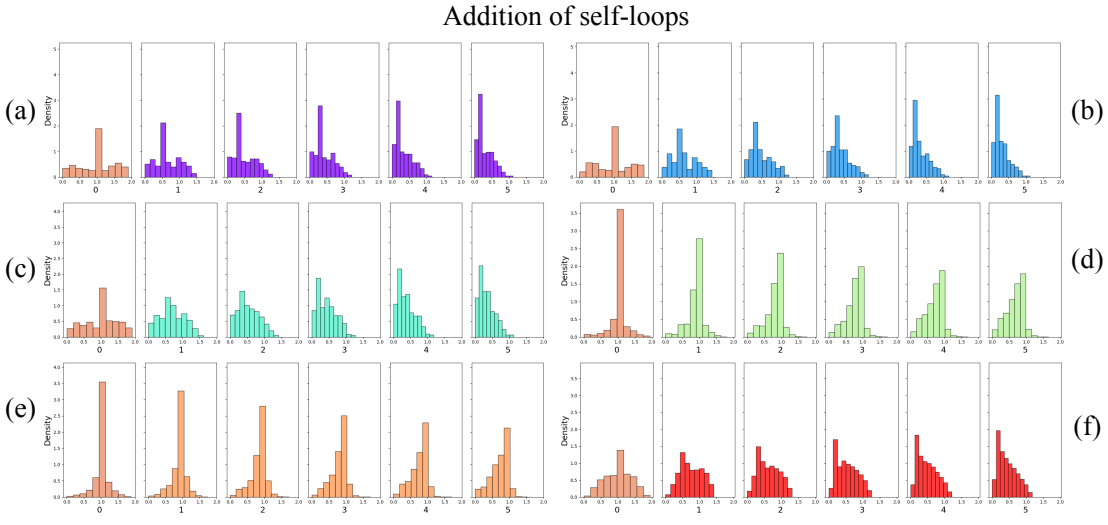


Figure 4.4: The distribution of eigenvalues of normalized graph Laplacian for the datasets (a) Cornell, (b) Texas, (c) Wisconsin, (d) Chameleon, (e) Squirrel, and (f) Actor is demonstrated after adding self-loops. The initial distribution is shown on the left side of each diagram. The number of self-loops varies from 1 to 5 and corresponding changes are recorded.

Addition of Parallel edges The addition of parallel edges improves the performance of Cornell, Texas, Wisconsin, Chameleon, and Squirrel. The theoretical analysis illustrates that

eigenvalues increase with the increasing number of parallel edges. Consequently, the number of higher frequencies will increase which seems to impede the performance of GCN. Refer to Figure 4.5((a), (b), (c), (d), (e)) to visualize the change in the distribution of eigenvalues of $\tilde{L}$ with the addition of the parallel edges. One common point should be mentioned that despite the increasing number of higher frequencies, many lower frequencies are still retained in the spectrum. This phenomenon performs the balance of the parity of lower and higher frequencies in the graph spectrum, eventually improving the performance of GCN.

On the contrary, GCN exhibited deteriorating performances on the Actor dataset with the increasing number of parallel edges. Refer to Figure 4.5(f) for the histogram asserting that the number of lower frequencies almost diminished. As a consequence, GCN witnessed a degrading performance.

Key Takeaway The crux of the experiments lies in the identification of trends (either increasing or decreasing) in the performance of low-pass filters with the addition of self-loops or parallel edges. Every input graph can be uniquely mapped to one of the four pre-defined categories depending on the performance trends. We marked the respective categories of all 17 heterophilic graphs involved in the experimentation. The assigned category characterizes the specific eigenvalue distribution of the normalized graph Laplacian for the corresponding graph. The distribution of the eigenvalues offers significant insights into the structural patterns of the graphs like connected components, community structures, expansion properties, clustering, robustness, etc. The unraveling of such properties is often accomplished by pursuing computationally expensive eigenvalue decomposition or time-consuming prevailing graph algorithms. Amid the growing size of the graph resorting to such strategies leads to a labyrinth of the methodologies.

The size of the graphs like Cornell, Texas, or Chameleon is moderate and we performed eigenvalue decomposition to study the eigenvalue distribution of $\tilde{L}$. The initial distributions of the six graphs are presented in Figure 4.3. One can easily contemplate the high-level structural properties by observing the initial patterns. For instance, Cornell, Texas, and Wisconsin have a distribution mostly symmetrical around 1 in the spectrum, asserting the balanced parity of lower and higher frequencies. The observation underscores the presence of weakly connected components or isolated nodes. This is also validated from Table 4.2

as Cornell contains 47.5% isolated nodes. A similar argument applies to both Texas and Wisconsin. Conversely, Chameleon and Squirrel both have higher frequencies which signifies that graphs are dense, have strongly connected components, and contain no isolated nodes as confirmed from Table 4.2. In a different vein, Actor has an almost balanced parity of lower and higher frequencies characterizing the lower number of isolated nodes with a moderate average degree and density compared to the other five previously mentioned graphs.

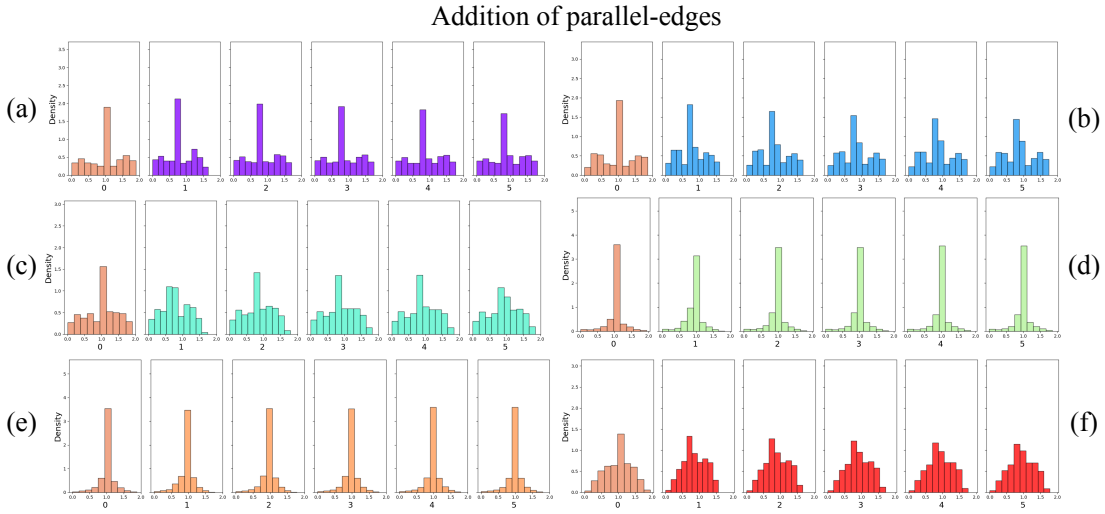


Figure 4.5: The distribution of eigenvalues of normalized graph Laplacian for the datasets (a) Cornell, (b) Texas, (c) Wisconsin, (d) Chameleon, (e) Squirrel, and (f) Actor are demonstrated after adding parallel edges. The initial distribution is shown on the left side of each diagram. The number of parallel edges varies from 1 to 5 and corresponding changes are recorded.

4.4.8 Inferring Characteristics of Spectrum for Large-scale Graphs

Evaluating the eigenvalue distribution of $\tilde{L}$ for a large-scale graph is critically challenging due to the potential computational overhead. Exploration of eigenvalue distribution is possible by separately observing the performance trends of the low-pass filters, with the addition of self-loops and parallel edges. This approach drastically reduces the computational budget and offers deeper insights into the intricate structural patterns in the given networks.

arxiv-year, snap-patents, and genius As per empirical evidence, GCN witnessed decreasing performance trends on arxiv-year, snap-patents, and genius in both cases of self-loops and parallel edges. The results emphasized that three networks contain a significantly lower num-

ber of non-zero frequencies with a substantial number of zero eigenvalues, asserting that the networks contain a large number of isolated nodes. The edge density and average degree of the graphs are also lower for the three graphs which can also be verified by referring to Table 4.2. Therefore, we can predict the weak connectivity and the presence of isolated nodes in those graphs by only observing the performance trends with the addition of self-loops and parallel edges.

Penn94 Penn94 was categorized in "C" resembling the category of Chameleon and Squirrel (Refer Figure 4.3). The initial frequency distribution of $\tilde{L}$ suggests a greater number of higher frequencies resulting in the densely connected network compared to the other five graphs (Refer Table 4.2). Also, Penn94 contains no isolated nodes sharing properties similar to those of Chameleon and Squirrel.

twitch-gamers The performance trends made twitch-gamers posited in category "B" which is identical to Actor. Thus, the eigenvalue distribution of twitch-gamers also resembles to Actor, having balanced centering to eigenvalue 1 and almost equal parity of lower and higher frequencies. Like Actor, twitch-gamers also have moderately dense and average node degrees with a lesser number of isolated nodes compared to the other five heterophilic graphs in this group.

pokec The category of pokec marked as A shares identical characteristics with Cornell, Texas, and Wisconsin. The performance trend indicated the presence of the eigenvalues centering around eigenvalue 1 having balanced parity of lower and higher frequencies. Identically, pokec may contain isolated nodes and also weakly connected components.

Table 4.6: Analysis of the performance of GCN on Chameleon dataset is presented with the variation of number of self-loops and parallel edges simultaneously across the network.

Dataset	# Parallel edges		1	2	3	4	5
Chameleon	# Self-loops	1	65.00 ± 1.32	68.61 ± 1.52	68.94 ± 1.53	69.14 ± 1.35	69.75 ± 0.98
		2	59.84 ± 1.97	65.06 ± 2.07	66.71 ± 1.35	68.50 ± 1.70	68.64 ± 1.77
		3	58.55 ± 2.15	68.61 ± 2.26	65.00 ± 1.95	66.88 ± 2.01	67.41 ± 1.39
		4	58.04 ± 2.78	59.53 ± 1.63	62.82 ± 2.03	64.42 ± 1.81	67.03 ± 1.90
		5	57.93 ± 2.31	59.16 ± 1.54	60.24 ± 2.55	63.70 ± 2.00	65.06 ± 1.49

Roman-empire, Amazon-ratings, Minesweeper, and Questions The four graphs have demonstrated declining performance trends while self-loops are added and performance is improved with the addition of parallel edges. The graphs thereby belonged to the category

of "C" similar to Chameleon and Squirrel. This predicament has underscored that graphs contain higher frequencies than lower frequencies. The graph spectra are indicative of the higher sparsity and lower average degree of the graphs, compared to those of the Tolokers.

Tolokers Tolokers is categorized as "D" having similarity with arxiv-year, snap-patents, and genius. The performance trends signal the presence of a higher number of zero eigenvalues in the spectrum, containing a good number of isolated nodes (Refer to Table 4.2).

4.4.9 Runtime Comparison with Traditional Eigendecomposition Algorithms

We conducted an extensive study on 17 heterophilic benchmarks to compare the runtime of our proposed strategy with that of traditional eigendecomposition algorithms. We applied an inbuilt function `np.linalg.eig` implemented in the Numpy package to execute eigendecomposition. All experiments are carried out on a single 24 NVIDIA GeForce RTX 3090 GPU and we obtained the results as presented in Table 4.7. For each graph, we leverage 10 GNN training sessions for 5 self-loops and 5 parallel edges. For both cases, we utilized `time` function from Python. The analyses reveal that for small-scale graphs, the traditional algorithm outperforms our approach but for medium-sized graphs, our approach exhibited faster runtime in comparison to the inbuilt algorithm. Additionally, for large-scale graphs, our system showed Out-of-Memory (OOM) or process killed and was unable to estimate eigendecomposition. The experimental results underscore the significance of our approach to gain insights into the spectra without performing the time and memory-intensive existing eigendecomposition techniques.

Table 4.7: The comparative study between traditional eigendecomposition algorithms and our method applied to 17 heterophilic benchmarks. OOM denotes out-of-memory.

Runtime (sec.)	Chameleon	Squirrel	Film	Texas	Cornell	Wisconsin
`np.linalg.eig`	2.5334	22.1533	86.9033	0.0442	0.0226	0.0599
Ours	73.4782	162.0774	371.0116	52.9237	52.5499	52.9963
Runtime (sec.)	arxiv-year	snap-patents	Penn94	pokec	twitch-gamers	genius
`np.linalg.eig`	OOM	OOM	OOM	OOM	OOM	OOM
Ours	365.9806	699.1057	2866.6932	1467.4539	1097.2305	576.4411
Runtime (sec.)	Roman-empire	Amazon-ratings	Minesweper	Tolokers	Questions	
`np.linalg.eig`	3240.4251	3032.0099	40.7877	94.4341	process killed	
Ours	95.5101	135.3722	79.7196	360.0264	247.7609	

4.4.10 Visualization of Spectrum for Heterophilic Graphs

We conducted a comparative study on the lower and higher frequencies of the six heterophilic graphs Chameleon, Squirrel, Texas, Cornell, Wisconsin, and Actor. The eigenvalue decomposition is performed on $\tilde{L}$ of individual datasets. The corresponding eigenvalues are divided into two sets $\lambda_{<1}$ and $\lambda_{\geq 1}$ as mentioned earlier. We estimated the percentage of the low frequencies and high frequencies of each dataset. Refer to Figure 4.6 for the detailed illustration. The figure delineates that the number of lower and higher frequencies for Cornell,

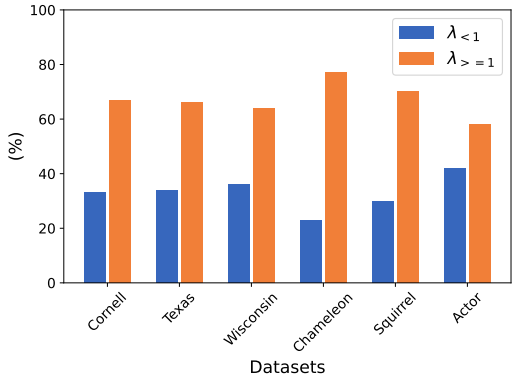


Figure 4.6: A comparative study on the number of low frequencies and high frequencies of the graph spectrum for six standard heterophilic graphs.

Texas, and Wisconsin are identical. In the experiments, their category is also similar which is "A". Concurrently, the Chameleon and Squirrel have similar patterns of the parity of eigenvalues and they also belong to a similar category "C". Furthermore, the pattern for the Actor is completely different from the rest of the others, and ends up marked as "B". The study established the unique interconnectedness of the parity of eigenvalues, frequency distribution of spectrum, and performance trends of low-pass filters.

4.4.11 Variation of both Self-loops and Parallel edges

A study is conducted to observe the effect on the performance of GCN with the variation of both self-loops and parallel edges. We consider the Chameleon as our candidate graph to serve the purpose. The number of self-loops and parallel edges both varied from 1 to 5, taking into account 25 combinations. GCN is applied to the modified Chameleon graph for each combination. The mean test accuracy with standard deviation estimated over 10

splits is reported against each combination. Refer to Table 4.6 to get a detailed illustration of the results. The uptick trend is noted for the increasing number of parallel edges in the network. While increasing the number of self-loops, a degradation in the model performance is observed. For a fixed number of self-loops, the test accuracy increases with the addition of parallel edges, irrespective of the beginning of the initial performance. Every combination of (α_i, γ_j) produces a filter with the adjusted lower and higher number of frequencies. Notably, the lowest performance is achieved when the number of self-loops is highest ($\alpha = 5$) and the number of parallel edges is lowest ($\gamma = 1$). The best performance is obtained with just the reverse settings like the lowest number of self-loops ($\alpha = 1$) and the highest number of parallel edges ($\gamma = 5$).

Intuitive Explanation. Adding self-loops increases the number of lower frequencies, shifting the graph spectrum towards the eigenvalue 0. Conversely, the addition of parallel edges increases the number of higher frequencies, shifting the spectrum toward the value 2. The addition of a maximum $\mathcal{P}$ number of self-loops and a maximum $\mathcal{Q}$ number of parallel edges shifts the graph spectrum to the left and right directions accordingly. Precisely, the addition of $\alpha \leq \mathcal{P}$-number of self-loops and $\gamma \leq \mathcal{Q}$-number of parallel edges will transform the graph spectrum intermediate between the two aforementioned extreme scenarios. The updated spectrum oscillation between the two extreme cases and the performance of GCN is also reflected in our quantitative analysis.

4.4.12 Effect on Performance with Very Large α and γ

We performed a study on the Chameleon, Squirrel, Roman-empire, and genius by increasing α and γ to higher values to monitor the performance of GCN. The numbers of both α and γ are varied from 1 to 20, and corresponding test accuracy with standard deviations are demonstrated in Figure 4.7. The study indicates the maintenance of the performance trends of the different datasets. The trend mostly depends on the input graph. For example, GCN on Chameleon showed a decrease (increase) in the number of self-loops (parallel edges). Conversely, GCN on genius exhibited a downtrend in performance by adding both self-loops and parallel edges. It is noteworthy that performance stabilized with the higher value of α or γ. Empirical observation points out that the change in eigenvalues will become comparably negligible when the value of α or γ exceeds a certain limit. This phenomenon can be attributed

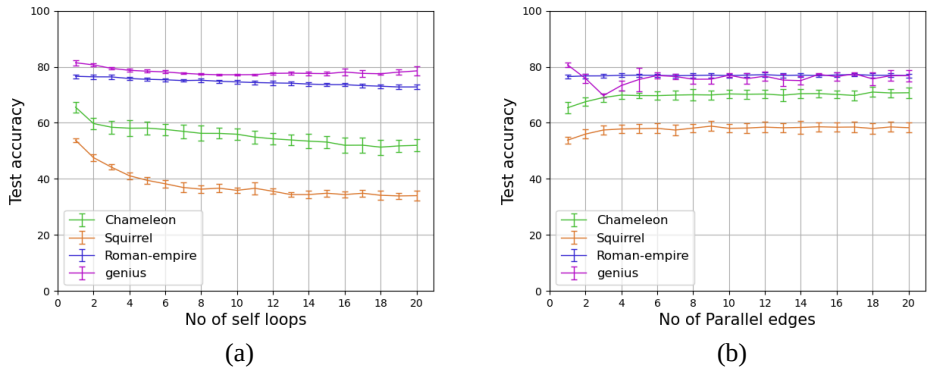


Figure 4.7: The effect on the performance of GCN on Chameleon, Squirrel, Roman-empire, and genius when (a) self-loops and (b) parallel edges are added a higher number of times is presented.

to the saturating performance trends observed in the experiment.

4.4.13 Utility for Practitioners

In this work, we primarily focus on determining properties of the graph spectrum without resorting to costly eigenvalue decomposition. Based on the patterns observed in GCN's performance, we map the input graph into one of the four categories. Beyond the theoretical insights and empirical evaluations, our work possesses some benefits for practitioners. We enjoy the following advantages without performing direct eigendecomposition.

- Our strategy reveals the shape of the graph spectrum characterizing structural properties like the presence of connected components, communities, etc. For example, regular graphs have a symmetric graph spectrum. Additionally, the spectral density can identify graph classes like scale-free, random, and small-world.

- Computing similarity measures between two large graphs is cumbersome. Our method offers insights into the spectrum distributions, whose comparisons enable quick similarity assessment between the large networks.

- We can detect anomalies of the evolving or dynamic networks by monitoring shifts in the respective distribution shifts.

4.5 Conclusion

We conduct detailed studies regarding the performance of low-pass filters like GCN on the heterophilic graphs. The studies revealed that GCN showed monotone performance trends when the input graph is equipped with an increasing number of self-loops or parallel edges. Based on these performance trends, we categorize the input graphs into four distinct categories. We further observed that the eigenvalues of the normalized graph Laplacian decrease and increase when self-loops or parallel edges are added. Consequently, each category entails a significant amount of information pertaining to the characteristics of the graph spectrum. Therefore, the performance trends depend solely on the distribution of the eigenvalues in the entire spectrum. The graph spectrum patterns reveal the graph's intrinsic characteristics, such as connected components, community structure, etc. Our work manifests a cost-effective pathway for estimating and understanding intrinsic properties and intricate patterns in the graph data, refraining from performing expensive computations. The design of effective application of GNNs to replace the prevailing costly algorithmic computations can be a potential avenue for future research directions.

4.6 Appendix

4.6.1 Proofs

This section will offer the detailed proofs and necessary derivations for Lemma 4.1, Lemma 4.2, Lemma 4.3, Lemma 4.4, Theorem 4.1, Theorem 4.2, Theorem 4.3, Theorem 4.4, and Corollary 4.2. Before delving into the detailed proofs and derivations, the following Lemma will assist in proving the following theorems. The following proposition is adopted from (Wu et al., 2019).

Proposition 1. *Let us assume* $\beta_1 \leq \beta_2 \leq \cdots \leq \beta_n$ *are the eigenvalues of* $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ *and* $\delta_1 \leq \delta_2 \leq \cdots \leq \delta_n$ *are the eigenvalues of* $\tilde{D}^{-\frac{1}{2}}A\tilde{D}^{-\frac{1}{2}}$ *where* $\tilde{D}_\alpha = D + \alpha I$*, then we have the following inequalities*

$$\delta_1 \geq \frac{max_i d_i}{\alpha + max_i d_i}\beta_1, \qquad \delta_n \leq \frac{min_i d_i}{\alpha + min_i d_i}. \tag{4.2}$$

Proof. Recall that $L_{\text{sym}} = \mathrm{I} - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ and a well-known fact is that 0 is an eigenvalue of L_{sym}. Therefore, we have $\beta_n = 1$. As A is free of self-loops then $\mathrm{Tr}(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}) = 0 = \sum_i \beta_i$ which implies $\beta_1 < 0$.

Choose x such that $||x|| = 1$ and consider $y = D^{\frac{1}{2}}\tilde{D}_\alpha^{-\frac{1}{2}}x$. Now, $||y||^2 = \sum_i \frac{d_i}{d_i+\alpha}x_i^2$. Also, we have $\frac{\min_i d_i}{\alpha+\min_i d_i} \leq ||y||^2 \leq \frac{\max_i d_i}{\alpha+\max_i d_i}$.

Applying the Rayleigh quotient, we have the following bound for the smallest eigenvalue α_1,

$$
\begin{aligned}
\delta_1 &= \min_{||x||=1}(x^\top \tilde{D}_\alpha^{-\frac{1}{2}}A\tilde{D}_\alpha^{-\frac{1}{2}}x) \\
&= \min_{||x||=1}(y^\top D^{-\frac{1}{2}}AD^{-\frac{1}{2}}y) \qquad \text{(by variable substitution)} \\
&= \min_{||x||=1}(\frac{y^\top D^{-\frac{1}{2}}AD^{-\frac{1}{2}}y}{||y||^2}||y||^2) \\
&\geq \min_{||x||=1}(\frac{y^\top D^{-\frac{1}{2}}AD^{-\frac{1}{2}}y}{||y||^2}) \max_{||x||=1}(||y||^2) \\
&(\because \min(f(z)g(z)) \geq \min(f(z))\max(g(z)) \text{ if} \\
&\min(f(z)) < 0, \forall g(z) > 0 \\
&\text{and } \min_{||x||=1}(\frac{y^\top D^{-\frac{1}{2}}AD^{-\frac{1}{2}}y}{||y||^2}) = \beta_1 < 0) \\
&= \beta_1 \max_{||x||=1}||y||^2 \\
&\geq \frac{\max_i d_i}{\alpha + \max_i d_i}\beta_1
\end{aligned}
\tag{4.3}
$$

Similarly, the upper bound for δ_n can be proved as $\delta_n \leq \frac{\min_i d_i}{\alpha+\min_i d_i}$. A similar problem can be solved for the parallel edge addition with $\tilde{D}_\gamma = (1+\gamma)D+I$. and $\delta_1 = \min_{||x||=1}(x^\top \tilde{D}_\gamma^{-\frac{1}{2}}A\tilde{D}_\gamma^{-\frac{1}{2}}x)$. This problem will yield the bound as $\delta_1 \geq \frac{\max d_i}{1+(1+\gamma)\max d_i}\beta_1$. The bound can be derived similarly as depicted in Eq. 4.3. □

Lemma 4.1 Consider a graph G with A and D as the adjacency and degree matrix. Now α-times self-loops are added in G with $\alpha \in \mathbb{Z}^+$. Assume $\lambda_\alpha^{\max}$ is the maximum eigenvalue of symmetrically normalized graph Laplacian $\tilde{L}_\alpha$ of the updated graph. If β_1 is the smallest eigenvalue of $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ and $\max_i d_i$ is the maximum degree of G, then $\lambda_\alpha^{\max} \leq \frac{\max_i d_i(1-\beta_1)}{\alpha+\max_i d_i}$.

Proof. After applying α-times self-loops the symmetrically normalized Laplacian is presented

as:

$$\begin{aligned}\tilde{L}_\alpha &= \mathrm{I} - \tilde{D}_\alpha^{-\frac{1}{2}} \tilde{A}_\alpha \tilde{D}_\alpha^{-\frac{1}{2}} \\ &= \mathrm{I} - \tilde{D}_\alpha^{-\frac{1}{2}} (A + \alpha \mathrm{I}) \tilde{D}_\alpha^{-\frac{1}{2}} \\ &= \mathrm{I} - \tilde{D}_\alpha^{-\frac{1}{2}} A \tilde{D}_\alpha^{-\frac{1}{2}} - \alpha \tilde{D}_\alpha^{-1}\end{aligned} \tag{4.4}$$

Let $\lambda_\alpha^{\max}$ is the maximum eigenvalue of $\tilde{L}_\alpha$ and applying Rayleigh quotient the following can be obtained

$$\begin{aligned}\lambda_\alpha^{\max} &= \max_{||x||=1} x^\top \tilde{L}_\alpha x \\ &= \max_{||x||=1} x^\top (\mathrm{I} - \tilde{D}_\alpha^{-\frac{1}{2}} A \tilde{D}_\alpha^{-\frac{1}{2}} - \alpha \tilde{D}_\alpha^{-1}) x \\ &\leq (1 - \min_{||x||=1} x^\top \tilde{D}_\alpha^{-\frac{1}{2}} A \tilde{D}_\alpha^{-\frac{1}{2}} x - \min_{||x||=1} \alpha x^\top \tilde{D}_\alpha^{-1} x) \\ &= 1 - \delta_1 - \frac{\alpha}{\alpha + \max_i d_i} \quad \text{(from Proposition 1)} \\ &\leq 1 - \frac{\max_i d_i}{\alpha + \max_i d_i} \beta_1 - \frac{\alpha}{\alpha + \max_i d_i} \\ &= \frac{\max_i d_i (1 - \beta_1)}{\alpha + \max_i d_i}\end{aligned} \tag{4.5}$$

When α increases the upper bound of $\lambda_\alpha^{\max}$ decreases which indicates the possible shrinking of the maximum eigenvalue of the graph spectrum.

□

Lemma 4.2 Consider a graph G with A and D as the adjacency and degree matrix. Now γ-times parallel edges are added in G with $\gamma \in \mathbb{Z}^+$. Assume $\lambda_\gamma^{\max}$ is the maximum eigenvalue of symmetrically normalized graph Laplacian $\tilde{L}_\gamma$ of the updated graph. If β_1 is the smallest eigenvalue of $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ and $\max_i d_i$ is the maximum degree of G, then $\lambda_\gamma^{\max} \leq \frac{(1+\gamma) \max_i d_i (1-\beta_1)}{1+(1+\gamma) \max_i d_i}$.

Proof. After applying γ-times parallel edges in the graph, the symmetrically normalized graph

Laplacian will be

$$\begin{aligned}\tilde{L}_\gamma &= \mathrm{I} - \tilde{D}_\gamma^{-\frac{1}{2}}\tilde{A}_\gamma\tilde{D}_\gamma^{-\frac{1}{2}} \\ &= \mathrm{I} - \tilde{D}_\gamma^{-\frac{1}{2}}((\gamma+1)A+\mathrm{I})\tilde{D}_\gamma^{-\frac{1}{2}} \\ &= \mathrm{I} - (\gamma+1)\tilde{D}_\gamma^{-\frac{1}{2}}A\tilde{D}_\gamma^{-\frac{1}{2}} - \tilde{D}_\gamma^{-1}\end{aligned} \tag{4.6}$$

Let $\lambda_\gamma^{\max}$ is the maximum eigenvalue of $\tilde{L}_\gamma$ and applying Rayleigh quotient the following can be obtained

$$\begin{aligned}\lambda_\gamma^{\max} &= \max_{||x||=1} x^\top \tilde{L}_\gamma x \\ &= \max_{||x||=1} x^\top(\mathrm{I} - (\gamma+1)\tilde{D}_\gamma^{-\frac{1}{2}}A\tilde{D}_\gamma^{-\frac{1}{2}} - \tilde{D}_\gamma^{-1})x \\ &\leq (1 - (\gamma+1)\min_{||x||=1} x^\top\tilde{D}_\gamma^{-\frac{1}{2}}A\tilde{D}_\gamma^{-\frac{1}{2}}x - \min_{||x||=1} x^\top\tilde{D}_\gamma^{-1}x) \\ &= 1 - (\gamma+1)\delta_1 - \frac{1}{1+(1+\gamma)\max_i d_i} \text{ (from Proposition 1)} \\ &\leq 1 - \frac{(\gamma+1)\max_i d_i}{1+(1+\gamma)\max_i d_i}\beta_1 - \frac{1}{1+(1+\gamma)\max_i d_i} \\ &= \frac{(1+\gamma)\max_i d_i(1-\beta_1)}{1+(1+\gamma)\max_i d_i}\end{aligned} \tag{4.7}$$

When γ increases, the upper bound of $\lambda_\gamma^{\max}$ increases which shows the possible expansion of the maximum eigenvalue of the graph spectrum.

□

Lemma 4.3 Given a k-regular graph $\mathcal{G}$, the eigenvalues of $\tilde{A}_N^\alpha$ will lie in $[-1, 1]$ $\forall\alpha \geq 1$.

Proof. Consider a k-regular graph $\mathcal{G}$ where each node has a degree k with the normalized adjacency matrix is $\tilde{A}_N = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ where $\tilde{A} = A + I, \tilde{D} = D + I$. If α-times (with $\alpha \geq 1$) self-loops are added, then the updated normalized adjacency matrix is $\tilde{A}_N^\alpha = \tilde{D}_\alpha^{-\frac{1}{2}}\tilde{A}_\alpha\tilde{D}_\alpha^{-\frac{1}{2}}$ where $\tilde{A}_\alpha = A + \alpha I, \tilde{D}_\alpha = D + \alpha I$. As the graph is regular then $\tilde{A}_N$ can be presented as:

$$\begin{aligned}\tilde{A}_N &= \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}} \\ &= \frac{1}{\sqrt{k+1}}(A+I)\frac{1}{\sqrt{k+1}} \\ &= \frac{1}{k+1}(A+I)\end{aligned} \tag{4.8}$$

In a similar fashion, we can express $\tilde{A}_N^{\alpha} = \frac{1}{k+\alpha}(A + \alpha I)$. Since, both $\tilde{A}_N$ and $\tilde{A}_N^{\alpha}$ are the linearly scaled transformations of A, then both will share a similar set of eigenvectors with different scaled eigenvalues. Assume v is an eigenvector of $\tilde{A}_N$ associated with any eigenvalue λ_1. Therefore,

$$\begin{aligned} &\tilde{A}_N v = \lambda_1 v \\ &\frac{1}{k+1}(A + I)v = \lambda_1 v \\ &Av = ((k+1)\lambda_1 - 1)v \end{aligned} \tag{4.9}$$

We can say that v is an eigenvector of A with corresponding eigenvalue $\lambda_A = ((k+1)\lambda_1 - 1)$. Consider the eigenvector v of $\tilde{A}_N^{\gamma}$ with the with the eigenvalue λ_2. Then, we have the following,

$$\begin{aligned} &\tilde{A}_N^{\alpha} v = \lambda_2 v \\ &\frac{1}{k+\alpha}(A + \alpha I)v = \lambda_2 v \\ &\lambda_2 = \frac{\lambda_A + \alpha}{k+\alpha} \end{aligned} \tag{4.10}$$

As the range of eigenvalues of $\tilde{A}_N$ is $[-1, 1]$, thus we have $\lambda_1 \leq 1$. The following inequality can be expressed,

$$\begin{aligned} &\lambda_1 \leq 1 \\ &(k+1)\lambda_1 \leq k+1 \\ &(k+1)\lambda_1 - 1 \leq k+1-1 \\ &\lambda_A \leq k \end{aligned} \tag{4.11}$$

Using the inequality we will prove the next stage as

$$\begin{aligned} &\lambda_A \leq k \\ &\alpha + \lambda_A \leq \alpha + k \\ &\frac{\alpha + \lambda_A}{k+\alpha} \leq \frac{\alpha + k}{\alpha + k} \\ &\lambda_2 \leq 1 \end{aligned} \tag{4.12}$$

In this way, we showed that any eigenvalue of $\tilde{A}_N^{\alpha}$ is 1. For the lower bound, we can write,

$$\begin{aligned} \lambda_1 &\geq -1 \\ (k+1)\lambda_1 &\geq -k-1 \\ (k+1)\lambda_1 - 1 &\geq -k-1-1 \\ \lambda_A &\geq -k-2 \end{aligned} \tag{4.13}$$

Using the inequality we will prove the next stage as,

$$\begin{aligned} \lambda_A &\geq -k-2 \\ \alpha + \lambda_A &\geq \alpha - k - 2 \\ \frac{\alpha+\lambda_A}{k+\alpha} &\geq \frac{\alpha-k-2}{\alpha+k} \\ \lambda_2 &\geq 1 - \frac{2(k+1)}{k+\alpha} \end{aligned} \tag{4.14}$$

The degree $k > 1$, then we have $\lambda_2 \geq -1$. Therefore, the addition of self-loops will not alter the range of the eigenvalues of the symmetrically normalized adjacency matrix. □

Lemma 4.4 Given a k-regular graph $\mathcal{G}$, the eigenvalues of $\tilde{A}_N^{\gamma}$ will lie in $[-1, 1]$ $\forall \gamma \geq 1$.

Proof. Consider a k-regular graph $\mathcal{G}$ where each node has a degree k with the normalized adjacency matrix is $\tilde{A}_N = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ where $\tilde{A} = A + I, \tilde{D} = D + I$. If γ-times (with $\gamma \geq 1$) parallel edges are added, then the updated normalized adjacency matrix is $\tilde{A}_N^{\gamma} = \tilde{D}_{\gamma}^{-\frac{1}{2}}\tilde{A}_{\gamma}\tilde{D}_{\gamma}^{-\frac{1}{2}}$ where $\tilde{A}_{\gamma} = (1+\gamma)A+I, \tilde{D}_{\gamma} = (1+\gamma)D+I$. As the graph is regular, then A_N can be presented as:

$$\begin{aligned} \tilde{A}_N &= \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}} \\ &= \frac{1}{\sqrt{k+1}}(A+I)\frac{1}{\sqrt{k+1}} \\ &= \frac{1}{k+1}(A+I) \end{aligned} \tag{4.15}$$

In a similar fashion, we can express $\tilde{A}_N^{\gamma} = \frac{1}{1+(1+\gamma)k}((1+\gamma)A + I)$. Since, both $\tilde{A}_N$ and $\tilde{A}_N^{\gamma}$ are the linearly scaled transformations of A, then both will share a similar set of eigenvectors with different scaled eigenvalues. Assume v is an eigenvector of $\tilde{A}_N$ associated with any

eigenvalue λ_1. Therefore,

$$\begin{aligned} \tilde{A}_N v &= \lambda_1 v \\ \frac{1}{k+1}(A+I)v &= \lambda_1 v \\ Av &= ((k+1)\lambda_1 - 1)v \end{aligned} \tag{4.16}$$

We can say that v is an eigenvector of A with corresponding eigenvalue $\lambda_A = ((k+1)\lambda_1 - 1)$. Consider the eigenvector v of $\tilde{A}_N^{\gamma}$ with the with the eigenvalue λ_2. Then, we have the following,

$$\begin{aligned} \tilde{A}_N^{\gamma} v &= \lambda_2 v \\ \frac{1}{1+(1+\gamma)k}((1+\gamma)A+I)v &= \lambda_2 v \\ \lambda_2 &= \frac{(1+\gamma)\lambda_A + 1}{(1+\gamma)k+1} \end{aligned} \tag{4.17}$$

As the range of eigenvalues of $\tilde{A}_N$ is $[-1, 1]$, thus we have $\lambda_1 \leq 1$. The following inequality can be expressed,

$$\begin{aligned} \lambda_1 &\leq 1 \\ (k+1)\lambda_1 &\leq k+1 \\ (k+1)\lambda_1 - 1 &\leq k+1-1 \\ \lambda_A &\leq k \end{aligned} \tag{4.18}$$

Using the inequality we will prove the next stage as

$$\begin{aligned} \lambda_A &\leq k \\ (1+\gamma)\lambda_A &\leq (1+\gamma)k \\ (1+\gamma)\lambda_A &\leq (1+\gamma)k \\ (1+\gamma)\lambda_A + 1 &\leq (1+\gamma)k + 1 \\ \frac{(1+\gamma)\lambda_A + 1}{(1+\gamma)k+1} &\leq 1 \\ \lambda_2 &\leq 1 \end{aligned} \tag{4.19}$$

In this way, we have shown the maximum eigenvalue of $\tilde{A}_N^{\gamma}$ is 1. The lower bound of the eigenvalues can be shown as

$$\begin{aligned} &\lambda_1 \geq -1 \\ &(k+1)\lambda_1 \geq -k-1 \\ &(k+1)\lambda_1 - 1 \geq -k-1-1 \\ &\lambda_A \leq -k-2 \end{aligned} \tag{4.20}$$

Using the inequality, we will prove the next stage as

$$\begin{aligned} &\lambda_A \leq -k-2 \\ &(1+\gamma)\lambda_A \geq (1+\gamma)(-k-2) \\ &(1+\gamma)\lambda_A + 1 \geq (1+\gamma)(-k-2)+1 \\ &\frac{(1+\gamma)\lambda_A + 1}{(1+\gamma)k+1} \geq \frac{(1+\gamma)(-k-2)+1}{(1+\gamma)k+1} \\ &\lambda_2 \geq 1 - \frac{2(k+1)(\gamma+1)}{(1+\gamma)k+1} \end{aligned} \tag{4.21}$$

As $k, \gamma > 0$, then we have $\lambda_2 \geq -1$. Therefore, the addition of parallel edges will not alter the range of the eigenvalues of the symmetrically normalized adjacency matrix. □

Theorem 4.1 Consider a k-regular graph with $\alpha_1, \alpha_2 \in \mathbb{R}^+$ with $\alpha_1 \leq \alpha_2$, then $\lambda_{\alpha_1}^i \geq \lambda_{\alpha_2}^i, \forall\ 1 \leq i \leq n$, where $\lambda_{\alpha_1}^i$ and $\lambda_{\alpha_2}^i$ are the i^{th} eigenvalues of $\tilde{L}_{\alpha_1}$ and $\tilde{L}_{\alpha_2}$ respectively.

Proof. Consider a k-regular graph $\mathcal{G}$ where each node of degree k with the normalized adjacency matrix is $A_N = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ where $\tilde{A} = A + I, \tilde{D} = D + I$. If α-times (with $\alpha \geq 1$) self-loops are added, then the updated normalized adjacency matrix is $\tilde{A}_N^{\alpha} = \tilde{D}_{\alpha}^{-\frac{1}{2}}\tilde{A}_{\alpha}\tilde{D}_{\alpha}^{-\frac{1}{2}}$ where $\tilde{A}_{\alpha} = A + \alpha I, \tilde{D}_{\alpha} = D + \alpha I$. As the graph is regular then we can have the following

$$\begin{aligned} A_N^{\alpha} &= \tilde{D}_{\alpha}^{-\frac{1}{2}}\tilde{A}_{\alpha}\tilde{D}_{\alpha}^{-\frac{1}{2}} \\ &= \frac{1}{\sqrt{k+\alpha}}(A+\alpha I)\frac{1}{\sqrt{k+\alpha}} \\ &= \frac{1}{(k+\alpha)}(A+\alpha I) \end{aligned} \tag{4.22}$$

Therefore, we can express $A_N^{\alpha_1} = \frac{1}{k+\alpha_1}(A+\alpha_1 I)$ and $A_N^{\alpha_2} = \frac{1}{k+\alpha_2}((A+\alpha_2 I)$. Since $A_N^{\alpha_1}$ and

$A_N^{\alpha_2}$ are the linear transformations of A, both will have the same set of eigenvectors but with different eigenvalues. Assume v is the eigenvector of A and its corresponding eigenvalue is λ_{α_1}. Therefore, the following can be written as

$$\begin{aligned} A_N^{\alpha_1} v &= \lambda_{\alpha_1} v \\ \frac{1}{(k+\alpha_1)}(A+\alpha_1 I)v &= \lambda_{\alpha_1} v \\ Av &= ((k+\alpha_1)\lambda_{\alpha_1} - \alpha_1)v \end{aligned} \tag{4.23}$$

We can say that v is the eigenvector of A with the corresponding eigenvalue $\lambda_A = ((k+\alpha_1)\lambda_{\alpha_1} - \alpha_1)$. For $A_N^{\alpha_2}$, for eigenvector v, the corresponding eigenvalue is λ_{α_2}. Now, we have the following:

$$\begin{aligned} A_N^{\alpha_2} v &= \lambda_{\alpha_2} v \\ \frac{1}{(k+\alpha_2)}(A+\alpha_2 I)v &= \lambda_{\alpha_2} v \\ Av &= ((k+\alpha_2)\lambda_{\alpha_2} - \alpha_2)v \end{aligned} \tag{4.24}$$

Similarly, we can also express $\lambda_A = ((k+\alpha_2)\lambda_{\alpha_2} - \alpha_2)$. Now, equating the two different

expressions of λ_A, we can express the following

$$(k+\alpha_1)\lambda_{\alpha_1} - \alpha_1 = (k+\alpha_2)\lambda_{\alpha_2} - \alpha_2$$

$$k\lambda_{\alpha_1} - \alpha_1(1-\lambda_{\alpha_1}) = k\lambda_{\alpha_2} - \alpha_2(1-\lambda_{\alpha_2})$$

$$k(\lambda_{\alpha_1} - \lambda_{\alpha_2}) = \alpha_1(1-\lambda_{\alpha_1}) - \alpha_2(1-\lambda_{\alpha_2})$$

As per provided condition, we have $\alpha_1 < \alpha_2$

$$\alpha_1(1-\lambda_{\alpha_1}) < \alpha_2(1-\lambda_{\alpha_1})$$

$$\alpha_1(1-\lambda_{\alpha_1}) - \alpha_2(1-\lambda_{\alpha_2}) < \alpha_2(1-\lambda_{\alpha_1}) - \alpha_2(1-\lambda_{\alpha_2})$$

$$k(\lambda_{\alpha_1} - \lambda_{\alpha_2}) < \alpha_2(1-\lambda_{\alpha_1}) - \alpha_2(1-\lambda_{\alpha_2})$$

$$k(\lambda_{\alpha_1} - \lambda_{\alpha_2}) < \alpha_2(\lambda_{\alpha_2} - \lambda_{\alpha_1})$$

$$(k+\alpha_2)(\lambda_{\alpha_!} - \lambda_{\alpha_2}) < 0$$

As $k, \alpha_2 > 0$ then

$$\lambda_{\alpha_!} - \lambda_{\alpha_2} < 0$$

$$\lambda_{\alpha_!} < \lambda_{\alpha_2}$$

The above equation holds for all $|\lambda_{\alpha_1}|, |\lambda_{\alpha_2}| \leq 1$ which is ensured from Lemma 4.3. The eigenvalue of $A_N^{\alpha_2}$ became greater than that of $A_N^{\alpha_1}$ when self-loops are added to the graph. We know that $\tilde{L}_\alpha = \mathrm{I} - \tilde{D}_\alpha^{-\frac{1}{2}} \tilde{A}_\alpha \tilde{D}_\alpha^{-\frac{1}{2}}$, indicates if the eigenvalue of A_N^{α} increases then the corresponding eigenvalue of $\tilde{L}_\alpha$ decreases. Therefore, it can be concluded that the eigenvalue of $\tilde{L}_\alpha$ decreases with the addition of self-loops. □

Theorem 4.2 Consider a k-regular graph with $\gamma_1, \gamma_2 \in \mathbb{R}^+$ with $\gamma_1 \leq \gamma_2$, then $\lambda_{\gamma_1}^i \leq \lambda_{\gamma_2}^i, \forall\ 1 \leq i \leq n$, where $\lambda_{\gamma_1}^i$ and $\lambda_{\gamma_2}^i$ are the i^{th} eigenvalues of $\tilde{L}_{\gamma_1}$ and $\tilde{L}_{\gamma_2}$ respectively.

Proof. Consider a k-regular graph $\mathcal{G}$ where each node has a degree k with the normalized adjacency matrix is $A_N = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ where $\tilde{A} = A + I, \tilde{D} = D + I$. If γ-times (with $\gamma \geq 1$) parallel edges are added, then the updated normalized adjacency matrix is $\tilde{A}_N^{\gamma} = \tilde{D}_\gamma^{-\frac{1}{2}} \tilde{A}_\gamma \tilde{D}_\gamma^{-\frac{1}{2}}$ where $\tilde{A}_\gamma = (1+\gamma)A + I, \tilde{D}_\gamma = (1+\gamma)D + I$. As the graph is regular then we can have the

following

$$\begin{aligned} A_N^{\gamma} &= \tilde{D}_{\gamma}^{-\frac{1}{2}} \tilde{A}_{\gamma} \tilde{D}_{\gamma}^{-\frac{1}{2}} \\ &= \frac{1}{\sqrt{1+(1+\gamma)k}}((1+\gamma)A+I)\frac{1}{\sqrt{1+(\gamma+1)k}} \\ &= \frac{1}{1+(1+\gamma)k}((1+\gamma)A+I) \end{aligned} \tag{4.25}$$

Therefore, we can express $A_N^{\gamma_1} = \frac{1}{1+(1+\gamma_1)k}((1+\gamma_1)A+I)$ and $A_N^{\gamma_2} = \frac{1}{1+(1+\gamma_2)k}((1+\gamma_2)A+I)$. Since $A_N^{\gamma_1}$ and $A_N^{\gamma_2}$ are the scalar transformations of A, both will have the same set of eigenvectors with different eigenvalues. Assume v is the eigenvector of A and its corresponding eigenvalue is λ_{γ_1}. Therefore, the following can be written as

$$\begin{aligned} &A_N^{\gamma_1} v = \lambda_{\gamma_1} v \\ &\frac{1}{1+(1+\gamma_1)k}((1+\gamma_1)A+I)v = \lambda_{\gamma_1} v \\ &Av = \frac{(1+(1+\gamma_1)k)\lambda_{\gamma_1}-1}{1+\gamma_1} v \end{aligned} \tag{4.26}$$

We can say that v is the eigenvector of A with the corresponding eigenvalue $\lambda_A = \frac{(1+(1+\gamma_1)k)\lambda_{\gamma_1}-1}{1+\gamma_1}$. For A_N^{γ}, the eigenvector is v with the eigenvalue λ_2. Then, we have the following

$$\begin{aligned} &A_N^{\gamma_2} v = \lambda_{\gamma_2} v \\ &\frac{1}{1+(1+\gamma_2)k}((1+\gamma_2)A+I)v = \lambda_{\gamma_2} v \\ &Av = \frac{(1+(1+\gamma_2)k)\lambda_{\gamma_2}-1}{1+\gamma_2} v \end{aligned} \tag{4.27}$$

We can also express $\lambda_A = \frac{(1+(1+\gamma_2)k)\lambda_{\gamma_2}-1}{1+\gamma_2}$. Now, equating the two different values of λ_A, we

can have the following

$$\frac{(1+(1+\gamma_1)k)\lambda_{\gamma_1}-1}{1+\gamma_1}=\frac{(1+(1+\gamma_2)k)\lambda_{\gamma_2}-1}{1+\gamma_2}$$

$$(1+\gamma_2)((1+(1+\gamma_1)k)\lambda_{\gamma_1}-1)=$$

$$(1+\gamma_1)(1+(1+\gamma_2)k)\lambda_{\gamma_2}-1)$$

$$(1+\gamma_2)(1+(1+\gamma_1)k)\lambda_{\gamma_1}-(1+\gamma_2)=$$

$$(1+\gamma_1)(1+(1+\gamma_2)k)\lambda_{\gamma_2}-(1+\gamma_1)$$

$$(1+\gamma_2+(1+\gamma_1)(1+\gamma_2)k)\lambda_{\gamma_1}-1-\gamma_2=$$

$$(1+\gamma_1+(1+\gamma_1)(1+\gamma_2)k)\lambda_{\gamma_2}-1-\gamma_1$$

$$(1+\gamma_2+(1+\gamma_1)(1+\gamma_2)k)\lambda_{\gamma_1}=$$

$$(\gamma_2-\gamma_1)+(1+\gamma_1+(1+\gamma_1)(1+\gamma_2)k)\lambda_{\gamma_2}$$

$$\begin{aligned}\lambda_{\gamma_1}=&\frac{\gamma_2-\gamma_1}{(1+\gamma_2+(1+\gamma_1)(1+\gamma_2)k)}+\\&\frac{(1+\gamma_1+(1+\gamma_1)(1+\gamma_2)k)}{(1+\gamma_2+(1+\gamma_1)(1+\gamma_2)k)}\lambda_{\gamma_2}\\=&\frac{\gamma_2-\gamma_1}{(1+\gamma_2+(1+\gamma_1)(1+\gamma_2)k)}+\\&\frac{(1+\gamma_2+(1+\gamma_1)(1+\gamma_2)k)-\gamma_2+\gamma_1}{(1+\gamma_2+(1+\gamma_1)(1+\gamma_2)k)}\lambda_{\gamma_2}\\=&\frac{\gamma_2-\gamma_1}{(1+\gamma_2+(1+\gamma_1)(1+\gamma_2)k)}+\\&(1-\frac{\gamma_2-\gamma_1}{(1+\gamma_2+(1+\gamma_1)(1+\gamma_2)k)})\lambda_{\gamma_2}\\=&\frac{\gamma_2-\gamma_1}{(1+\gamma_2+(1+\gamma_1)(1+\gamma_2)k)}(1-\lambda_{\gamma_2})+\lambda_{\gamma_2}\end{aligned}\tag{4.28}$$

According to the condition provided $\gamma_2 > \gamma_1$ we can say

$\lambda_{\gamma_1} > \lambda_{\gamma_2}$

The eigenvalue of $A_N^{\gamma_2}$ became lesser than that of $A_N^{\gamma_1}$ with the addition of parallel edges. The Eq. 4.28 holds for $|\lambda_{\gamma_2}| \leq 1$ which is assured from Lemma 4.4. We know that $\tilde{L}_\gamma = \mathrm{I} - \tilde{D}_\gamma^{-\frac{1}{2}} \tilde{A}_\gamma \tilde{D}_\gamma^{-\frac{1}{2}}$, indicates if the eigenvalue of A_N^γ decreases then the corresponding eigenvalue of $\tilde{L}_\gamma$ increases. Therefore, it can be concluded that the eigenvalue of $\tilde{L}_\gamma$ increases with the addition of parallel edges for the regular graphs. □

Corollary 4.2. *The increase in the eigenvalues of L_γ is independent of the number of self-loop additions in $\mathcal{G}$. On the contrary, the eigenvalues of $\tilde{L}_\gamma$ will increase if at least one self-loop is added per node in $\mathcal{G}$.*

Proof. Let us prove the statement by contradiction. We know $L = D - A$ and after adding α-times self-loops and γ-times parallel edges, the $L_\gamma = D_\gamma - A_\gamma$ where $\tilde{A}_\gamma = (1+\gamma)A + \alpha I, \tilde{D}_\gamma = (\gamma+1)D + \alpha I$. Then,

$$\begin{aligned} L_\gamma &= (\tilde{D}_\gamma - \tilde{A}_\gamma) \\ &= (((\gamma+1)D + \alpha I) - ((\gamma+1)A + \alpha I)) \\ &= (\gamma+1)(D - A) \\ &= (\gamma+1)L \end{aligned} \tag{4.29}$$

The equation is independent of the number of self-loops we confirm that the effect of adding parallel edges prevails. Similarly, let us also prove the second part by contradiction. We know that $\tilde{L}_\gamma = \mathrm{I} - \tilde{D}_\gamma^{-\frac{1}{2}} \tilde{A}_\gamma \tilde{D}_\gamma^{-\frac{1}{2}}$ and consider the expression without self-loops with the addition of γ-times parallel edges as $\tilde{D} = (1+\gamma)D, \tilde{A} = (1+\gamma)A$.

$$\begin{aligned} \tilde{L}_\gamma &= \mathrm{I} - \tilde{D}_\gamma^{-\frac{1}{2}} \tilde{A}_\gamma \tilde{D}_\gamma^{-\frac{1}{2}} \\ &= \mathrm{I} - \frac{1}{\sqrt{(1+\gamma)}} D^{-\frac{1}{2}} (1+\gamma) A \frac{1}{\sqrt{(1+\gamma)}} D^{-\frac{1}{2}} \\ &= \mathrm{I} - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \\ &= \tilde{L} \end{aligned} \tag{4.30}$$

Therefore, adding parallel edges without at least one self-loop per node does not change the normalized graph Laplacian. Hence, the result is proved. □

Theorem 4.3 Consider a connected graph $\mathcal{G}$ with $A_N = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. Assuming the diagonal of A of G is perturbed by a significantly small $\alpha > 0$, then the updated normalized adjacency matrix will be A_N^α. The change in the eigenvalues of A_N^α with respect to the eigenvalues of A_N will increase when α increases.

Proof. If the diagonal of A is perturbed by α, the symmetrically normalized graph Laplacian

of G will be,

$$\tilde{L}_\alpha = \mathrm{I} - \tilde{D}_\alpha^{-\frac{1}{2}} \tilde{A}_\alpha \tilde{D}_\alpha^{-\frac{1}{2}}, \tag{4.31}$$

where $\tilde{A}_\alpha = A + \alpha\mathrm{I}$ and $\tilde{D}_\alpha = D + \alpha\mathrm{I}$. Let us denote the normalized adjacency matrix without self-loops as $A_N = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. The element of A_N is represented as:

$$A_N[i][j] = \begin{cases} \frac{A_{ij}}{\sqrt{d_i d_j}}, & i \neq j \\ 0, & i = j \end{cases} \tag{4.32}$$

After perturbed by α, the normalized adjacency will be $A_N^\alpha = \tilde{D}_\alpha^{-\frac{1}{2}} \tilde{A}_\alpha \tilde{D}_\alpha^{-\frac{1}{2}}$. The elements of $\tilde{A}_N^\alpha$ can be represented as:

$$A_N^\alpha[i][j] = \begin{cases} \frac{A_{ij}}{\sqrt{\alpha+d_i}\sqrt{\alpha+d_j}} & i \neq j \\ \frac{\alpha}{\alpha+d_i} & i = j \end{cases} \tag{4.33}$$

The entry-wise change in the normalized adjacency matrix is presented as:

$$\delta A_N[i][j] = \begin{cases} \frac{A_{ij}}{\sqrt{\alpha+d_i}\sqrt{\alpha+d_j}} - \frac{A_{ij}}{\sqrt{d_i d_j}}, & i \neq j \\ \frac{\alpha}{\alpha+d_i}, & i = j \end{cases} \tag{4.34}$$

Following the notion of Theorem 4 stated in (Karhadkar et al., 2022) we can assume x is a normalized eigenvector of A_N with corresponding eigenvalue λ. Therefore, the first-order

change in the corresponding eigenvalue can be represented as:

$$\begin{aligned} x^\top(\delta A_N)x &= \sum_{i\neq j}\delta A_N[i][j]x_ix_j + \sum_{i=j}\delta A_N[i][j]x_i^2 \\ &= \sum_{i\neq j}\left(\frac{A_{ij}}{\sqrt{\alpha+d_i}\sqrt{\alpha+d_j}} - \frac{A_{ij}}{\sqrt{d_id_j}}\right)x_ix_j \\ &\quad + \sum_{i=j}\frac{\alpha}{\alpha+d_i}x_i^2 \\ &= \sum_{i\neq j}\left(\frac{1}{\sqrt{\alpha+d_i}\sqrt{\alpha+d_j}} - \frac{1}{\sqrt{d_id_j}}\right)A_{ij}x_ix_j \\ &\quad + \sum_{i=j}\frac{\alpha}{\alpha+d_i}x_i^2 \end{aligned} \tag{4.35}$$

Consider,

$$\begin{aligned} F_{ij}^{(1)}(\alpha) &= \left(\frac{1}{\sqrt{\alpha+d_i}\sqrt{\alpha+d_j}} - \frac{1}{\sqrt{d_id_j}}\right) \\ F_i^{(2)}(\alpha) &= \frac{\alpha}{\alpha+d_i}x_i^2 \end{aligned} \tag{4.36}$$

If $d_i, d_j \gg \alpha$, then $F_{ij}^{(1)}(\alpha) \approx 0$ which lead to

$$x^\top(\delta A_N)x \approx F_i^{(2)}(\alpha) \approx \sum_{i=j}\frac{\alpha}{\alpha+d_i}x_i^2. \tag{4.37}$$

If α increases then $F_i^{(2)}(\alpha)$ also increases. This reflects the change in the eigenvalue of A_N^α increases, indicating the decrease in the eigenvalues of the $\tilde{L}_\alpha$. □

Theorem 4.4 Consider a connected graph $\mathcal{G}$ with normalized adjacency matrix $A_N = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. Assuming each element of A except the diagonal is multiplied by $1+\gamma$ where $\gamma > 0$ is a significantly small quantity, then the updated normalized adjacency matrix will be A_N^γ. The change in the eigenvalues of A_N^γ with respect to the eigenvalues of A_N will decrease when γ increases.

Proof. If the non-diagonal elements of A are multiplied by $1+\gamma$, then the symmetrically

normalized graph Laplacian will be,

$$\tilde{L}_\gamma = \mathrm{I} - \tilde{D}_\gamma^{-\frac{1}{2}} \tilde{A}_\gamma \tilde{D}_\gamma^{-\frac{1}{2}}, \tag{4.38}$$

where $\tilde{A}_\gamma = (\gamma+1)A + \mathrm{I}$ and $\tilde{D}_\gamma = (\gamma+1)D + \mathrm{I}$. Let us denote the normalized adjacency matrix without self-loops as $A_N = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. The element of A_N is represented as:

$$A_N[i][j] = \begin{cases} \frac{A_{ij}}{\sqrt{d_i d_j}}, & i \neq j \\ 0, & i = j \end{cases} \tag{4.39}$$

After multiplying $(1+\gamma)$ to the non-diagonal elements of A and adding one self-loops, the normalized adjacency will be $A_N^\gamma = \tilde{D}_\gamma^{-\frac{1}{2}} \tilde{A}_\gamma \tilde{D}_\gamma^{-\frac{1}{2}}$. The elements of $\tilde{A}_N^\gamma$ can be represented as:

$$A_N^\gamma[i][j] = \begin{cases} \frac{(\gamma+1)A_{ij}}{\sqrt{1+(\gamma+1)d_i}\sqrt{1+(\gamma+1)d_j}} & i \neq j \\ \frac{1}{1+(1+\gamma)d_i} & i = j \end{cases} \tag{4.40}$$

The entry-wise change in the normalized adjacency matrix is presented as:

$$\delta A_N[i][j] = \begin{cases} \frac{(\gamma+1)A_{ij}}{\sqrt{1+(\gamma+1)d_i}\sqrt{1+(\gamma+1)d_j}} - \frac{A_{ij}}{\sqrt{d_i d_j}}, & i \neq j \\ \frac{1}{1+(1+\gamma)d_i}, & i = j \end{cases} \tag{4.41}$$

Following the notion of the Theorem 4 stated in (Karhadkar et al., 2022) we can assume x is a normalized eigenvector of A_N with corresponding eigenvalue λ. Therefore, the first-order

change in the spectral gap can be represented as:

$$\begin{aligned} x^\top(\delta A_N)x &= \sum_{i\neq j}\delta A_N[i][j]x_ix_j + \sum_{i=j}\delta A_N[i][j]x_i^2 \\ &= \sum_{i\neq j}\left(\frac{(\gamma+1)A_{ij}}{\sqrt{1+(\gamma+1)d_i}\sqrt{1+(\gamma+1)d_j}} - \frac{A_{ij}}{\sqrt{d_id_j}}\right)x_ix_j \\ &\quad + \sum_{i=j}\frac{1}{1+(1+\gamma)d_i}x_i^2 \\ &= \sum_{i\neq j}\left(\frac{(\gamma+1)}{\sqrt{1+(\gamma+1)d_i}\sqrt{1+(\gamma+1)d_j}} - \frac{1}{\sqrt{d_id_j}}\right)A_{ij}x_ix_j \\ &\quad + \sum_{i=j}\frac{1}{1+(1+\gamma)d_i}x_i^2 \end{aligned} \tag{4.42}$$

Consider the following,

$$\begin{aligned} F_{ij}^{(1)}(\gamma) &= \left(\frac{(\gamma+1)}{\sqrt{1+(\gamma+1)d_i}\sqrt{1+(\gamma+1)d_j}} - \frac{1}{\sqrt{d_id_j}}\right) \\ F_i^{(2)}(\gamma) &= \frac{1}{1+(1+\gamma)d_i}x_i^2 \end{aligned} \tag{4.43}$$

Now we can rewrite,

$$\begin{aligned} F_{ij}^{(1)}(\gamma) &= \left(\frac{(\gamma+1)}{\sqrt{1+(\gamma+1)d_i}\sqrt{1+(\gamma+1)d_j}} - \frac{1}{\sqrt{d_id_j}}\right) \\ &= \left(\frac{1}{\sqrt{\frac{1}{1+\gamma}+d_i}\sqrt{\frac{1}{1+\gamma}+d_j}} - \frac{1}{\sqrt{d_id_j}}\right) \end{aligned} \tag{4.44}$$

As we mentioned γ is a sufficiently small quantity and with $d_i, d_j \gg 1$, then $F_{ij}^{(1)}(\gamma) \approx 0$. Now we have,

$$x^\top(\delta A_N)x \approx F_i^{(2)}(\gamma) \approx \frac{1}{1+(1+\gamma)d_i}x_i^2 \tag{4.45}$$

If γ increases then $F_i^{(2)}(\gamma)$ decreases reflecting the change in the eigenvalues of $\tilde{A}_N^\gamma$ decreases. This indicates the increase in the eigenvalues of $\tilde{L}_N^\gamma$. □

Chapter 5

Asynchronous Message Passing for Addressing Oversquashing in Graph Neural Networks

Summary

Graph Neural Networks (GNNs) face the potential limitation of Oversquashing that occurs on tasks requiring long-range interactions. The problem arises from the presence of bottlenecks that limit the propagation of messages among distant nodes. Recently, graph rewiring methods directly alter edge connectivity and are expected to perform well on long-range tasks. Yet, graph rewiring compromises the inductive bias, incurring significant information loss in solving the downstream task. Furthermore, increasing channel capacity may increase the number of trainable parameters, enhancing the model complexity. To alleviate the shortcomings, we attempt to design an asynchronous message passing framework to process exponentially growing information with distinct time stamps rather than simultaneously accessing the given channel with a fixed capacity. Our work proposes a framework for asynchronous message propagation that updates node features based on chosen layer-specific batches. The batches are constructed based on the importance of nodes in the graph. We also theoretically found that our proposed framework is capable of addressing oversquashing. We applied our framework to six benchmark graph datasets to perform graph classification and achieved impressive performance over the standard baselines.

5.1 Introduction

Graph Neural Networks (Scarselli et al., 2008; Kipf and Welling, 2016; Veličković et al., 2017; Wu et al., 2019; Chen et al., 2020c; Corso et al., 2020) predominantly aggregate neighboring information in a localized manner. The localized nature of message exchanges seems inadequate for solving downstream tasks like molecular property predictions (Dwivedi et al., 2020) where long-range interaction is necessary. A straightforward way to alleviate the problem is by simply stacking multiple message-passing layers facilitating interaction between the long-distant nodes. However, the solution does not come without it costs of the twin infamous problems of Oversmoothing (Li et al., 2020a) and Oversquashing (Alon and Yahav, 2020; Cai and Wang, 2020b). Oversmoothing is typically attributed to transforming node features into indistinguishable ones due to layer-wise recursive message propagation. The oversquashing, an information bottleneck, stems from the necessity to compress information aggregated from the exponentially growing neighborhoods into fixed-sized vectors.

To eliminate both problems instantly, Graph Transformers (GTs) (Dwivedi and Bresson, 2020) enable message passing between every pair of nodes, dismissing the requirement for stacking multiple layers. Despite the profound success, the training of GTs is overburdened with a larger number of parameters and high training time, which poses computational hurdles to solving the target tasks. As an intermediate solution, the graph rewiring emerges in the landscape. A plethora of rewiring methods like (Topping et al., 2021b; Karhadkar et al., 2022; Arnaiz-Rodríguez et al.; Barbero et al., 2023; Black et al., 2023a) either performs spectral or spatial rewiring of the input graph topology. Yet, the problem of parameters and training time are somehow reduced, and the rewiring still causes the alteration of the graph structure. may incur information loss concerning the inductive bias. In this context, the recent work PANDA (Choi et al., 2024a) proposed expanding the width of the node features with high centrality values. PANDA typically chooses a set of high-centrality nodes and expands their feature vectors. Subsequently, they select top-ranked features containing sufficient information to maintain consistency in feature dimensions. This approach still discards some of the feature dimensions, compromising relevant information and increasing the parameters while attempting to mitigate oversquashing.

In this work, we attempt to design a message-passing strategy that efficiently addresses

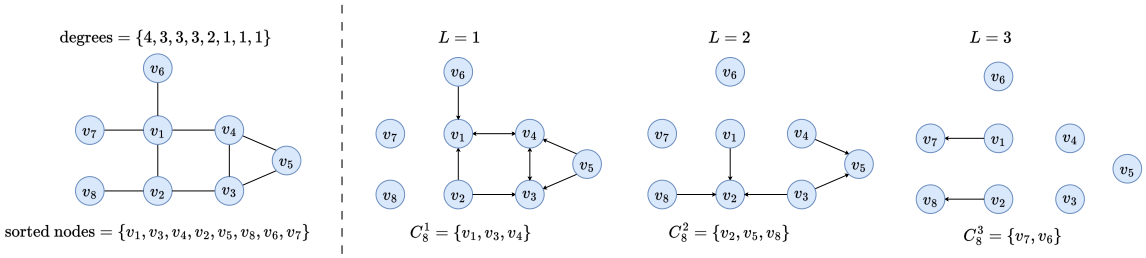


Figure 5.1: The working mechanism of CAMP is presented. Initially, the node centrality values are precomputed (here it is degree) and sorted in descending order. A 3-layered GNN model is applied and the centrality set is divided into three disjoint subsets. In each layer, the features of the nodes belonging to the pertinent subset are updated. The structure of the graph is changed at every layer but the updated node features are carried forward to the next layer.

oversquashing without affording the risk of information loss. Revisiting the core reason for oversquashing, we aim to process the aggregated messages by accessing the channels of fixed capacity in a fixed interval of time stamps or delays rather than processing the exponentially growing information simultaneously. This strategy leads us to explore the asynchronous message-passing paradigm. In this connection, we laid the foundation of the asynchronous message-passing framework by proposing **R**andomized **A**synchronous **M**essage **P**assing or **RAMP** which asynchronously update node features, unlike standard Message Passing GNNs (MP-GNNs) like GCN, GAT, GCNII, etc. At every layer, a batch of nodes is chosen randomly whose node features will be updated and the features of the rest of the nodes remain unchanged. As RAMP pursues layer-wise random node selection, some nodes may never be selected during propagating through hidden layers, which may impact the performance.

We overcome the limitation by proposing another framework **C**entrality-aware **A**synchronous **M**essage **P**assing or **CAMP** that selects the batch of nodes guided by the node centrality or the importance of the nodes in the graph (like degree, betweenness, etc). In stark contrast to RAMP, CAMP performs ordering (either ascending or descending) of the node centrality values and partitions the whole set into multiple disjoint subsets which is equal to the number of message-passing layers. The complete workflow is illustrated in Figure 5.1. From the beginning, each subset is assigned as a batch of nodes for the corresponding layer, ensuring each node gets a chance to exchange messages with its neighbors. Empirically, we have

shown that CAMP is more effective than RAMP in solving the target task. Additionally, the effect of centrality values gradually diminishes (or enhances) while progressing through message-passing layers for the descending (ascending) ordering. The asynchronous feature updates in a fixed channel capacity are poised to accommodate exponentially growing information, ultimately diminishing the effect of oversquashing. Unlike PANDA, our method does not require increasing the dimension of the node features, asserting the learning in a low-parameter regime. Notably, PANDA performs synchronous message updates, which is a striking difference from CAMP.

Contribution Our contributions are summarized as follows,

- **Asynchronous Message Passing** We design the generalized framework RAMP for performing asynchronous message passing, unlike standard MP-GNNs. RAMP samples the batches of nodes randomly at every layer and updates the features. To improve the node selection criteria we introduce CAMP that samples layer-wise node batches guided by the centrality of the nodes. Our proposed frameworks are flexible and can be seamlessly plugged with a diverse set of GNN models
- **Oversquashing Mitigation** Asynchronous message propagation allows the processing of larger volumes of information in different time slots for a fixed channel capacity. This approach effectively utilizes fixed-sized features by successfully avoiding the processing of an exponential amount of information simultaneously, ultimately tackling oversquashing.
- **Theoretical Insights** We theoretically proved (in Lemma 5.1) that CAMP is more powerful than RAMP to address oversquashing in GNNs.

5.2 Proposed Method

5.2.1 Preliminaries & Notations

Assume an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X, A)$ where $\mathcal{V}$ denotes the set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the edge set, $X \in \mathbb{R}^{n \times d}$ is the feature matrix containing d-dimensional node features, and $A \in \mathbb{R}^{n \times n}$ is the adjacency matrix. The neighborhood of a node v is represented as $N(v)$.

Revisiting MP-GNN The existing MP-GNN computes messages between the pair of adjacent nodes (u, v) by a `MESSAGE` function Ψ, and subsequently messages are aggregated across the neighborhood $N(v)$. The features of the centering node v are updated by a `COMBINE` function Φ. Notably, the feature updates are performed simultaneously, and thereby MP-GNN performs "synchronous" message passing across neighborhoods. The updated features of node v is $x_v^{(l+1)}$ which is represented as follows,

$$x_v^{(l+1)} = \Phi_l \left(x_v^{(l)}, \sum_{u \in N(v)} \Psi_l(x_v^{(l)}, x_u^{(l)}) \right) \tag{5.1}$$

where Ψ_l and Ψ_l ar respectively layer-specific message and combine function.

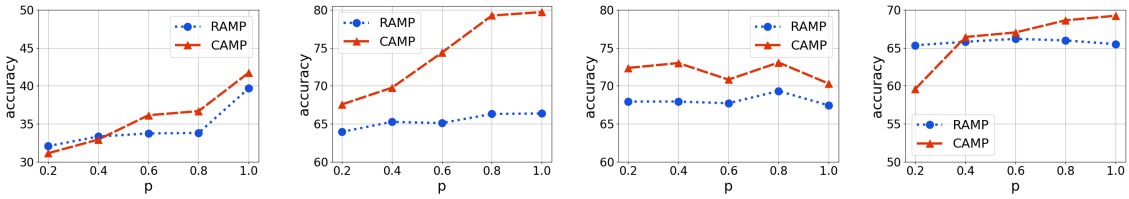
Oversmoothing The problem mostly persists in the deep MP-GNNs causing performance degradation when long-range interaction is required (Oono and Suzuki, 2019; Cai and Wang, 2020b). As an effect, node features became indistinguishable influenced by the recursive neighborhood aggregation in the deeper layers. The growing receptive field in deep layers became almost identical and thereby computes the similar embeddings for each node, resulting in oversmoothing. In this work, we will demonstrate that asynchronous message passing may alleviate oversmoothing in deep GNNs.

Oversquashing The oversquashing phenomenon is first observed by (Alon and Yahav, 2020) originated from GNNs leveraging long-range dependencies. Compressing aggregated features from a large number of neighbors into a fixed capacity of vectors results in information loss. The effect of oversquashing can be measured by the sensitivity bounds proposed by (Di Giovanni et al., 2023b; Topping et al., 2021a). assuming there exists a l-length path between u and v. The sensitivity bound is estimated as the Jacobian of $h_v^{(l)}$ and $h_u^{(0)}$ which can be represented as,

$$\left\| \frac{\partial h_v^{(l)}}{\partial h_u^{(0)}} \right\|_1 \leq \underbrace{(cwp)^l}_{\text{model}} \underbrace{(\tilde{S}^l)_{uv}}_{\text{topology}} . \tag{5.2}$$

The effect of oversquashing depends on the c the Lipschitz constant of the model parameters, w the maximum entry of weight matrices, and width p. Also, oversquashing is affected by the $\tilde{S}^l$ where $\tilde{S}$ can be normalized adjacency matrix. The lower value of the bound signals diminished the effect on the message propagation of a node by its neighbors.

At this stage, we will lay the foundation of the asynchronous message-passing frame-



(a) 4-layered GIN on ENZYMES (b) 10-layered GCN on REDDIT-BINARY (c) 4-layered GIN on PROTEINS (d) 10-layered GCN on COLLAB

Figure 5.2: A comparative study is conducted between the performance of RAMP and CAMP on four molecular graphs with various settings. The analysis reveals CAMP outperforms RAMP exhibiting the influence of node centrality in the message-passing steps.

work and subsequently propose centrality-guided asynchronous message propagation that is supposed to tackle oversquashing.

5.2.2 Asynchronous Message Passing Framework

We propose RAMP which performs asynchronous node updates during the propagation through message-passing layers. Initially, a set of nodes say B with $|\mathrm{B}| \leq n$ is selected randomly. The features of nodes in B will only be updated by neighborhood aggregation and the remaining node features will be carried forward to the next layer. Propagation through hidden layers can be perceived as an increase in layers and also an increase in the time stamp. If the feature of a node $v \in B$ is updated at l^{th} layer then it crosses the time-stamp l. Every node is tagged with a tuple representing layer number and corresponding time-stamp. The following equation represents the asynchronous message-passing operation,

$$x_v^{(l+1,l+1)} = \begin{cases} \Phi_l\left(x_v^{(l_v,l)}, \sum\limits_{u \in N(v)} \Psi_l(x_v^{(l_v,l)}, x_u^{(l_u,l)})\right), \\ \qquad\qquad \forall v \in \mathrm{B} \\ x_v^{(l_v,l+1)}, \ \forall v \notin \mathrm{B}. \end{cases} \tag{5.3}$$

where Ψ_l, Φ_l are respectively `MESSAGE` and `COMBINE` functions for l^{th} layer, l_v denotes the layer at which x_v is last updated with $0 \leq l_v \leq l$. If a node does not participate in the message-passing operation, then its time-stamp will increase by one but the last updated layer remains unchanged. Importantly, RAMP considers node set which not necessarily should be part of any subgraph, strictly differing from the methods pursuing subgraph sampling like (Hamilton

et al., 2017; Zeng et al., 2019)

Limitations of RAMP One major drawback of RAMP is that some nodes may never get selected to interact in the message propagation, acting as isolated nodes in the intermediate layers. We term them "lazy nodes" and the set "lazy set." The features of the lazy nodes are not updated during the entire propagation in the hidden layers. The exclusion of such nodes distorts the graph structure, causing information loss and performance degradation. In the following section, we propose another asynchronous framework to mitigate the shortcoming.

5.2.3 Centrality-aware Asynchronous Message Passing

We design a novel strategy to select mode batches prioritizing the importance of the nodes in a given network. In this context, the centrality values of nodes are considered as the selection criteria for creating node batches. We adopt five different well-recognized centrality measures Degree (Borgatti and Halgin, 2011), Betweenness (Freeman, 1977), Closeness (Freeman et al., 2002), Load (Goh et al., 2001), and PageRank (Page et al., 1999). The centrality scores of nodes assign the nodes to be chosen but do not provide any idea of when to select. We offer an approach to tackle the missing time component during the node selection by tagging each node with a particular layer. Integrating node centrality and message-passing layers solves the problem of selecting node batches.

Initially, we estimate the centrality scores of each node in the pre-processing stage. Concurrently, we also assign ordering (either ascending or descending) of the centrality values which confirms the monotone effects of the importance of nodes and also reduces randomness. For instance, the centrality of nodes in descending order can be presented as follows,

$$C_n = \{c^{(1)}, c^{(2)}, \cdots, c^{(n)} : c^{(i+1)} \geq c^{(i)}, \forall\ 1 \leq i \leq n-1\}, \tag{5.4}$$

where $c^{(i)}$ denotes the centrality of the node i. To ensure the participation of every node at some layer, we divide the centrality values into the L number of smaller subsets where L denotes the total number of layers in the base GNN. Note that, each subset maintains a similar order to C_n. The l^{th} subset can be presented as,

$$C_n^{(l)} = \{c_l^{(1)}, c_l^{(2)}, \cdots, c_l^{(m)} : c_l^{(i+1)} \geq c_l^{(i)}, \forall\ 1 \leq i \leq m-1\} \tag{5.5}$$

where $c_k^{(i)}$ is the centrality of i^{th} node which belong to the l^{th} subset and $m = |C_n^{(l)}| = \lfloor \frac{C_n}{L} \rfloor$ with $l \in [1, L]$.

Node-batch Selection We select a batch of nodes at every layer to update features. For layer l, we consider the subset C_n^l as our candidate node batch. This way of selection ensures the participation of each node in the message-passing operation exactly once. Notably, layer-wise batching of nodes enables centrality-dependent delays in interacting across the neighborhoods. We also offer a provision for selecting a portion of nodes from each subset C_n^l. Assuming a sampling probability is p, then the size of the node batch will be $mp = |C_n^l|p$ where $0 < p \leq 1$. We term the framework as p-CAMP. If we sample a full batch at every layer, then our framework is termed as 1-CAMP or simply CAMP.

Update Rule The feature update rule of CAMP can be defined as follows,

$$x_v^{(l+1,l+1)} = \begin{cases} \Phi_l \left(x_v^{(l_v,l)}, \sum\limits_{u \in N(v)} \Psi_l(x_v^{(l_v,l)}, x_u^{(l_u,l)}) \right), \\ \qquad\qquad \forall v \in C_n^{(l)} \\ x_v^{(l_v,l+1)}, \ \forall v \notin C_n^{(l)}. \end{cases} \tag{5.6}$$

All notations carry similar meanings as discussed earlier.

Why CAMP Works? The disjoint partition of node centrality values has a direct impact on the information propagation and feature updates within the framework. The features of a candidate node batch get updated by aggregating features from their neighboring nodes. These updates rely on older versions of neighboring features, which are stored in channels of fixed capacity. Additionally, newly updated features may remain unchanged for a period and can be utilized by other node batches in subsequent layers. This approach ensures efficient utilization of the channels to store and process information with some delays or at regular intervals. The time-dependent access to these channels optimizes resource utilization, facilitating the flow of relevant information across the network. Moreover, this mechanism prevents the storage of exponentially growing information in fixed-sized channels, thereby mitigating the effects of oversquashing.

Performance Comparison between RAMP and CAMP We performed an initial study to compare the prowess of RAMP and CAMP in mitigating oversquashing on the molecular

graphs. Refer to Figure 5.2 for a detailed illustration. In different settings, the performance of CAMP consistently outperforms RAMP over various batch sampling rates. The performance of RAMP is exacerbated due to the presence of lazy nodes. The lazy nodes fail to provide the necessary features and structural information to the other active nodes. Conversely, CAMP guarantees each node's participation in the aggregation process, thereby enriching it with appropriate feature and structural information.

5.2.4 Theoretical Analysis

In this section, we provide theoretical underpinnings to validate the efficacy of CAMP in mitigating oversquashing.

Lemma 5.1 (Asynchronous Sensitivity Bound for CAMP)**.** *Consider an L-layered GNN applied to a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ integrated with framework CAMP, where Φ_ℓ and Ψ_ℓ are differentiable* `COMBINE` *and* `MESSAGE` *functions at the ℓ-th layer. Assume that for every layer ℓ, the Jacobians of Φ_ℓ and Ψ_ℓ are Lipschitz bounded by,*

$$\|\nabla \Phi_\ell\| \leq \alpha, \qquad \|\nabla \Psi_\ell\| \leq \beta, \tag{5.7}$$

where α, β are real constants.

Then, for any pair of nodes $u, v \in \mathcal{V}$, the feature sensitivity under CAMP satisfies

$$\left| \frac{\partial x_v^{(L)}}{\partial x_u} \right| \leq (\alpha\beta)^{K(u,v)} \sum_{\pi \in \mathcal{P}_{async}(u,v)} \prod_{t \in \pi} S_{i_{t+1} i_t}^{(\tau_t)}, \tag{5.8}$$

where $\mathcal{P}_{async}(u, v)$ is the set of all valid asynchronous walks from u to v respecting the activation schedule $\{C_n^{(\ell)}\}_{\ell=1}^{L}$, and $K(u, v) \leq L$ denotes the number of layers in which the nodes along the path π were updated.

Proof. We compare the asynchronous update rule (**??**) with the classical synchronous message passing update:

$$h_v^{(\ell+1)} = \Phi_\ell\Big(h_v^{(\ell)}, \sum_u S_{vu}\, \Psi_\ell(h_v^{(\ell)}, h_u^{(\ell)})\Big), \tag{5.9}$$

with bound $\left| \frac{\partial h_v^{(L)}}{\partial x_u} \right| \leq (\alpha\beta)^L (S^L)_{uv}$..

Recursive derivative for CAMP. Differentiating (??) with respect to x_s gives

$$\frac{\partial x_v^{(\ell+1,\ell+1)}}{\partial x_s} = \begin{cases} \nabla_1 \Phi_\ell \dfrac{\partial x_v^{(\ell_v,\ell)}}{\partial x_s} \\ \quad + \nabla_2 \Phi_\ell \displaystyle\sum_u S_{vu}^{(\ell)} M_{ij}^{(\ell)}, & v \in C_n^{(\ell)}, \\ \frac{\partial x_v^{(\ell_v,\ell+1)}}{\partial x_s}, & v \notin C_n^{(\ell)}. \end{cases} \tag{5.10}$$

where $M_{ij}^{(l)} = \left(\nabla_1 \Psi_\ell \frac{\partial x_v^{(\ell_v,\ell)}}{\partial x_s} + \nabla_2 \Psi_\ell \frac{\partial x_u^{(\ell_u,\ell)}}{\partial x_s}\right)$. Applying the Lipschitz bounds (5.7) yields

$$\left|\frac{\partial x_v^{(\ell+1,\ell+1)}}{\partial x_s}\right| \leq \begin{cases} \alpha \left|\dfrac{\partial x_v^{(\ell_v,\ell)}}{\partial x_s}\right| \\ \quad + \alpha\beta \displaystyle\sum_u S_{vu}^{(\ell)} \left|\frac{\partial x_u^{(\ell_u,\ell)}}{\partial x_s}\right|, & v \in C_n^{(\ell)}, \\ \left|\frac{\partial x_v^{(\ell_v,\ell)}}{\partial x_s}\right|, & v \notin C_n^{(\ell)}. \end{cases} \tag{5.11}$$

Matrix form. Define the indicator matrix $M^{(\ell)} = \mathrm{diag}(m_v^{(\ell)})$, where $m_v^{(\ell)} = 1$ if $v \in C_n^{(\ell)}$, and 0 otherwise. Then (5.11) can be written compactly as

$$\boldsymbol{\delta}^{(\ell+1)}(s) \leq \left[I - M^{(\ell)} + \alpha M^{(\ell)}(I + \beta S^{(\ell)})\right] \boldsymbol{\delta}^{(\ell)}(s), \tag{5.12}$$

where $\boldsymbol{\delta}^{(\ell)}(s)$ has entries $\delta_v^{(\ell)}(s) = \left|\frac{\partial x_v^{(\ell)}}{\partial x_s}\right|$.

Propagating through L layers. Iterating (5.12) from the input layer ($\boldsymbol{\delta}^{(0)}(s) = \mathbf{e}_s$) gives

$$\boldsymbol{\delta}^{(L)}(s) \leq \prod_{\ell=1}^{L} \left[I - M^{(\ell)} + \alpha M^{(\ell)}(I + \beta S^{(\ell)})\right] \mathbf{e}_s. \tag{5.13}$$

Since inactive nodes preserve their previous features, the only contributing terms correspond to *valid asynchronous walks* that traverse nodes updated at each relevant layer τ_t. Collecting all these paths yields (5.8).

Comparing with synchronous sensitivity. In the synchronous setting, each layer multiplies by the same S, giving $(S^L)_{uv}$ with spectral radius $\rho(S) \leq 1$. In CAMP, the product

involves distinct $S^{(\tau_t)}$ acting on smaller induced subgraphs. For each layer ℓ,

$$\rho(S^{(\ell)}) = \frac{d_{\text{avg}}^{(\ell)}}{d_{\text{max}}^{(\ell)}} > \frac{d_{\text{avg}}}{d_{\text{max}}} = \rho(S), \tag{5.14}$$

since batching reduces maximum degree $d_{\text{max}}^{(\ell)}$. Moreover, the effective depth $K(u,v) \leq L$ because not all layers affect a given node pair. Thus, for $0 < \alpha\beta \leq 1$,

$$(\alpha\beta)^{K(u,v)} \prod_{t=1}^{K(u,v)} \rho(S^{(\tau_t)}) \geq (\alpha\beta)^L \rho(S)^L, \tag{5.15}$$

which implies entrywise

$$(\alpha\beta)^{K(u,v)} \Big(\prod_{t=1}^{K(u,v)} S^{(\tau_t)} \Big)_{uv} \geq (\alpha\beta)^L (S^L)_{uv}. \tag{5.16}$$

Comparison with synchronous case. Equation (5.16) guarantees that CAMP maintains a *higher sensitivity bound* than the synchronous counterpart, whenever at least one of the following holds:

(i) $K(u,v) < L$ (not all layers affect the $u \to v$ path), or

(ii) $\rho(S^{(\ell)}) > \rho(S)$ for some ℓ (stronger local connectivity).

Hence, the oversquashing effect, which corresponds to the exponential decay of $(S^L)_{uv}$ in depth L is alleviated since the asynchronous term decays more slowly.

□

5.2.5 Properties of CAMP

Layer-wise Rewiring CAMP updates the features of node batches during propagation through hidden layers. The candidate node batch aggregates features of the immediate neighbors, and the rest of the node features remain unaltered. This process can be perceived as the edges involving the candidate node batch remaining connected, and the rest of the nodes becoming isolated. When we proceed to the next layer, the edges of the new candidate batch are preserved, and the edges connecting the previous node batch are removed. The structure

Table 5.1: The performances of GCN and GIN coupled with CAMP are presented. For each dataset, the best and second-best results are respectively marked in green and blue colors.

Method	ENZYMES	MUTAG	PROTEINS	COLLAB	IMDB-BINARY	REDDIT-BINARY
GCN (None)	27.66 ± 1.16	72.15 ± 2.44	70.98 ± 0.73	33.78 ± 0.48	49.77 ± 0.81	68.25 ± 1.09
+ Last Layer FA	26.46 ± 1.20	70.05 ± 2.02	71.01 ± 0.96	33.32 ± 0.43	48.98 ± 0.94	68.48 ± 0.94
+ Every Layer FA	18.33 ± 1.03	70.45 ± 1.96	60.03 ± 0.92	51.79 ± 0.41	48.17 ± 0.80	48.49 ± 1.04
+ DIGL	27.51 ± 1.05	71.35 ± 2.39	70.60 ± 0.73	53.35 ± 0.29	49.91 ± 0.84	49.98 ± 0.68
+ SDRF	28.36 ± 1.17	71.05 ± 1.87	70.92 ± 0.79	33.44 ± 0.47	49.40 ± 0.90	68.62 ± 0.85
+ FoSR	25.06 ± 0.99	80.00 ± 1.57	73.42 ± 0.81	33.83 ± 0.58	49.66 ± 0.86	70.33 ± 0.72
+ BORF	24.70 ± 1.00	75.80 ± 1.90	71.00 ± 0.80	Time-out	50.10 ± 0.90	Time-out
+ GTR	27.52 ± 0.99	79.10 ± 1.86	72.59 ± 2.48	33.05 ± 0.40	49.92 ± 0.99	68.99 ± 0.61
+ CT-Layer	17.38 ± 1.03	75.89 ± 3.02	60.35 ± 1.06	52.14 ± 0.41	50.32 ± 0.94	51.58 ± 1.01
+ PANDA	**31.55 ± 1.23**	**85.75 ± 1.39**	**76.00 ± 0.77**	**68.40 ± 0.45**	**63.76 ± 1.01**	**80.69 ± 0.72**
+ GOKU	27.60 ± 1.20	81.00 ± 2.00	71.90 ± 0.80	-	-	-
+ CAMP (Ours)	**31.73 ± 2.30**	**81.80 ± 3.68**	**74.43 ± 1.64**	**69.86 ± 0.70**	**62.44 ± 2.15**	**84.96 ± 0.86**
GIN (None)	33.80 ± 0.11	77.70 ± 0.36	70.80 ± 0.82	72.99 ± 0.38	70.18 ± 0.99	86.78 ± 1.05
+ Last Layer FA	44.40 ± 1.38	83.05 ± 1.74	72.30 ± 0.66	71.05 ± 0.40	70.91 ± 0.78	88.31 ± 0.65
+ Every Layer FA	28.38 ± 1.05	72.55 ± 3.01	70.37 ± 0.91	32.98 ± 0.39	49.16 ± 0.87	50.36 ± 0.68
+ DIGL	35.71 ± 1.19	79.70 ± 2.15	70.75 ± 0.77	54.50 ± 0.41	64.39 ± 0.90	76.03 ± 0.77
+ SDRF	35.81 ± 1.00	78.40 ± 2.80	69.81 ± 0.79	72.95 ± 0.41	69.72 ± 1.15	86.44 ± 0.59
+ FoSR	29.20 ± 1.36	78.40 ± 2.80	73.10 ± 0.81	73.27 ± 0.41	71.21 ± 0.91	87.35 ± 0.59
+ BORF	35.50 ± 1.20	80.80 ± 2.50	73.70 ± 0.80	Time-out	71.30 ± 1.50	Time-out
+ GTR	30.57 ± 1.42	77.60 ± 2.84	73.13 ± 0.69	72.93 ± 0.42	71.28 ± 0.86	86.98 ± 0.66
+ CT-Layer	16.58 ± 0.90	56.85 ± 4.25	61.10 ± 1.18	52.30 ± 0.60	50.00 ± 0.97	54.58 ± 1.75
+ PANDA	**46.20 ± 1.41**	**88.75 ± 1.57**	**75.75 ± 0.85**	**75.11 ± 0.21**	**72.56 ± 0.91**	**91.05 ± 0.40**
+ GOKU	33.80 ± 1.20	78.40 ± 2.50	73.90 ± 1.00	-	-	-
+ CAMP (Ours)	**46.60 ± 2.26**	**87.40 ± 3.95**	**76.07 ± 1.73**	**74.14 ± 0.65**	**73.48 ± 1.47**	**89.24 ± 1.03**

Table 5.2: Results of GCN with None, SDRF, FoSR, BORF, and PANDA on PEPTIDES-FUNC and PEPTIDES-STRUCT. The best and second-best results are respectively marked in green and blue colors.

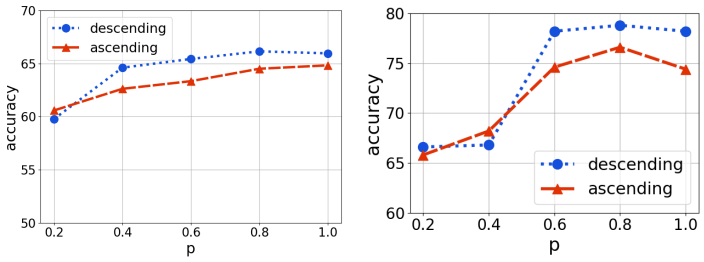
Method	PEPTIDES-FUNC(AP ↑)	PEPTIDES-STRUCT(MAE ↓)
GCN (None)	59.30 ± 0.23	0.3496 ± 0.0013
+ SDRF	59.47 ± 1.26	0.3478 ± 0.0013
+ FoSR	59.47 ± 0.35	0.3473 ± 0.0007
+ BORF	**59.94 ± 0.37**	0.3514 ± 0.0009
+ PANDA	**60.28 ± 0.31**	**0.3272 ± 0.0001**
+ CAMP (Ours)	56.67 ± 0.76	**0.3138 ± 0.0041**

of the graph alters in each layer according to the selected node batches, unlike the existing sequential rewiring methods (Barbero et al., 2023; Karhadkar et al., 2022), etc. Therefore, CAMP leverages layer-specific rewiring and enables asynchronous message aggregation.

Variable Hop Aggregation CAMP simulates a variable hop aggregation strategy, unlike the standard MP-GNNs. For any layer l, the corresponding candidate node batch C_l^n updates features from the neighbors which can be either already updated in some previous layers ($< l$) or will be updated in the upcoming layers ($> l$). The updated features of the current

node batch are the aggregated form of variable hops, updated features of their corresponding neighbors. This variable hop aggregation is the key characteristic of the asynchronous message-passing frameworks.

Order matters: Ascending or Descending The effectiveness of CAMP can be influenced by the order of the centrality values of the nodes. The descending order of the centrality values enables the aggregation of rich information through the high centrality nodes in earlier layers. Those updated node features are further utilized by lower centrality nodes, and information propagation is efficiently leveraged. Conversely, for the ascending order of centrality values, the lower centrality nodes will not be able to collect structural information like the high centrality nodes mentioned in the previous scenario. However, the features aggregated by those nodes may not be sufficient to provide the necessary information to higher centrality nodes. In this context, we experimented to study the impact of ordering on the performance of CAMP, which is demonstrated in Figure 5.3. The empirical analysis reveals that the descending order of set C_n yields better performance over the same for the descending order. Furthermore, nodes with high centrality connect a larger volume of edges, causing oversmoothing in deeper layers. For a descending order of C_n, the faster interaction of high-centrality nodes compared to lower ones is supposed to diminish the effect of oversmoothing.



(a) 4-layered GCN on COLLAB (b) 10-layered GIN on MUTAG

Figure 5.3: The comparative study between the order of node centrality values on the performance of CAMP is presented. The performance is executed on the various batch sampling rates.

5.2.6 Complexity Analysis

The time complexity of CAMP-GNN is at most that of its base GNN because node features are updated asynchronously of the batches in every layer. This approach seems to be advantageous compared to the rest of the approaches. Apart from message propagation, CAMP only

adds an extra task of computing centrality scores performed as a pre-processing step. Notably, the cost of computing centrality values further depends on the choice of centrality, like the cost of degree centrality is $\mathcal{O}(|\mathcal{E}|)$, the cost of measuring closeness is $\mathcal{O}(|\mathcal{V}||\mathcal{E}|)$, etc. Unlike PANDA, we do not increase the number of parameters during the message propagation, thus, the parameter complexity also remains unaffected.

Table 5.3: A comparative study of the performance of CAMP-GCN based on five centrality measures. The best results are boldfaced.

Centrality	ENZYMES	MUTAG	PROTEINS	COLLAB	IMDB-BINARY	REDDIT-BINARY
Degree	28.93 ± 2.65	78.40 ± 4.46	72.00 ± 1.75	69.25 ± 0.71	58.52 ± 2.34	80.10 ± 0.90
Betweenness	30.20 ± 2.56	78.20 ± 4.20	71.54 ± 1.58	69.81 ± 0.82	58.24 ± 2.32	79.90 ± 0.94
PageRank	29.40 ± 2.37	$\mathbf{78.40 \pm 4.07}$	71.14 ± 1.90	69.75 ± 0.86	$\mathbf{60.48 \pm 1.89}$	81.28 ± 1.14
Load	31.13 ± 2.22	78.20 ± 4.16	72.11 ± 1.98	$\mathbf{69.86 \pm 0.70}$	59.32 ± 1.95	79.94 ± 1.06
Closeness	$\mathbf{31.73 \pm 2.30}$	77.60 ± 3.88	$\mathbf{74.43 \pm 1.64}$	69.25 ± 0.83	59.52 ± 2.14	$\mathbf{84.96 \pm 0.86}$

5.2.7 Instances of CAMP

We will now illustrate the examples of CAMP-GNN by using GCN (Kipf and Welling, 2016) and GIN (Xu et al., 2018a). The layer-wise update rule of CAMP-GCN is,

$$h_u^{(l+1,l+1)} = h_u^{(l,l)} + \sigma\Big(\sum_{v \in N(v) \cap C_n^{(l+1)}} W^{(l+1)} h_v^{(l_v,l)}\Big), \tag{5.17}$$

where $W^{(l+1)}$ is the parameterized weight matrix. Observe that neighborhood aggregation is performed on the nodes that are neighbors and belong to the candidate node batch. Similarly, we define the update rule of CAMP-GIN as follows,

$$h_u^{(l+1,l+1)} = \text{MLP}_l\Big((1+\epsilon)h_u^{(l,l)} + \sum_{v \in N(v) \cap C_n^{(l+1)}} W^{(l+1)} h_v^{(l_v,l)}\Big), \tag{5.18}$$

where MLP_l denotes the combination of the linear layers with ReLU activation and ϵ is the learnable parameter. In this case, the node features are also considered for aggregation if they belong to both the neighborhood and candidate batch.

5.3 Experiments

5.3.1 Datasets

For graph classification, we considered six benchmark datasets, ENZYMES, MUTAG, PROTEINS, COLLAB, IMDB-BINARY, and REDDIT-BINARY, obtained from TUDatasets (Morris et al., 2020). The details of the datasets are provided in Table 5.4.

Table 5.4: Details of six datasets are provided, which are obtained from TUDatasets.

Dataset	#train	#valid	#test	#avg nodes	#avg edges	metric	node feats
ENZYMES	480	60	60	32.63	124.27	accuracy	yes
MUTAG	150	18	20	17.93	39.58	accuracy	yes
PROTEINS	890	111	112	39.05	145.63	accuracy	yes
COLLAB	4000	500	500	74.49	4914.43	accuracy	no
IMDB-BINARY	800	100	100	19.77	193.66	accuracy	no
REDDIT-BINARY	1600	200	200	429.62	995.51	accuracy	no

5.3.2 Experimental Settings

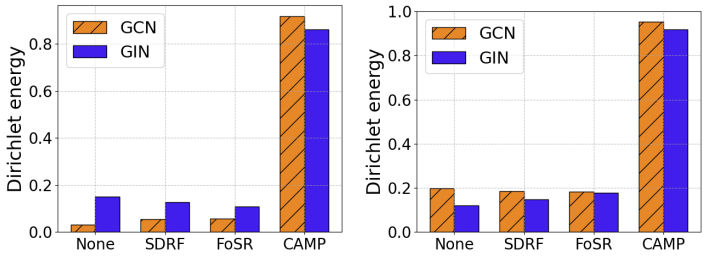
We applied CAMP coupled with GCN and GIN to conduct the experiments. We compare performance with no rewiring method and several baselines Fully Adjacent (FA), DIGL, SDRF, FoSR, BORF, GTR, CT-layer, and PANDA. We split each dataset into 80/10%/10% for the train. validation, and test respectively. We executed experiments for 25 randomly generated splits and produced mean and standard deviations over the test sets. More details on experimental setup and hyperparameters are provided respectively in Section 5.5.1 and Section 5.5.2 of the Appendix. Our implementation is available at https://anonymous.4open.science/r/camp-ABCF/README.md.

5.3.3 Results & Discussions

The performance of CAMP paired with GCN and GIN is presented in Table 5.1. The results suggest that our proposed framework either outperforms all contenders or attains a second position, but outperforms at least all rewiring methods. The consistent performance of CAMP establishes that oversquashing can be tackled by enabling asynchronous message propagation without distorting the original graph structure. CAMP showed commendable improvements on REDDIT-BINARY, containing graphs with larger diameters, where oversquashing is highly likely to occur. Additionally, CAMP also exhibited improvements in

COLLAB, which also contains the bigger graphs. CAMP is ranked as runner-up on MUTAG, PROTEINS, and IMDB-BINARY due to their smaller diameters with a very low chance of occurring oversquashing. In this context, our primary objective was to design an efficient approach beyond rewiring to mitigate oversquashing, and CAMP effectively achieved that, comparable to non-rewiring methods like PANDA.

The quantitative results are provided in Table 5.2. Furthermore, CAMP outperforms Peptide-struct as its features delineate structural information that differs from the features of Peptides-func, representing functional properties.

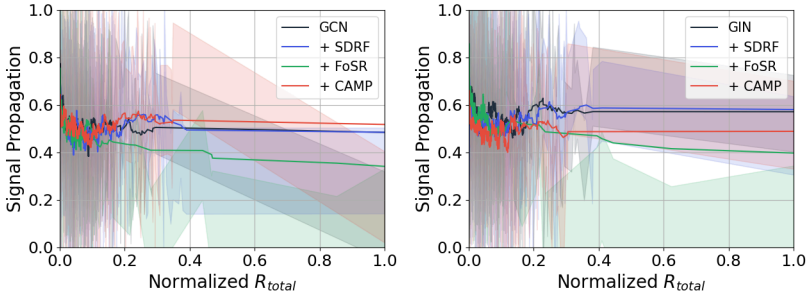


(a) COLLAB (b) MUTAG

Figure 5.4: A comparative study on the Dirichlet energies is presented. CAMP performs better than other rewiring methods.

Effect of Centrality Measure We studied the effect of various node centrality measures on the performance of CAMP. We employed 10-layered GCN and GIN as our backbones applied to six datasets. The results are demonstrated in Table 5.3. For instance, CAMP-GCN performed best on PROTEINS and REDDIT-BINARY when closeness centrality was employed. In IMDB-BINARY, Pagerank acted as the most suitable centrality measure to attain optimal performance. The variation in performance underscored the utility of employing a diverse array of centrality measures that dictate the performance of CAMP. The results for CAMP-GIN can be found in Section 5.6.1 of the Appendix.

Variation of Dirichlet Energy We applied 10-layered GCN and GIN paired with CAMP on COLLAB and MUTAG and estimated the Dirichlet energies for the final layers. Figure 5.4 suggests Dirichlet energy for CAMP is higher than familiar rewiring methods like FoSR and SDRF. The node features remained distinguishable in deeper layers confirmed by the higher values of energies. The results clearly state that CAMP improves the model performance in the multi-layered GNNs without pursuing rewiring. Thus, we can claim that CAMP is



(a) GCN on ENZYMES (b) GIN on ENZYMES

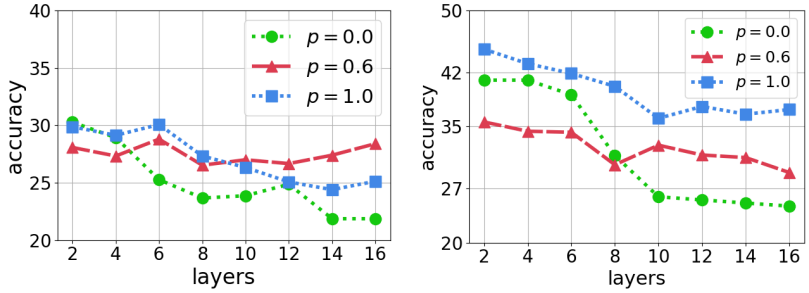
Figure 5.5: The signal propagation with respect to normalized R_{total} is presented. CAMP maintains steady trends compared to others, with increasing resistance.

capable of mitigating oversquashing and oversmoothing in multi-layered GNNs. More results are available in Section 5.6.2 of the Appendix.

Signal Propagation The total effective resistances (R_{total}) (Ghosh et al., 2008) of the graph create hindrances to propagating information. A graph with higher effective resistance is highly likely to be grappled with oversquashing (Black et al., 2023a). The efficacy of CAMP is measured with the signal propagation against the normalized R_{total} as adopted from (Di Giovanni et al., 2023b). Refer to Figure 5.5 which illustrates that CAMP performs better compared to other rewiring methods by effectively propagating signal across the network with increasing effective resistance. More results are available in the Section 5.6.4 of Appendix.

Effect on Oversmoothing We studied the effect of oversmoothing of multi-layered GCN and GIN on ENZYMES with various sampling rate of node batches. Figure 5.6 delineates the performance trends in the various network depths with varying p. CAMP demonstrated resilience to oversmoothing for the higher values of p. The performance strictly degrades when the CAMP is not applied, asserting the prowess of the proposed framework in mitigating oversmoothing in deeper GNNs. The higher number of layers increases the number of node batches which also enhances the number of time-stamps to access fixed channel capacity for processing aggregated features. Thus, CAMP mostly showed optimal performance when the network depth was higher. The results for the remaining datasets are provided in Section 5.6.3 of the Appendix.

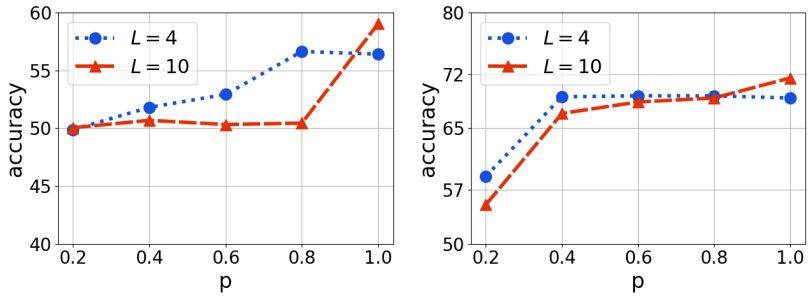
Variation of Batch Size We attempted to study the impact of various node batch sampling rates p on the performance of GCN and GIN. The experiment is conducted on IMDB-



(a) GCN on ENZYMES (b) GIN on ENZYMES

Figure 5.6: The performance of CAMP in deeper layers is presented. The performance improved when the node batch sampling rate was increased.

BINARY, and classification accuracy is presented in Figure 5.7 for two different model depths. The trends illustrate that an increase in the sampling rate leads to an uptrend in performance. This is due to the larger number of nodes participating in the message-passing operation and aggregating more relevant information with the increasing p. The plots for other datasets are provided in Section 5.6.5 of the Appendix.



(a) GCN on IMDB-BINARY (b) GIN on IMDB-BINARY

Figure 5.7: The performance for various node batch sampling rates for two different numbers of model layers are presented. The performance improves when the sampling rate is higher.

5.4 Conclusion

We designed the generalized asynchronous message passing framework RAMP, which randomly selects node batches at every layer to update corresponding features. However, we identified the limitations of RAMP and proposed CAMP, which creates layer-specific node batches based on node centrality. Our proposed framework can be flexibly paired with a mul-

titude of MP-GNNs. Furthermore, we also offer theoretical guarantees that CAMP is superior to RAMP in mitigating oversquashing. We applied CAMP to six standard graph datasets to validate the effectiveness of the framework. The experimental results indicate that CAMP demonstrated impressive performance, underscoring the prowess of asynchronous message passing over the structural rewiring. Exploring and designing asynchronous message-passing algorithms for dynamic graphs to alleviate oversquashing and oversmoothing can be potential future research directions.

5.5 Appendix A

5.5.1 Experimental Setup

We executed experiments on the base GNN models coupled with CAMP on the 25 random splits. For each split, the trained model is applied to the test set. The final test accuracy is presented with mean and standard deviation over all splits. The standard deviations are further scaled by the formula $s.d. \times \frac{2}{\sqrt{T}}$ where T is the total number of random trails (here $T = 25$), which is also followed by (Karhadkar et al., 2022). Among the datasets, COLLAB, IMDB-BINARY, and REDDIT-BINARY do not have node features. As per the standard policy, we assigned ones as the features to every node.

Table 5.5: The hyperparameters for each dataset are provided to reproduce the best results. L denotes the number of message-passing layers of the underlying GNN

Model	ENZYMES		MUTAG		PROTEINS		COLLAB		IMDB-BINARY		REDDIT-BINARY	
	L	Centrality	L	Centrality	L	Centrality	L	Centrality	L	Centrality	L	Centrality
GCN	10	Closeness	16	Degree	16	Degree	10	Load	12	Pagerank	16	Closeness
GIN	10	Closeness	12	Degree	16	Closeness	12	Betweenness	10	Pagerank	12	Closeness

5.5.2 Hyperparameter Details

The models are trained with a learning rate of 0.001 and a dropout rate of 0.50. Model parameters are optimized with Adam, and weight decay is fixed at 10^{-5}. In each experiment, we employed both 4-layered GCN and GIN models with a batch size of 64. These are the fixed hyperparameters for the entire experiment. The other data-specific hyperparameters are provided in Table 5.5.

5.5.3 Estimation of Dirichlet Energy

Let us consider graph G with adjacency matrix A and symmetrically normalized graph Laplacian $\tilde{L} = \mathrm{I} - \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, then the Dirichlet energy (Chung, 1997) of the set of features X with Laplacian is represented as following,

$$\begin{aligned} \mathrm{DE}(X, \tilde{L}) &= \mathrm{Tr}(X^\top \tilde{L} X) \\ &= \frac{1}{2} \sum_{(i,j)\in\mathcal{E}} A_{ij} \left(\frac{X_i}{\sqrt{d_i}} - \frac{X_j}{\sqrt{d_j}} \right)^2 \end{aligned} \tag{5.19}$$

The Dirichlet energy is directly connected with the measure of oversmoothing in the GNNs. The lower value of DE indicates the node features became indistinguishable when neighborhood aggregation was performed in the deeper layers. In our experiments, we utilized the Eq **??** to compute the energies at the final layer of the architectures.

5.5.4 Signal Propagation and Total Effective Resistance

We will present the estimation of signal propagation across the graph with respect to the normalized total effective resistance as discussed in the Experiments. For any input graph, first, consider a source node v and assign a p-dimensional feature vector with zero vectors assigned to the rest of the nodes. Initialize a model randomly with m layers, then the amount of signal propagated can be estimated as,

$$h_{\odot}^{(m)} = \frac{1}{p \max_{u\neq v} d_{\mathcal{G}}(u,v)} \sum_{j=1}^{p} \sum_{u\neq v} \frac{h^{(m),j}}{\|h^{(m),j}\|} d_{\mathcal{G}}(u,v) \tag{5.20}$$

where $h^{(m),j}$ denotes the j^{th} component of the feature at the m^{th} layer, $d_{\mathcal{G}}(u,v)$ is the shorted distance between the node u and v. The normalized signal $\frac{h^{(m),j}}{\|h^{(m),j}\|}$ weighted by $d_{\mathcal{G}}(u,v)/\max_{u\neq v} d_{\mathcal{G}}(u,v)$ is propagated from source node v to all other nodes. The final signal is averaged over the number of feature dimensions p. In our context, for each graph, we selected 10 source nodes and the final signal $h_{\odot}^{(m)}$ with total effective resistance averaged over each source node. This process is continued over all graphs, and total effective resistance and signal propagation are plotted.

5.6 Appendix B

5.6.1 Effect on Centrality on Performance of CAMP-GIN

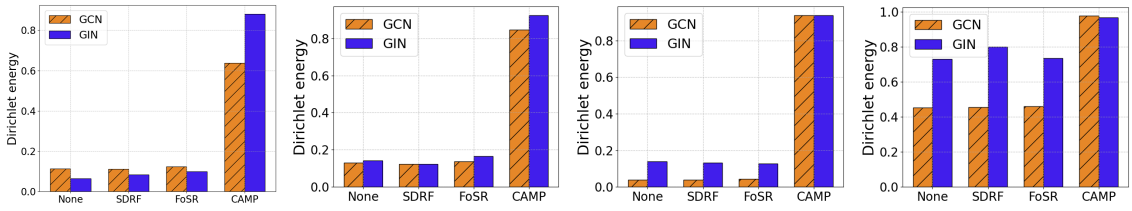
We presented the comparative study of the performance of CAMP-GIN in Table 5.6. The experiment is conducted on a 10-layered GIN architecture. The optimal performances for each dataset are attained for different centrality measures, highlighting the importance of centrality choices.

Table 5.6: A comparative study of the performance of CAMP-GIN is presented based on five different centrality measures.

Centrality	ENZYMES	MUTAG	PROTEINS	COLLAB	IMDB-BINARY	REDDIT-BINARY
Degree	43.13 ± 2.88	$\mathbf{80.00 \pm 4.36}$	72.39 ± 1.89	72.34 ± 0.77	70.64 ± 1.68	75.38 ± 1.13
Betweenness	41.87 ± 2.62	77.00 ± 4.16	72.07 ± 2.02	$\mathbf{73.30 \pm 0.86}$	69.76 ± 2.15	72.70 ± 0.90
PageRank	41.00 ± 2.54	74.00 ± 4.08	72.68 ± 1.55	72.66 ± 0.79	$\mathbf{71.48 \pm 1.47}$	77.98 ± 1.37
Load	39.93 ± 2.44	76.00 ± 4.28	72.64 ± 2.26	72.97 ± 0.85	69.84 ± 1.94	72.82 ± 0.78
Closeness	$\mathbf{46.60 \pm 2.26}$	76.20 ± 2.96	$\mathbf{74.07 \pm 1.73}$	72.57 ± 0.61	71.00 ± 1.99	$\mathbf{81.98 \pm 1.20}$

5.6.2 Effect on Dirichlet Energy

The additional results are presented in Figure 5.8, where CAMP produces higher Dirichlet energy compared to other rewiring methods. Thus, CAMP showed the capacity to control oversmoothing in GNNs.

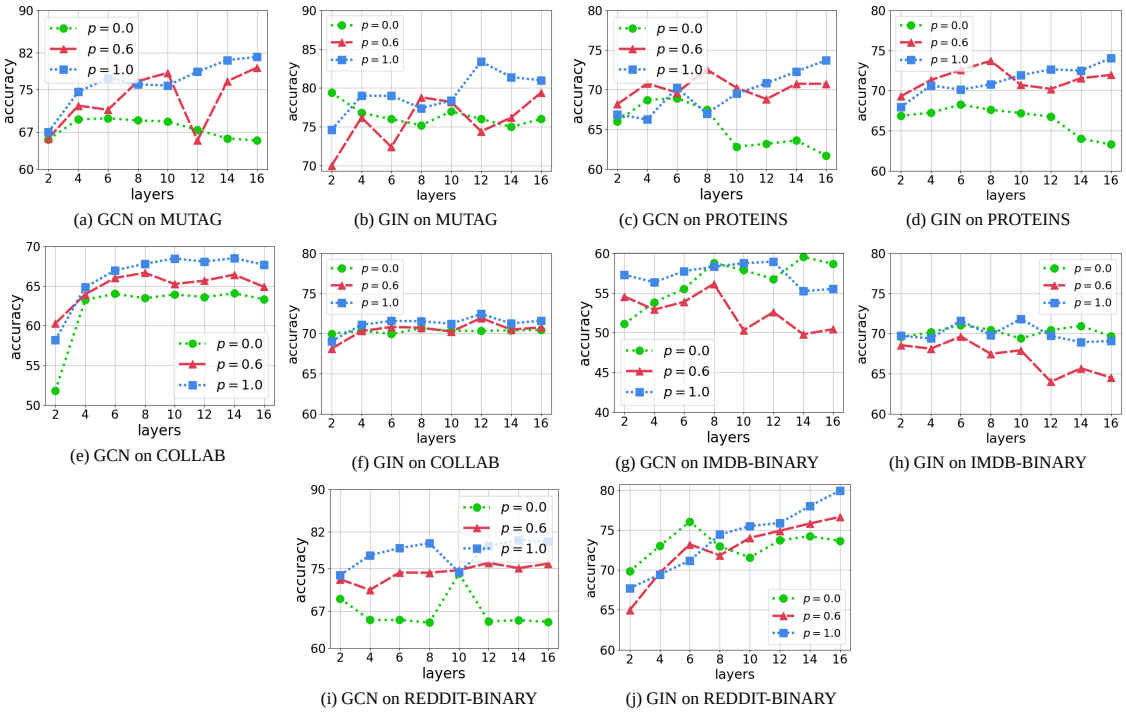


(a) ENZYMES (b) PROTEINS (c) IMDB-BINARY (d) REDDIT-BINARY

Figure 5.8: The Dirichlet energies of the last layer of the models are presented. CAMP attains better energy values than other rewiring methods.

5.6.3 Effect on Oversmoothing

The additional plots for the performance of CAMP in deeper layers are presented in Figure 5.9. The performance trends suggest the power of CAMP in tackling oversmoothing in the deep GNNs.



(a) GCN on MUTAG (b) GIN on MUTAG (c) GCN on PROTEINS (d) GIN on PROTEINS

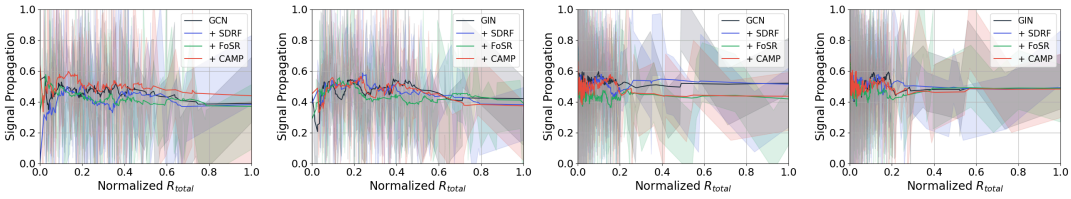
(e) GCN on COLLAB (f) GIN on COLLAB (g) GCN on IMDB-BINARY (h) GIN on IMDB-BINARY

(i) GCN on REDDIT-BINARY (j) GIN on REDDIT-BINARY

Figure 5.9: Performance of CAMP with different batch sampling rates are presented for various network depths. The test accuracy is observed when the full subset is selected as the node batches.

5.6.4 Signal Propagation

The additional results pertaining to the efficiency of CAMP in propagating signal are presented in Figure 5.10.



(a) GCN on MUTAG (b) GIN on MUTAG (c) GCN on PROTEINS (d) GIN on PROTEINS

Figure 5.10: The signal propagation with respect to normalized R_{out} is presented for MUTAG and PROTEINS. CAMP exhibited competitive performance compared to other rewiring methods by propagating a steady signal across the network with increasing resistance.

5.6.5 Effect on Variation of Batch

The additional results of the performance of GCN and GIN paired with CAMP are presented over various sampling probabilities. The experiments are performed for two different layers 4 and 10. The plots are demonstrated in Figure 5.11. CAMP achieved best performances in most cases when the sampling probability is 1, indicating the full subset is chosen at a particular layer.

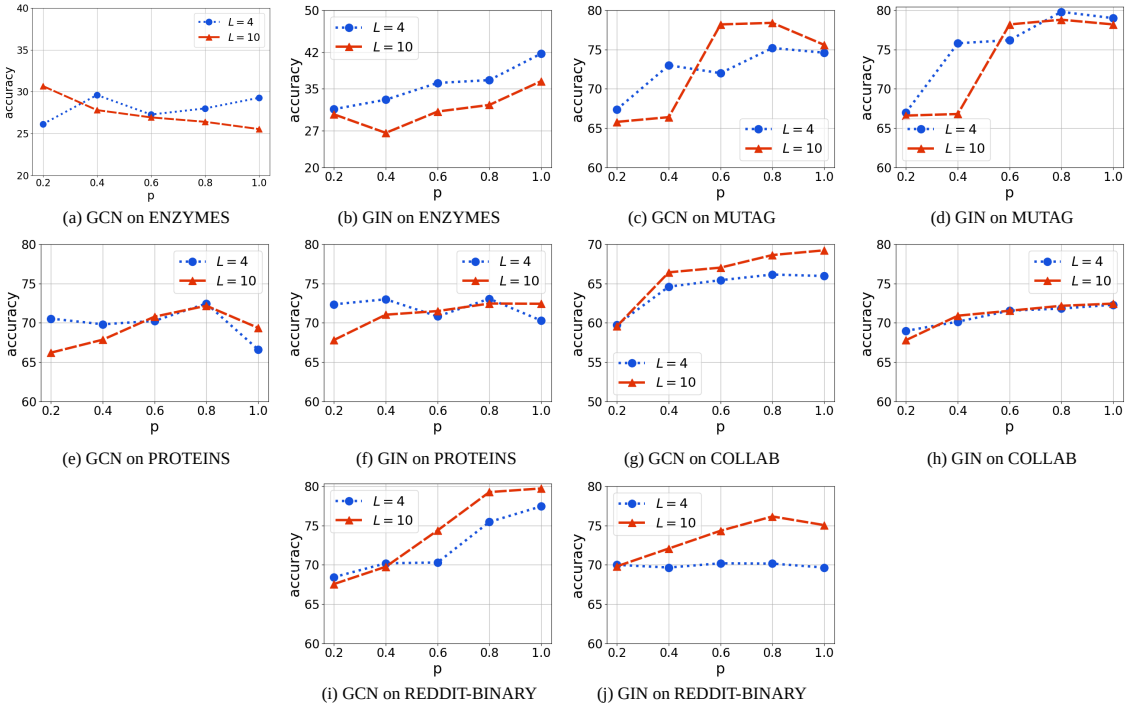


Figure 5.11: The performance variation is illustrated with various node batch sampling rates. The best results are mostly obtained when the complete subset is chosen as a node batch for the hidden layers.

Chapter 6

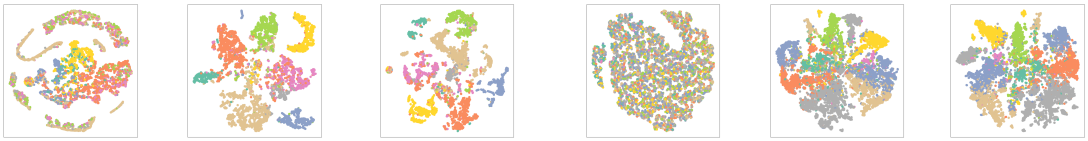
Designing Hyperbolic Positional Encodings for Enhancing Graph Transformer

Summary

Graph Transformers (GTs) facilitate the comprehension of complex relationships on graph-structured data by leveraging self-attention of the possible pairs of nodes. The structural information or inductive bias of the input graph is provided as positional encodings into the GT. The positional encodings are mostly Euclidean and are not able to capture the complex hierarchical relationships of the corresponding nodes. To address the limitation, we introduce a novel and efficient framework, HyPE, that generates learnable positional encodings in the non-Euclidean hyperbolic space to capture the geometrical distribution of the data better. Unlike existing methods, HyPE can generate a set of hyperbolic positional encodings, empowering us to explore diverse options for the optimal selection of PEs for specific downstream tasks. Additionally, we repurpose the generated hyperbolic positional encodings to mitigate the impact of oversmoothing in deep Graph Neural Networks (GNNs). Furthermore, we provide extensive theoretical underpinnings to offer insights into the working mechanism of the HyPE framework. Comprehensive experiments on four molecular benchmarks, including the four large-scale Open Graph Benchmark (OGB) datasets, substantiate the effectiveness of hyperbolic positional encodings in enhancing the performance of Graph Transformers. We also consider Coauthor and Copurchase networks to establish the efficacy of HyPE in controlling oversmoothing in deep GNNs.

6.1 Introduction

Graph Transformers (GTs) are earmarked as one of the milestones for modeling the interactions between the node pairs in the graph. As the existing graph neural network suffers from a few glaring shortcomings like oversmoothing (Li et al., 2018), which occurs due to the recursive neighborhood aggregation, oversquashing (Alon and Yahav, 2020), an information bottleneck caused by the exponential growth of information while increasing the size of the receptive field, and bounded expressive power (Xu et al., 2018a; Morris et al., 2019). Graph Transformers confront the limitations by assuming the entire graph as complete and estimating self-attention for all possible node pairs. Yet, such an approach alleviates the limitations of GTs, but still loses the structural inductive bias, which causes the loss of positional information of the nodes. As an effective solution, the positional encodings as vectors are integrated with the node features to make the respective nodes topologically aware in the graph. Recently, many efforts were made to generate effective positional encodings for the GTs, like spectral decomposition-based learnable encoding (Kreuzer et al., 2021), structure-aware PEs generated from the rooted subgraph or subtree of the nodes (Chen et al., 2022a), encoding the structural dependency (Ying et al., 2021b), random walk-based learnable positional encodings (Dwivedi et al., 2021), and many more.



(a) (b) (c) (d) (e) (f)

Figure 6.1: The visualization of node embeddings of Amazon Photo and CoauthorCS generated by 128-layered GCN for the PE category of 4 and 3 respectively. (a) Node embeddings for Amazon Photo without using any positional encodings. (b) Node embeddings of Amazon Photo integrated with the HyPE. (c) Embeddings of hyperbolic PEs from HyPE generated for Amazon Photo. (d) Node embeddings of Coauthor CS without using PEs. (e) Node embeddings of Coauthor CS integrated with the HyPE. (f) Embeddings of hyperbolic PEs from HyPE generated for Coauthor CS.

The positional encodings derived from the existing works suffer from several critical limitations. For example, the Spectral Attention Network (SAN) (Kreuzer et al., 2021) generates learnable positional encodings by spectral decomposition of the Laplacian matrix. SAN

requires high computational time as well as memory, especially for the generation of edge-feature-based Laplacian positional encodings. Structure-aware Transformer (SAT) estimates pairwise attention scores depending on the respective rooted sub-graphs or sub-trees. SAT requires the extraction of multi-hop subgraphs, increasing the pre-processing time and consuming high memory. Even though the depth of the rooted subtree increases, the model may suffer from oversmoothing. Additionally, the proposed methods operate in Euclidean space, which restricts us from exploring the hierarchical relationships of the node pairs in designing positional encodings. This work aims to fill the void by proposing a novel framework named **Hy**perbolic **P**ositional **E**ncodings-based **G**raph **T**ransformer or **HyPE-GT**, which is capable of generating a set of learnable positional encodings in the hyperbolic space. To our knowledge, we are the first to foster hyperbolic geometry in designing positional encodings for the Graph Transformers.

HyPE-GT generates trainable hyperbolic positional encodings, consisting of three pivotal modules (1) the initialization of positional encodings (PE_{init}), (2) the type of hyperbolic manifold (M), where the entire operation will take place, and (3) the employed hyperbolic neural architectures (HNA) which transforming the encodings into the chosen hyperbolic space. Each module can take values from a relevant set of entities and each positional encoding is the result of the unique combination of the entities chosen from the pre-defined sets. The maximum number of positional encodings is given by $|PE_{\text{init}}| \times |M| \times |HNA|$, with the framework offering diverse design choices based on entity selection, generating a wide spectrum of PEs. These hyperbolic positional encodings provide practitioners with versatile options for solving downstream tasks. The learnable hyperbolic PEs can be seamlessly integrated with standard graph transformers, supplying essential positional information. Additionally, we combine positional encodings with the node features from hidden layers in deep Graph Neural Networks (GNNs) for the mitigation of oversmoothing.

Contribution Our contributions throughout the paper can be summarized in the following way,

- We propose a novel and efficient framework named HyPE-GT that generates a set of learnable positional encodings in the hyperbolic space, a non-euclidean domain. HyPE-GT is tailored to encode the hierarchical structural patterns into the generated po-

sitional encodings and offers better information than Euclidean counterparts. To the best of our knowledge, we are the first to incorporate hyperbolic PEs with the Graph Transformer. The PEs are learned by passing through either hyperbolic neural networks or hyperbolic graph convolutional networks, which produce useful positional and structural encodings.

- The hyperbolic positional encodings are re-purposed to diminish the effect of oversmoothing in deep GNN models. The PEs are incorporated with the transformed features of the layers of GNNs.

- We present detailed theoretical analyses offering clarity on the effectiveness of HyPE-GT on the graph datasets and mitigation of oversmoothing in deeper GNNs. The detailed proofs and derivations are discussed in Section 6.9.1 of the Appendix 6.9.

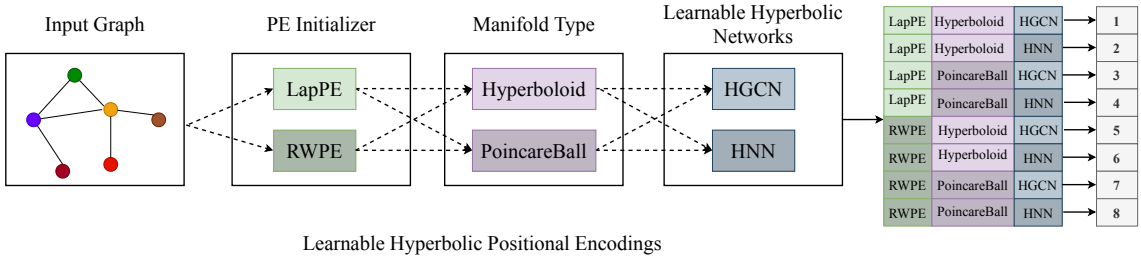


Figure 6.2: Schematic representation of the process for generating a family of learnable hyperbolic positional encodings (8 different categories) in the HyPE-GT framework. Each category can be generated by following a particular path (shown with arrow-marked dotted lines), which begins from the PE initialization block and ends at the learnable hyperbolic networks block. Each positional encoding is assigned a unique number, shown on the right side of the diagram.

6.2 Background

Graph Transformers The attention mechanism on the graph data is primarily introduced with Graph Attention Network (GAT) (Veličković et al., 2017). However, GAT can only learn from the connected neighborhoods via sparse message passing. The limitation was overcome when Dwivedi and Bresson generalized the Transformer architecture for the graphs (Dwivedi and Bresson, 2020) and showed its utility on various categories of tasks. Later, many other

significant approaches were taken, like Spectral Attention Network (SAN) (Kreuzer et al., 2021), which relies on learning the eigenvectors of the Laplacian matrix. The learnable eigenvectors act as PE, which is concatenated with the original node features for the Transformer. Structure-aware Transformer (SAT) (Chen et al., 2022a) computes the self-attention among node pairs by incorporating structural information of the extracted subgraphs rooted at each node. Another line of work, GraphiT (Mialon et al., 2021), learns relative positional encoding via diffusion kernels, which computes attention between the node pairs. On the other hand, Graphormer (Ying et al., 2021b) designs spatial representations like degree encoding, edge encoding, etc., which capture the structural dependency of the graph. The encodings are added as the bias terms in the self-attention matrix. GraphTrans (Wu et al., 2021) proposes a model that first learns from multiple GNN layers stacked together, and then the updated node representations are provided as input to the standard Graph Transformer layer. Recently, another framework GraphGPS (Rampášek et al., 2022) allows the integration of message-passing models with the module that computes global attention, resulting in an effective and scalable architecture. TokenGT (Kim et al., 2022) considers every node and edge of the graph as independent tokens. The tokens are associated with token embeddings that are the input to the standard Transformer. Edge-augmented Graph Transformer(EGT) (Hussain et al., 2022) introduces edge embeddings for every pair of nodes, which act as the edge gates to control information flow in the Transformer. Graph Transformer Networks (GTN) (Yun et al., 2019) and Heterogeneous Graph Transformer (HGT) (Hu et al., 2020b) are dedicated to heterogeneous graphs, which extract effective meta-paths based on attention.

Hyperbolic Graph Neural Networks Recently, efforts were made to make deep neural networks suitable in the non-Euclidean space like hyperbolic space equipped with negative curvature (Ganea et al., 2018; Khrulkov et al., 2020). Subsequently, the hyperbolic neural networks were extended for the graph-structured data as Hyperbolic Graph Neural Networks (Liu et al., 2019a). On advancement, the hyperbolic graph convolutional networks (Chami et al., 2019) and hyperbolic graph attention networks (Zhang et al., 2021) strengthen the family of the hyperbolic graph neural networks.

6.3 Proposed Method

6.3.1 Preliminaries and Notations

Assume an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$, where $\mathcal{V}$ denotes a set of vertices, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes set of edges. $X \in \mathbb{R}^{n \times d}$ presents the node attributes where n denotes the number of nodes, and each node is associated with a d-dimensional feature. The adjacency matrix A is the symmetric matrix with binary elements denoting the edges or node connections in the graph. D is a diagonal matrix where each element in the diagonal is the degree of the corresponding nodes. Consider $||X||_F$ as the matrix Frobenius norm of X.

6.3.2 Transformers on Graphs

Inspired by Vashwani *et al.* (Vaswani et al., 2017), Dwivedi, and Bresson (Dwivedi and Bresson, 2020) extended the philosophy of Transformer for the graph-structured data. Message-passing Graph Neural Networks (MP-GNNs) implement sparse message passing where GTs assume the fully connected graph structure. Unlike MP-GNNs, Graph Transformers are designed to compute the attention coefficient between pairs of nodes without considering the graph structure. A single Transformer layer consists of a self-attention module followed by a feed-forward network. The feature matrix X is transformed into query Q, key K, and values V by multiplying with the projection matrices as $Q = XW_Q$, $K = XW_K$, and $V = XW_V$. The self-attention matrix is computed as follows,

$$X_A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_{out}}}\right)V, \tag{6.1}$$

where $X_A \in \mathbb{R}^{n \times d_{out}}$ is the self-attention matrix with d_{out} is the output dimension. W_Q, W_K, and W_V are trainable parameters. To boost the impact of the self-attention module, we often employ a multi-head attention (MHA) strategy. The multi-head attention is the result of the concatenation of multiple instances of the Eqn. 6.1. The multi-head attention can be expressed as:

$$X_A = M\Big\|_{k=1}^{H}\left(\sum_{j \in N(i)} \alpha_{ij}^{(k)} V^{(k)} X_j\right), \tag{6.2}$$

where $\alpha_{ij}^{(k)}$ is the attention coefficient between node i and j from k^{th} head. M and $V^{(k)}$ are the trainable parameters and X_j is the j^{th} node features. The estimated self-attention matrix is followed by a residual connection and then passes through a feed-forward network (FFN) with the normalization layers as follows,

$$\begin{aligned} X' &= \text{Norm}(X + X_A), \\ X'' &= W_2(\text{ReLU}(W_1 X')), \\ X_o &= \text{Norm}(X' + X''), \end{aligned} \tag{6.3}$$

where X_o is the final output of the transformer layer and $W_1 \in \mathbb{R}^{d_{out} \times d}$, $W_2 \in \mathbb{R}^{d \times d}$ are trainable parameters. We can either use Batchnorm (Ioffe and Szegedy, 2015) or Layernorm (Ba et al., 2016) for the feature normalization. Each Transformer layer generates node-level representations, which are permutation equivariant. The absence of positional information on the nodes generates similar outputs. Therefore, it is necessary to incorporate appropriate positional encodings to leverage the learning process in the Transformer.

6.3.3 Preliminaries on Hyperbolic Spaces

Hyperbolic geometry deals with the smooth manifold with constant negative curvature. Let us consider the manifold $\mathcal{M} \in \mathbb{R}^d$ embedded in $\mathbb{R}^{d+1}$ with constant curvature c. Let us have the following definitions.

Tangent space, Logarithmic map, and Exponential map For every point $x \in \mathcal{M}$, the tangent space $\mathcal{T}_x\mathcal{M}$ is defined as a d-dimensional hyperplane approximates the $\mathcal{M}$ around x. Define exponential map $\exp_x^c$ as $\exp_x^c : \mathcal{T}_x\mathcal{M} \to \mathcal{M}$ for any point x, which transforms any vector in $\mathcal{T}_x\mathcal{M}$ on the $\mathcal{M}$. The logarithmic map $\log_x^c$ is the inverse of the exponential map that is $\log_x : \mathcal{M} \to \mathcal{T}_x\mathcal{M}$, transforms any point on the manifold to the tangent space. The Riemannian metric $g^{\mathcal{M}}$ on the manifold is defined as the inner product on the tangent space like $g^{\mathcal{M}}(.,.) : \mathcal{T}_x\mathcal{M} \times \mathcal{T}_x\mathcal{M} \to \mathbb{R}$. Along with these definitions, in what follows, we will discuss two well-adopted models from hyperbolic geometry.

Hyperboloid Model One of the models in the hyperbolic geometry is also commonly known as the Lorentz model. The model is defined as Riemannian manifold $(\mathbb{H}_c^n, g_x^{\mathbb{H}})$ with negative

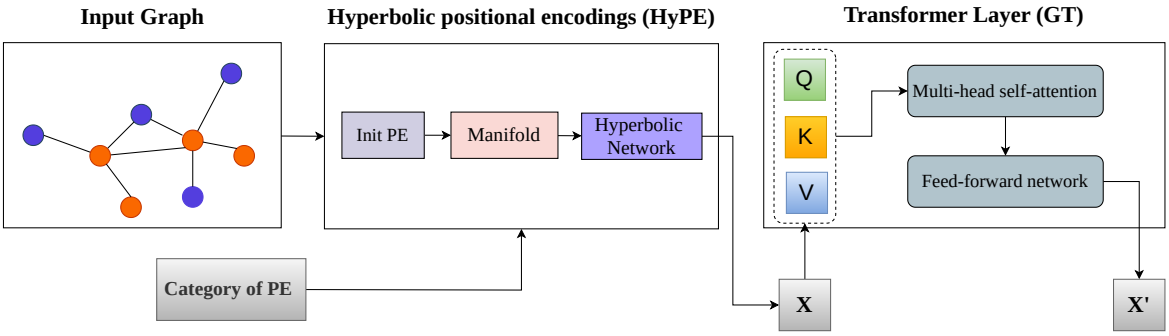


Figure 6.3: The workflow of HyPE-GT is presented. The framework combines two independent blocks: hyperbolic positional encodings or HyPE and standard Graph Transformer or GT. Depending on one's choice, HyPE will generate a fixed type of PE. The PE is added with the feature matrix X before fetching it to the Transformer layer.

curvature as follows:

$$\mathbb{H}_c^n = \{x \in \mathbb{R}^{n+1} : < x, x >_{\mathbb{L}} = -c, x_0 > 0\} \tag{6.4}$$

$$g_x^{\mathbb{H}} = diag([-1, 1, \cdots, 1])_n, \tag{6.5}$$

where $\langle ., . \rangle_{\mathcal{L}}$ denotes Minskowski inner product. The distance between two points $x, y \in \mathbb{H}_c^n$ is defined as

$$d_{\mathcal{L}}^c(x, y) = \sqrt{c} \cosh^{-1}(-\frac{\langle x, y \rangle_{\mathcal{L}}}{c}). \tag{6.6}$$

For any $v \in \mathbb{H}_c^n$, the exponential and logarithmic maps for the hyperboloid model are as follows:

$$\begin{aligned} \exp_x^c(v) &= \cosh\left(\frac{||v||_{\mathcal{L}}}{\sqrt{c}}\right) x + \sqrt{c} \sinh\left(\frac{||v||_{\mathcal{L}}}{\sqrt{c}}\right) \frac{v}{||v||_{\mathcal{L}}}, \\ \log_x^c(y) &= d_{\mathcal{L}}^c(x, y) \frac{y + \frac{1}{k}\langle x, y \rangle_{\mathcal{L}} x}{||y + \frac{1}{k}\langle x, y \rangle_{\mathcal{L}} x||_{\mathcal{L}}}. \end{aligned} \tag{6.7}$$

Poincarê Ball Model Another prominent model that is equipped with negative curvature $c(c < 0)$. The model is defined as the Riemannian manifold $(\mathbb{B}_c^n, g_x^{\mathbb{B}})$ as follows:

$$\begin{aligned} \mathbb{B}_c^n &= \{x \in \mathbb{R}^n : ||x||^2 < -\frac{1}{c}\}, \\ g_x^{\mathbb{B}} &= \frac{2}{(1 + c||x||_2^2)} g^E, \end{aligned} \tag{6.8}$$

where g^E is the Euclidean metric which is I_n. The model represents an open ball of radius of $\frac{1}{\sqrt{c}}$. If two points $x, y \in \mathbb{B}_c^n$, then the distance between them is

$$d_{\mathbb{B}}(x,y) = \cosh^{-1}\left(1 + 2\frac{||x-y||^2}{(1-||x||^2)(1-||y||^2)}\right). \tag{6.9}$$

For any $v \in \mathbb{B}_c^n$, the exponential and logarithmic maps are defined in the following way:

$$\begin{aligned} \exp_x^c(v) &= x \oplus_c \left(\tanh\left(\sqrt{c}\frac{\lambda_x^c||v||}{2}\right)\frac{v}{\sqrt{c}||v||}\right), \\ \log_x^c(y) &= \frac{2\lambda_x^c}{\sqrt{c}}\tanh^{-1}(\sqrt{c}||-x\oplus y||)\frac{-x\oplus y}{||-x\oplus y||}, \end{aligned} \tag{6.10}$$

where $\lambda_x^c = \frac{2}{1-c||x||^2}$.

6.3.4 Learnable Hyperbolic Positional Encodings: An Overview

This section presents a brief overview of the HyPE-GT framework for easy apprehension regarding the rest of the work. HyPE-GT consists of three key modules, namely, (1) the initialization of the positional encodings, (2) the type of the manifolds where the transformed PEs will be projected, and the last one is (3) learning the hyperbolic positional encodings by employing either Hyperbolic Neural Network (HNN) or Hyperbolic Graph Convolutional Network (HGCN). As we previously mentioned, the characteristics of the generated positional encodings are nothing but the manifestation of the choice of entities in those three modules. This work considers two entities for each module: LapPE and RWPE for PE initialization, Hyperboloid, and PoincarêBall for manifold type. For learnable networks, we select HNN and HGCN. Refer to Figure 6.2 for more intricate details of the flow of the framework. Thus, there will be 8 different categories of PEs that can be generated, and each of them is assigned a unique number from $1-8$ for ease of reference. The rest of the paper is organized to provide more insights regarding the proposed framework, followed by extensive experimentation.

6.3.5 Initialization of Positional Encodings

The effectiveness of the learnable hyperbolic positional encodings is heavily reliant on the initialization of positional encodings. Despite several existing PEs (Mialon et al., 2021), (Zhou et al., 2020). (Li et al., 2020b), (Ying et al., 2021b) etc, we adopt two well-known PE

initialization techniques as proposed in (Dwivedi et al., 2021), which are Laplacian Positional Encodings (LapPE) and Random Walk Positional Encoding (RWPE). LapPE is generated by performing eigen-decomposition of the Laplacian matrix of the input graph. For the i^{th} node, the respective LapPE can be presented as:

$$p_i^{\text{LapPE}} = [U_{i1}, U_{i2}, \cdots, U_{ik}] \in \mathbb{R}^k, \tag{6.11}$$

where U_i denotes the i^{th} eigenvectors of the graph Laplacian and first k elements are chosen resulting in the k-dimensional encoding. The p_i^{LapPE} eigenvector is designated as the initialized position vector for the i^{th} node in the graph. The eigenvectors are assumed to have information on the spectral properties of the input graph. The latter one is the k-dimensional RWPE, generated by k-steps of the diffusion process with the degree-normalized random walk matrix. For the i^{th} node, RWPE can be represented as follows:

$$p_i^{\text{RWPE}} = [\hat{A}_{ii}, \hat{A}_{ii}^2, \cdots, \hat{A}_{ii}^k] \in \mathbb{R}^k, \tag{6.12}$$

where $\hat{A} = AD^{-1}$. RWPE captures the structural patterns of the k-hop neighborhoods. Finally, the initialization of the PE module has two positional encodings.

6.3.6 Choice of Manifold

The whole mechanism of generating the positional encodings heavily relies on the nature of the manifolds where the initialized positional encodings will be projected. Hence, the choice of a manifold will be one of the critical decisive factors for generating effective PEs as well as the performance enhancement of the Graph Transformers. In HyPE-GT, we will involve two dominant choices named `Hyperboloid` and `PoincarêBall` manifolds to serve the purpose.

6.3.7 Learning through Hyperbolic Neural Architectures

The initialized positional encodings are learned by applying hyperbolic neural network architectures, such as hyperbolic neural networks and hyperbolic graph convolutional networks. Let the initialized positional encodings p^{init} is transformed to $\hat{p} = W_0 p^{\text{init}}$ into some low dimensional space via the parameterized transformation W_0. Then $\hat{p}$ is learnable in the

following ways:

Hyperbolic Neural Network (HNN) The transformed initial positional encodings are fed into a hyperbolic multi-layered feed-forward neural network. The HNN produces encodings into the hyperbolic space by layer-wise propagation. The transformation can be formulated as follows,

$$p^{\text{HNN}} = \text{HNN}_{\mathcal{M}}^{(L)}(\hat{p} \,|\, \Theta), \tag{6.13}$$

where Θ is the trainable parameters of the HNN, L denotes the number of hidden layers, and $\mathcal{M}$ denotes the pre-defined manifold type, which can be either Hyperboloid or PoincarêBall. the learnable positional encodings p^{HNN} are projected into this manifold. Notably, HNNs only consider the node features as input without considering graph adjacency. Thus, HNNs cannot learn structure-aware embeddings from the input graph. The limitation can be alleviated by using a graph convolutional-based architecture.

Hyperbolic Graph Convolutional Network (HGCN) We fed the positional encodings to the hyperbolic graph convolutional networks to extract necessary information from the graph structure. Like the previous one, firstly, the initial positional features are mapped into low-dimensional space, and then positional encodings are learned by the hyperbolic graph convolutional network. The transformation of HGCN can be described as follows:

$$p^{\text{HGCN}} = \text{HGCN}_{\mathcal{M}}^{(L)}(\hat{p} \,|\, \Phi), \tag{6.14}$$

where Φ denotes the trainable parameters of the HGCN, L is the number of stacked convolutional layers, and $\mathcal{M}$ can be either Hyperboloid or PoincarêBall model. Note that HGCN takes input from both node features and adjacency and successfully overcomes the shortcomings faced by HNN. The learned PEs will be integrated with the Transformer architecture to provide positional information on the nodes.

6.3.8 Complete Pipeline of HyPE-GT Framework

HyPE-GT comprises two key modules, the first one is HyPE, which generates learnable hyperbolic positional encodings and the second one is the standard Graph Transformer (GT). HyPE is designed to produce various categories of positional encodings for a single down-

Table 6.1: Performance on three benchmark datasets (Dwivedi et al., 2020). For each category of PE, the results are presented with mean and standard deviations from 10 runs with different random seeds. The category of PE and the optimal number of layers are mentioned, respectively, inside the parentheses. The top three results **first**, second, and third are marked.

Method	PATTERN	CLUSTER	MNIST	CIFAR10
	Accuracy ↑	Accuracy ↑	Accuracy ↑	Accuracy ↑
GCN Kipf and Welling (2016)	71.892 ± 0.334	68.498 ± 0.976	90.705 ± 0.218	55.710 ± 0.381
GIN Xu et al. (2018a)	85.387 ± 0.136	64.716 ± 1.553	96.485 ± 0.252	55.255 ± 1.527
GAT Veličković et al. (2017)	78.271 ± 0.186	70.587 ± 0.447	95.535 ± 0.205	64.223 ± 0.455
Gated-GCN Bresson and Laurent (2017)	85.568 ± 0.088	73.840 ± 0.326	97.340 ± 0.143	67.312 ± 0.311
PNA Corso et al. (2020)	–	–	97.94 ± 0.12	70.350 ± 0.630
DGN Zhou et al. (2020)	86.680 ± 0.034	–	–	72.838 ± 0.417
GraphTransformer + LapPE Dwivedi and Bresson (2020)	84.718 ± 0.068	73.169 ± 0.622	–	–
Graphormer Ying et al. (2021b)	–	–	–	–
k-subgraph SAT Chen et al. (2022a)	**86.848 ± 0.037**	77.856 ± 0.104	–	–
SAN Kreuzer et al. (2021)	86.581 ± 0.037	76.691 ± 0.65		
EGT Hussain et al. (2022)	**86.821 ± 0.020**	**79.232 ± 0.348**	98.173 ± 0.087	68.702 ± 0.409
GraphGPS Rampášek et al. (2022)	86.685 ± 0.059	78.016 ± 0.180	98.051 ± 0.126	72.298 ± 0.356
Exphormer Shirzad et al. (2023)	86.734 ± 0.008	–	98.414 ± 0.056	**74.754 ± 0.194**
GRED Ding et al. (2024)	86.759 ± 0.020	**78.495 ± 0.103**	98.383 ± 0.012	**76.853 ± 0.165**
TIGT Choi et al. (2024b)	86.681 ± 0.062	78.025 ± 0.223	98.231 ± 0.132	73.963 ± 0.364
Graph-Mamba Wang et al. (2024)	86.710 ± 0.050	76.800 ± 0.360	**98.420 ± 0.080**	73.700 ± 0.340
HyPE-GT (ours)	**86.779 ± 0.038**(1, 1)	**78.228 ± 0.126**(1, 1)	**98.510 ± 0.007**(8, 2)	74.680 ± 0.009(7, 2)
HyPE-GTv2 (ours)	86.756 ± 0.141	78.139 ± 0.054	**98.617 ± 0.009**	**74.916 ± 0.042**

stream task. The category of PE is decided by choosing the three components: (1) initialization of positional encodings, (2) the type of the manifold, and (3) the hyperbolic networks. Figure 6.3 illustrates the complete workflow of generating hyperbolic positional encodings of a given input graph and the integration with the Transformer architecture. For an external input for the category number, we have a pre-defined triplet that produces a fixed category of positional encodings enumerated as $\{1, 2, \cdots, 8\}$. For example, if someone wants to generate the category 4, then the triplet should be like $\{\texttt{LapPE}, \texttt{PoincarêBall}, \texttt{HGCN}\}$. This way, HyPE-GT can produce 8 in diverse categories of positional encodings. However, the generated positional encodings lie in the hyperbolic space, and node features belong to the Euclidean domain. Therefore, the integration of the two should not be straightforward. We devised two different strategies for the addition of hyperbolic PEs with the node features in the following way:

Incorporating Hyperbolic PEs with GT The generated PEs from the HyPE pipeline are embedded in the hyperbolic space. Therefore, performing the direct addition between Euclidean node features and hyperbolic positional encodings is not supported. The issue is resolved by transforming the node features into hyperbolic space and is incorporated by performing Möbius addition (Ganea et al., 2018) with the hyperbolic positional encodings. The final output is embedded in the hyperbolic space. If $p_k^{\mathbb{H}}$ is the generated positional

encodings in the hyperbolic space with manifold type $\mathbb{H}$ for the k^{th} category and $\hat{x}_v^{\mathbb{E}}$ is the initial node feature in Euclidean space for the v^{th} node, then addition of PEs with node features will be as following:

$$\begin{aligned} \hat{x}_v^{\mathbb{H}} &= \exp_0(\tan_{\text{proj}}(\hat{x}_v^{\mathbb{E}})), \\ x_v^{\mathbb{H}} &= \hat{x}_v^{\mathbb{H}} \oplus_c p_k^{\mathbb{H}}, \end{aligned} \tag{6.15}$$

where $\hat{x}_v^{\mathbb{H}}$ and $x_v^{\mathbb{H}}$ are the node features in hyperbolic space before and after addition of the hyperbolic positional encodings respectively, $\oplus_c$ denotes the Möbius addition with space curvature c. The function $\tan_{\text{proj}}$ maps the features from the Euclidean domain to the tangent space of the hyperbolic space at the point 0 of the manifold. The updated node features are subsequently reverted into the Euclidean space by applying the log map to provide input to the Transformer.

$$x_v^{\mathbb{E}} = \log_0(x_v^{\mathbb{H}}), \tag{6.16}$$

where $x_v^{\mathbb{E}}$ is the updated node features in the Euclidean space.

Utility in Deep GNNs Oversmoothing (Li et al., 2018) is posed as a dominant challenge of deep GNNs. The node features become indistinguishable due to the recursive neighborhood aggregation during the message propagation in deeper layers. Hence, we re-purpose HyPE-generated hyperbolic positional encodings to couple with the intermediate node features to mitigate the issue. The following operations represent the usage of positional encodings in the output of multi-layered GNNs:

$$\begin{aligned} \hat{x}_v^E &= \text{GNN}_L(\hat{x}_v \,|\, \theta), \\ \hat{x}_v^H &= \exp\big(\tan_{\text{proj}}(\hat{x}_v^E)\big), \\ x_v^H &= \hat{x}_v^H \oplus_c p_k^{hyp}, \\ x_v^E &= \log_0(x_v^H), \end{aligned} \tag{6.17}$$

where $\text{GNN}_L(\hat{x} \,|\, \theta)$ denotes the output of a L-layered GNN architecture with the trainable parameters θ, x is the input node features, x_o is the final output, and p_k^{hyp} is the hyperbolic PE of k^{th} category. x_v^E is the transformed output after adding positional encodings with the

node features.

6.3.9 Motivation behind choosing the Hyperbolic Space

Hyperbolic space is equipped with constant negative curvature where the volume of a ball grows in exponential order with respect to the radius. Unlike in the Euclidean domain where the volume of the ball grows in polynomial order because the space is free of curvature. Therefore, the input graphs can be embedded in the hyperbolic space with lower distortion than in the Euclidean space. In addition, the embeddings in hyperbolic space preserve the neighborhood of every node of the graph. In our framework, HyPE-GT learns positional encodings in the hyperbolic space, which captures the complex patterns of the neighborhoods. Therefore, the positional encodings in the hyperbolic space might be able to represent the topological characteristics corresponding to the nodes in the graph. Refer to Figure 6.1 for detailed visualizations of our proposed framework's node embeddings in the hyperbolic space. The learned PEs are thereby integrated with the node features as stated in Eqn. 6.15 6.16, and 6.17. Hence, embeddings in the hyperbolic space underscore the effectiveness of the positional encodings, which will be fed to the Graph Transformer.

6.3.10 Theoretical Analysis

In this section, we theoretically analyze the properties of distances in non-Euclidean spaces and their implications for positional encodings in graphs. We establish that distances between points in Poincaré Ball and Hyperboloid spaces are greater than their Euclidean counterparts under specific conditions (Lemma 6.1 and Lemma 6.2). Furthermore, for a connected graph, we demonstrate that the distance between the positional encodings of nodes increases when these encodings are transformed via Hyperbolic Neural Networks (HNN) or Hyperbolic Graph Convolutional Networks (HGCN), especially for nodes with higher degrees (Theorem 6.1 and Theorem 6.2). All proofs are deferred to Section 1 of the supplementary document.

6.3.10.1 Distance Properties

Lemma 6.1. *Consider an n-dimensional Poincarê Ball $\mathbb{B}^n$ of unit radius and unit curvature. Let us assume that two points $x, y \in \mathbb{B}^n$ and their distance by using Poincarê metric is*

$d_{\mathbb{B}}(x,y)$. *If we apply the Euclidean metric to them, the distance will be* $d_E(x,y)$. *Then* $\forall k \in (0,1)$, *we have* $d_{\mathbb{B}}(x,y) \geq 2k d_E(x,y)$ *if* $d_E(x,y) \in [0, \frac{\sqrt{1-k^2}}{k}]$.

Lemma 6.2. *Consider an n-dimensional Hyperboloid space* $\mathbb{H}^n$ *of unit radius and unit curvature. Let us assume that two points* $x, y \in \mathbb{H}^n$ *and their distance by using the Hyperboloid distance metric is* $d_{\mathbb{H}}(x,y)$. *If we apply the Euclidean metric to them, the distance will be* $d_E(x,y)$. *Then* $\forall k \in [1,\infty)$, *we have* $d_{\mathbb{H}}(x,y) \geq \frac{k}{2} d_E(x,y)$ $\forall d_E(x,y) \in [1, \sqrt[4]{\frac{1+k^2}{k^2}}]$.

Remark 6.1. *Lemmas 1 and* **??** *suggest that the distance between any two points lying in the non-Euclidean space (here either Poincarê Ball or Hyperboloid space) will be greater than the scaled Euclidean distance estimated between them under certain conditions.*

6.3.10.2 Distinctive Properties of Hyperbolic Positional Encodings via Learning Models

Theorem 6.1. *Consider a pair of nodes* 1 *and* 2 *of a connected graph* $\mathcal{G}$ *whose degrees are* d_1 *and* d_2 *respectively. Their initialized positional encodings are* $p_1, p_2 \in \mathbb{R}^d$. *The Euclidean distance between them is estimated as* $d_E(p_1,p_2)$. *Suppose,* p_1, p_2 *are to be transformed by either HNN or HGCN with the underlying hyperbolic space as a n-dimensional Poincarê Ball* $\mathbb{B}^n$ *of unit radius and unit curvature, then we have the following:*

1. ***HNN****: If the encodings are transformed by passing through an HNN of parameters* Θ. *The transformed encodings are respectively* p_1^{hyp} *and* p_2^{hyp} *whose distance is* $d_{\mathbb{B}}(p_1^{hyp}, p_2^{hyp})$, *then* $\exists\, \Theta'$ *such that* $d_{\mathbb{B}}(p_1^{hyp}, p_2^{hyp}) \geq k' ||\Theta'||_F d_E(p_1,p_2)$ *for some* $k' \in (0,2)$ *and* $||\Theta'||_F \leq 1$.

2. ***HGCN****: If the encodings are transformed by passing through an HGCN of parameters* Φ. *then* $\exists\, \Phi'$ *with* $||\Phi'||_F \leq 1$ *such that* $d_{\mathbb{B}}(p_1^{hyp}, p_2^{hyp}) \geq \frac{k'}{d} ||\Phi'||_F d_E(p_1,p_2)$ *where* $d = \max\{d_1, d_2\}$ *for some* $k' \in (0,2)$.

Theorem 6.2. *Consider a pair of nodes* 1 *and* 2 *of a connected graph* $\mathcal{G}$ *whose degrees are* d_1 *and* d_2 *respectively. Their initialized positional encodings are* $p_1, p_2 \in \mathbb{R}^d$. *The Euclidean distance between them is estimated as* $d_E(p_1,p_2)$. *Suppose* p_1, p_2 *are to be transformed y either HNN or HGCN with the underlying hyperbolic space as a n-dimensional Hyperboloid model* $\mathbb{H}^n$ *of unit radius and unit curvature, then we have the following:*

Table 6.2: Performance of HyPE-GT on four OGB datasets is presented. The results are the average of 10 different runs, are reported with the standard deviation. The best category of PE and the number of hyperbolic layers are mentioned in the parentheses. The top three results **first**, second, and third are marked.

Method	ogbg-molhiv	ogbg-ppa	ogbg-molpcba	ogbg-code2
	AUROC ↑	Accuracy ↑	Avg. precision ↑	F1 Score ↑
GCN+virtual node	0.7599 ± 0.0119	0.6857 ± 0.0061	0.2424 ± 0.0034	0.1595 ± 0.0018
GIN+virtual node	0.7707 ± 0.0149	0.7037 ± 0.0107	0.2703 ± 0.0023	0.1581 ± 0.0026
DeeperGCN Li et al. (2020a)	0.7858 ± 0.0117	0.7712 ± 0.0071	0.2781 ± 0.0038	0.1570 ± 0.0032
GSN (GIN+VN base) Bouritsas et al. (2022)	0.7799 ± 0.0100	-	-	
ExpC Yang et al. (2022a)	0.7799 ± 0.0082	0.7976 ± 0.0072	0.2342 ± 0.0029	-
GraphTransformer + LapPE Dwivedi and Bresson (2020)	0.7619 ± 0.0141	0.6864 ± 0.0047	0.1846 ± 0.0158	0.1738 ± 0.0381
GraphTrans (GCN-virtual) Wu et al. (2021)	-	-	0.2761 ± 0.0029	0.1830 ± 0.0024
SAN Kreuzer et al. (2021)	0.7785 ± 0.2470	-	0.2765 ± 0.0042	-
GraphGPS Rampášek et al. (2022)	0.7880 ± 0.0101	**0.8015 ± 0.0033**	0.2907 ± 0.0028	**0.1894 ± 0.0024**
Specformer Bo et al. (2023)	**0.7889 ± 0.0124**	–	**0.2972 ± 0.0023**	–
Exphormer Shirzad et al. (2023)	0.7834 ± 0.0044	–	0.2849 ± 0.0025	–
GRIT Ma et al. (2023)	0.7835 ± 0.0054	–	0.2362 ± 0.0020	–
GECO Sancak et al. (2025)	0.7780 ± 0.0200	**0.7982 ± 0.0042**	0.2961 ± 0.0008	**0.1915 ± 0.0020**
HyPE-GT (ours)	**$0.7893 \pm 0.0005(1,2)$**	**$0.7981 \pm 0.0043(6,2)$**	**$0.2967 \pm 0.0079(7,2)$**	$0.1855 \pm 0.0054(5,1)$
HyPE-GTv2 (ours)	**0.7937 ± 0.0068**	0.7932 ± 0.0091	**0.2971 ± 0.0045**	**0.1857 ± 0.0079**

1. ***HNN**: If the encodings are transformed by passing through an HNN of parameters Θ. The transformed encodings are respectively p_1^{hyp} and p_2^{hyp} whose distance is $d_{\mathbb{H}}(p_1^{hyp}, p_2^{hyp})$, then $\exists\, \Theta'$ such that $d_{\mathbb{H}}(p_1^{hyp}, p_2^{hyp}) \geq \frac{k'}{2}||\Theta'||_F d_E(p_1, p_2)$ for some $k' \in [1, \infty)$ and $||\Theta'||_F \leq 1$.*

2. ***HGCN**: If the encodings are transformed by passing through an HGCN of parameters Φ. then there exists a Φ' with $||\Phi'||_F \leq 1$ such that $d_{\mathbb{B}}(p_1^{hyp}, p_2^{hyp}) \geq \frac{k'^{\prime}}{2d}||\Phi'||_F d_E(p_1, p_2)$ where $d = \max\{d_1, d_2\}$ for some $k' \in [1, \infty)$.*

Remark 6.2. *Theorems **??** and **??** validate that there exists an HNN or HGCN architecture that transforms the positional encodings of any pair of nodes in a connected graph such that the distance between them increases under certain conditions. Furthermore, if any node has a higher degree, then it is assumed that the node is of higher importance within the graph. Our proposed framework HyPE guarantees the distinctiveness of the PEs when the nodes are of higher degrees.*

6.3.11 Complexity Analysis

HyPE-GT consists of three key modules: initialization of PEs, the choice of manifold, and learning through hyperbolic neural networks. We know that n is the number of nodes in the input graph with the dimension of node features is d. The time complexity of the Laplacian

positional encodings is $\mathcal{O}(N^3)$. The time complexity of random walk positional encodings will be $\mathcal{O}(dN^3)$.

Time Complexity of HNN The HNN performs three consecutive operations: logarithmic map, Möbius matrix-vector multiplication, and exponential map (as defined in Section 6.3.3). If the logarithmic or exponential map is applied at $x = 0$, then Möbius addition $\oplus_c$ is omitted thus the complexity is reduced to $\mathcal{O}(d)$. Consider h, b are the d-dimensional features and bias vectors in the hyperbolic space respectively. Therefore, if $W \in \mathbb{R}^{d \times d'}$, then HNN will perform $\exp_0^c(W \log_0^c(h)) \oplus_c b$. The cost of the operation will be $\mathcal{O}(dd' + d + d')$.

Time Complexity of HGCN The HGCN will pursue a similar set of operations as described for HNN except for the neighborhood aggregation. The HGCN will perform $\exp_0^c(\sum_{j \in N(i)} W \log_0^c(h)) \oplus_c b$. The updated operation will cost $\mathcal{O}(d|\mathcal{E}\|)d' + d + d')$.

The computation of Laplacian PEs or random walk PEs is accomplished in the pre-processing stages which is not expected to show impacts on the training time and inference time. During training time, we utilize the estimated positional encodings. The time complexity of the Graph Transformer is $\mathcal{O}(n^2)$. The final complexity of using HNN is $\mathcal{O}(dd')$ and for HGCN is $\mathcal{O}(d|\mathcal{E}|d')$. Incorporating the HyPE framework, the overall time complexity of the graph transformer will cost $\mathcal{O}(n^2 + dd')$ or $\mathcal{O}(n^2 + d|\mathcal{E}|d')$. If $dd' < n^2$, then we have $\mathcal{O}(n^2 + dd') \approx \mathcal{O}(n^2)$. Similarly, if $d|\mathcal{E}|d') < n^2$, then we can write $\mathcal{O}(n^2 + d|\mathcal{E}|d') \approx \mathcal{O}(n^2)$. The tactful selection of the dimension of hyperbolic space will not enhance the overall time complexity of the HyPE-GT compared to the existing standard Graph Transformer.

6.4 Experiments & Results

6.4.1 Datasets

We evaluate HyPE-GT on several benchmark datasets like CLUSTER, PATTERN, MNIST, and CIFAR10 (Dwivedi et al., 2020) for solving node-level and graph-level classification tasks. We further run experiments on large-scale graph datasets like ogbg-molhiv, ogbg-ppa, ogbg-molpcba, and ogbg-code2 from the open graph benchmark (OGB) (Hu et al., 2020a). The performance of deep GNNs is estimated on the co-author and co-purchase datasets (Shchur et al., 2018). The complete details of the datasets can be found in Section 2 of the Supple-

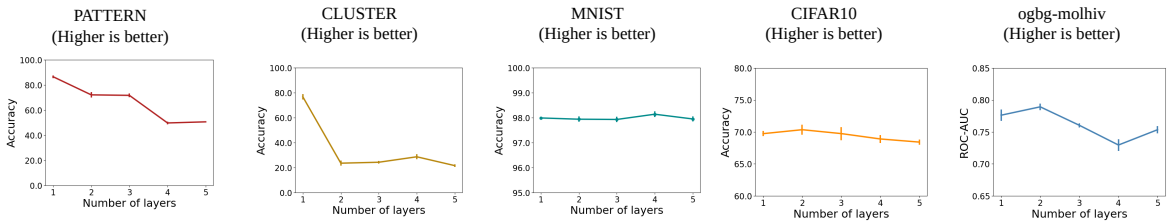


Figure 6.4: Effect of depth of hyperbolic neural architectures for all five datasets. The number of layers is varied from 1 to 5. For each layer, we averaged the 4 runs on different random seeds.

mentary.

6.4.2 Experimental Setup

We consider MNIST and CIFAR10 to perform classification on the superpixel-based graphs. The classification tasks related to molecular properties are solved on three OGB datasets ogbg-molhiv, ogbg-ppa, and ogbg-molpcba. On the other hand, the inductive node classification is performed on PATTERN and CLUSTER. Whereas the subtoken prediction task is solved on the ogbg-code2 dataset. The transductive semi-supervised node classification is performed on Amazon Photo, Amazon Computers, Coauthor CS, and Coauthor Physics. The train/valid/test splits for every dataset are provided in Table 6.3.

Table 6.3: The number of instances for each split is provided for all datasets used in the experiments.

Dataset	# train	# valid	# test
PATTERN	10000	2000	2000
CLUSTER	10000	1000	1000
MNIST	55000	5000	10000
CIFAR10	45000	5000	10000
ogbg-molhiv	3201	4113	4113
ogbg-ppa	78200	45100	34800
ogbg-molpcba	350343	43793	43793
ogbg-code2	407976	22817	21948
Amazon Computers	200	300	13252
Amazon Photo	160	240	7250
Coauthor CS	600	2250	15483
Coauthor Physics	100	150	34243

We performed experiments on different random seeds and executed 10 different runs. The final results are the average of all runs with standard deviations. The training is stabilized and becomes faster by employing Batchnorm (Ioffe and Szegedy, 2015) added between the layers

of the GT. In the case of multi-layered GNN, Layernorm (Ba et al., 2016) is applied for faster convergence of training. The parameters of the models are optimized by Adam. Overfitting is averted by utilizing `ReduceLROnPlateau` during the training process. The data-specific details of the hyper-parameters are provided in Section 4 in the Supplementary document. We also perform a comparative study on the number of parameters of HyPE-GT compared to other Graph Transformer-based approaches in Section 6 of the Supplementary document. The Pytorch and DGL-based implementation of the HyPE-GT framework is available at https://github.com/kushalbose92/HyPE-GT.

Comparison with Baselines. HyPE-GT can generate 8 different positional encodings depending on the choice of three independent submodules. In the first version, we trained HyPE-GT on eight distinct settings across all datasets and evaluated the test set on each trained model. For each dataset, we reported the best test accuracy among 8 settings. The selection of over 8 choices may incur the higher cost of training time and hyperparameter search. To ensure a fair comparison with competing methods, we now adopt a standardized evaluation protocol: the optimal configuration is first selected based solely on validation performance, and the corresponding model is then evaluated once on the test set. This procedure is replicated across all datasets, and we termed this approach HyPE-GTv2.

6.4.3 Results & Discussion

We will try to resolve the following research questions through the empirical evidences.

RQ1. How do hyperbolic positional encodings improve the performance of the standard graph transformer ?

HyPE-GT is applied on PATTERN, CLUSTER, MNIST, and CIFAR10 and corresponding results are reported in Table 6.1. We also applied HyPE-GT on the four OGB datasets whose results are presented in Table 6.2. Experiments are performed for 8 for different categories of hyperbolic PEs and we reported the best one among them with mean and standard deviation. The corresponding metrics for all datasets are also mentioned.

HyPE-GT attains 3^{rd} position on PATTERN with the PE category of 1. PATTERN contains small graphs with the target task of identifying whether a node belongs to a predefined pattern. Our framework with a single-layered HGCN successfully captures the pattern

in the neighborhoods. Our framework also secures 3^{rd} position on CLUSTER with the same category of PE as PATTERN which also contains small-sized graphs. The target task of the CLUSTER is to identify the cluster type to which each node belongs. The task requires information about the local neighborhoods, underscoring that a single-layered HGCN performs optimally. On the other hand, HyPE-GT exhibited superior performance on MNIST for the category 8 outperforming all the contenders. The results suggest the generation of PE by employing HNN entailing the importance of node features rather than the graph structure. Again, HYPE-GT secures 3^{rd} position on CIFAR10 for a PE category of 7. CIFAR10 requires higher-order interaction as it is built on RGB images and justifies interactions from the higher-order neighborhoods, underscoring the contribution of a 2-layered HGCN model.

HyPE-GT is further applied on four OGB graphs and the results are presented in Table 6.2. Our framework outperforms all contenders on ogbg-molhiv for the PE category of 1 with a 2-layered HGCN. ogbg-molhiv contains smaller graphs with chemical hierarchies and sparse edge connectivity. HyPE-GT efficiently captures hierarchies and demonstrates optimal performance. ogbg-molpcba has a structural resemblance to the ogbg-molhiv with a comparatively larger graph size. HyPE-GT secures 2^{nd} position with a PE category of 7 by employing a HGCN architecture. HyPE-GT attains 3^{rd} position on ogbg-code2 for the PE category of 5 with a 1-layered HGCN. The dataset contains programming syntax trees where higher-order interactions may not be beneficial. On the other hand, HyPE-GT attains 3^{rd} rank on ogbg-ppa but still outperforms DeeperGCN and Transformer+LapPE. The results are evidence of the utility of generating multiple hyperbolic positional encodings which provides a diverse scope to search for the optimal positional encodings.

RQ2. Can hyperbolic positional encodings improve the performance of deep GNN models ?

Graph Transformer leverages long-range interaction by measuring the attention among the node pairs which tackles the oversmoothing issue to some extent. Yet, we want to explore the effect of incorporating hyperbolic positional encodings directly with the multi-layered graph convolution-based architectures. The generated positional encodings are flexibly integrated with the learned node features from the hidden layers to boost the model performance. We

Table 6.4: The performances of GCN, JKNet, and GCNII on co-author and co-purchase networks are presented for different depths of the networks coupled with the HyPE framework. The best results are marked in **green** with the category of PE also mentioned in the parenthesis where optimal performance is obtained. (standard deviations are omitted due to space constraints)

	Method / Layers	2	4	8	16	32	64	128
Amazon Photo	GCN	85.57	84.44	51.53	49.78	52.74	52.92	50.54
	GCN + HyPE	**90.4(7)**	**80.54(8)**	**82.48(2)**	**81.24(3)**	**80.5(3)**	**80.91(3)**	**80.73(3)**
	JKNet	80.46	86.00	82.98	83.35	80.9	83.72	86.11
	JKNet + HyPE	**89.34(4)**	**90.87(4)**	**90.04(8)**	**89.67(8)**	**89.78(4)**	**89.56(4)**	**90.43(4)**
	GCNII	83.13	86.92	85.39	87.45	85.82	86.06	86.08
	GCNII + HyPE	**91.1(8)**	**92.14(7)**	**92.07(7)**	**91.32(7)**	**91.43(8)**	**91.1(7)**	**91.48(7)**
Amazon Computers	GCN	69.94	67.55	49.0	49.38	48.6	49.55	48.66
	GCN + HyPE	**82.61(8)**	**75.78(4)**	**72.86(3)**	**72.6(3)**	**73.97(3)**	**72.91(3)**	**73.75(3)**
	JKNet	64.88	74.03	53.21	54.57	58.26	55.05	67.7
	JKNet + HyPE	**80.03(5)**	**81.13(4)**	**76.86(4)**	**76.49(3)**	**75.8(3)**	**76.12(3)**	**75.63(3)**
	GCNII	71.93	74.58	64.04	72.59	69.54	69.99	68.68
	GCNII + HyPE	**82.66(8)**	**82.86(7)**	**81.48(7)**	**81.75(4)**	**80.55(7)**	**81.07(4)**	**80.75(7)**
Coauthor CS	GCN	89.89	83.83	16.42	15.37	12.01	21.36	11.66
	GCN + HyPE	**92.09(8)**	**88.96(3)**	**82.58(4)**	**82.04(4)**	**82.16(3)**	**82.17(4)**	**81.52(3)**
	JKNet	91.45	89.5	89.05	87.99	88.39	87.6	87.28
	JKNet + HyPE	**92.64(2)**	**92.58(7)**	**92.11(4)**	**92.22(8)**	**92.31(8)**	**92.24(8)**	**92.17(7)**
	GCNII	90.74	90.14	88.55	92.82	93.02	93.08	93.08
	GCNII + HyPE	**93.19(5)**	**93.01(8)**	**93.5(3)**	**93.65(3)**	**93.58(4)**	**93.68(4)**	**93.58(8)**
Coauthor Physics	GCN	93.9	90.97	89.46	51.81	52.96	49.29	51.8
	GCN + HyPE	**94.26(4)**	**93.49(3)**	**90.26(4)**	**89.88(2)**	**90.16(4)**	**89.7(2)**	**90.01(2)**
	JKNet	93.56	93.31	92.77	91.99	92.29	93.4	92.09
	JKNet + HyPE	**94.23(7)**	**94.44(8)**	**94.23(7)**	**94.33(7)**	**93.93(5)**	**94.37(8)**	**94.23(8)**
	GCNII	94.0	93.54	93.97	94.1	94.24	94.03	94.02
	GCNII + HyPE	**94.37(3)**	**94.45(4)**	**94.6(6)**	**94.62(3)**	**94.53(4)**	**94.43(3)**	**94.76(6)**

carry out an extensive experiment on co-purchase and co-author datasets Amazon Photo, Amazon Computers, Coauthor CS, and Coauthor Physics aiming to solve the task of semi-supervised mode classification. Our experiment encompasses the three well-adopted base GNN models like GCN (Kipf and Welling, 2016), JKNet (Xu et al., 2018b), and GCNII (Chen et al., 2020c). For every dataset, we applied three chosen base models. For each base GNN model, once we run experiments without using any positional encodings and in the second phase HyPE is coupled with the corresponding base model. The process is repeated for every network depth chosen from the set $\{2, 4, 8, 16, 32, 64, 128\}$. The numerical results are reported in Table 6.4. The performance is measured with the test accuracy which is obtained by taking the mean of the 10 different runs on multiple random seeds. We run experiments for all 8 categories and we only report the optimal one among them. The category of PE is

also mentioned along with the highlighted test accuracy for HyPE-GT.

The reported results are evident for the better applicability of the hyperbolic positional encodings integrating with deeper GNN architectures. The base models witnessed an uptick in performance when associated with the HyPE framework compared with the performance of the same without involving the positional encodings at every network depth. The optimal results are obtained for different categories of PEs which also indicates the benefits of generating a diverse set of hyperbolic positional encodings. The node features in the embedding space collapse with the increase of convolutional layers due to the decrease in the inter-cluster distance. The incorporation of positional encodings with the features from the hidden layers prevents the features from collapsing signifying the control of oversmoothing. Notably, the hyperbolic PEs are sufficiently capable of separating similar nodes toward each other, increasing inter-cluster distance. The performance of vanilla GCN is typically hindered by the effect of oversmoothing which is alleviated by employing the HyPE framework. Again the performance improvement on GCNII and JKNet is lesser than GCN due to those models are already designed for controlling oversmoothing but our framework is still able to outperform with a good margin which underlines the efficiency of the proposed framework.

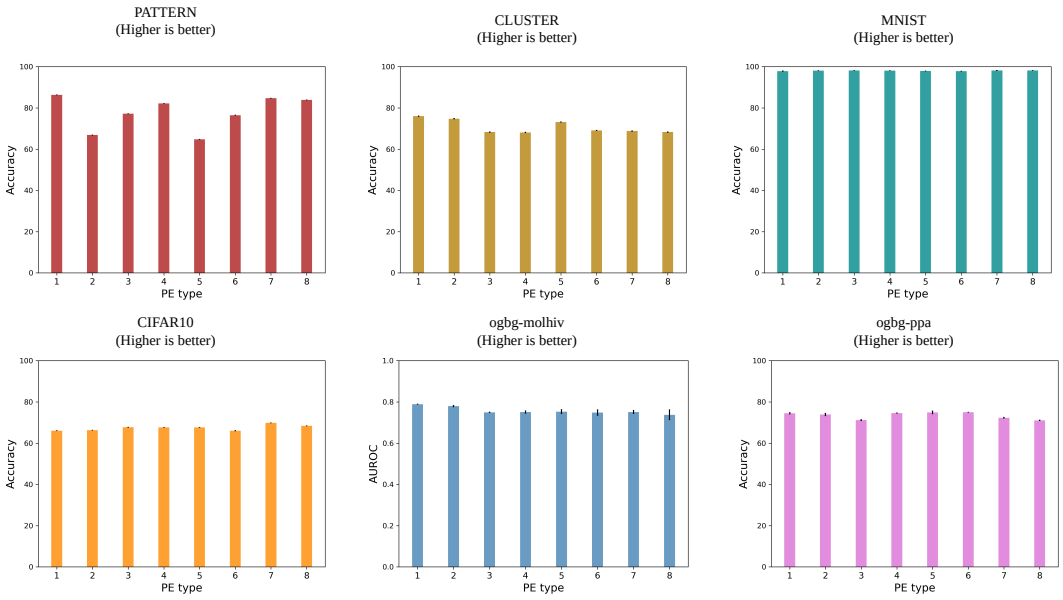


Figure 6.5: The performance of HyPE-GT on six benchmark datasets is presented. Each possible combination of the key modules in the framework is considered. All eight types of positional encodings are generated for each dataset.

6.4.4 Depth of the Hyperbolic Networks in HyPE-GT

We investigated the effect of the depth of hyperbolic networks (both HNN and HGCN) on the performance of HyPE-GT. We include PATTERN, CLUSTER, MNIST, CIFAR10, and ogbg-molhiv to conduct the experiments. HyPE-GT generates 8 different positional encodings for every dataset and we only run the experiment for the category where the best performance is obtained. For example, MNIST attains the best performance for the PE category of 8. The best category can be found in the results available in Table 6.1 and 6.2. We vary the number of layers of the respective hyperbolic networks from 1-5. For each network depth, we run the experiments for 4 times and plot the mean along with standard deviations in Figure 6.4. The performance metric is also mentioned against each plot.

The performance of HyPE-GT deteriorates on PATTERN and CLUSTER due to the oversmoothing issue in the hyperbolic graph convolutional-based architectures. Yet, these datasets contain graphs with smaller radii and performance depends on local substructures. Thus, increasing depth might not be beneficial for the performance gain. HyPE-GT exhibited stable performance on MNIST as it hinges on the HNN and mostly relies on the rich node features. On the other side, the performance of HyPE-GT on CIFAR10 is optimal when the network depth is 2. The superpixel graphs of CIFAR10 are created from RGB images which requires aggregating features from higher-order neighborhoods. Thus, the degradation of performance is lower compared to PATTERN and CLUSTER. The performance on ogbg-molhiv is optimal when the network depth is 2. The molecular graphs containing local substructures prefer the localized feature aggregation which is further validated by the performance degradation with increased network depth.

6.5 Ablation Study

We conduct a comprehensive ablation study on our proposed framework HyPE-GT to analyze the impact of individual modules within the framework. HyPE-GT comprises three modules as discussed earlier. We explore various options for each of the modules which prompts the generation of 8 different positional encodings. We run the experiments on PATTERN, CLUSTER, MNIST, CIFAR10, ogbg-molhiv, and ogbg-ppa, and the results are presented in Figure 6.5. The ablation study on ogbg-molpcba and ogbg-code2 can be found in Section 5 of

the Supplementary document. The experiments are executed for each category of positional encodings for every dataset. Each result is the average of 10 runs along with the standard deviation with different random seeds. The variation in the corresponding performance metric can be observed across 8 different positional encodings. Notably, optimal performances are obtained for certain combinations of individual modules in the framework, highlighting the interdependence of the modules. The capacity to generate multiple PEs of HyPE broadens the scope of finding the best one for solving target tasks.

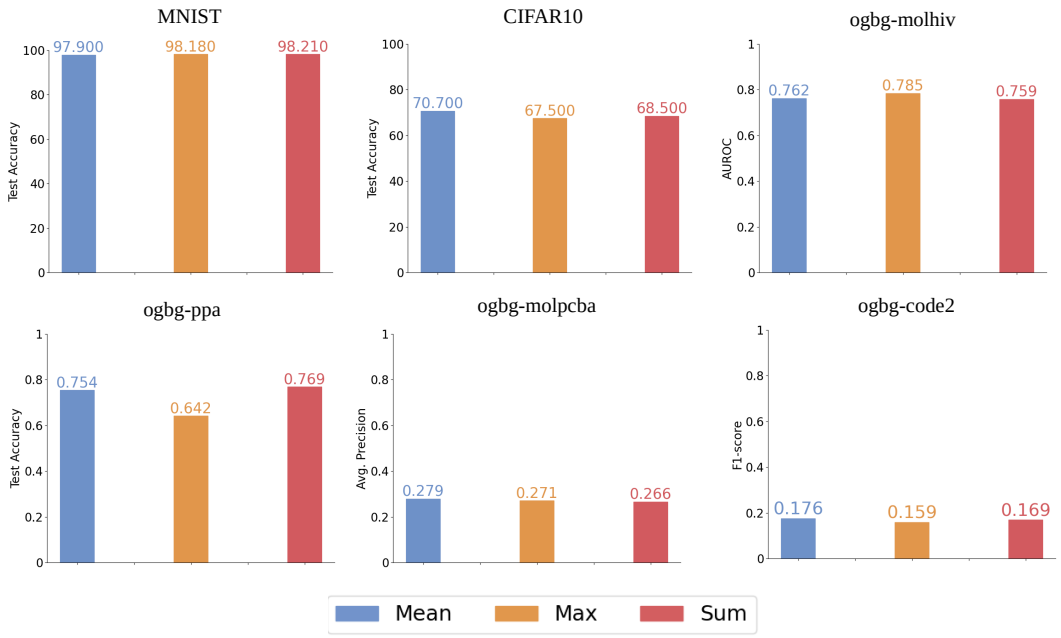


Figure 6.6: Various readout methods like Mean, Max, and Sum are applied on both 6 graph classification datasets, and their effects are presented. The corresponding metrics are mentioned. The optimal performance can be obtained by tactfully selecting the readout methods which entirely depends on the input graphs.

6.5.1 Selection Criteria of Positional Encodings

An obvious question will naturally arise regarding the determination of the optimal triplet from the set of positional encodings for solving the downstream tasks. We attempt to resolve the issue by analyzing the experimental results. We devised an intuitive strategy to minimize the search time and avoid the time-consuming effort of searching randomly for the best positional encoding. Our framework achieves optimal results on PATTERN, CLUSTER,

CIFAR10, ogbg-molhiv when the PEs are learned via HGCN rather than HNN. Therefore, HGCN may be an appropriate candidate to initiate the search as it captures the structural patterns of the input graph. Only HyPE-GT on MNIST shows the best performance when HNN is employed also the second best result occurred with HNN. Experiments suggest that Hyperboloid dominates over PoincarêBall in most of the cases. Again LapPE and RWPE both work well in the experiments. We still recommend an exhaustive search to find the most effective positional encodings for the downstream tasks.

6.6 Effect of the Readout Methods

The selection of the readout methods is pivotal for performing the graph classification tasks. The fact prompts us to study the effects of three well-adopted readout methods `Mean`, `Max`, and `Sum` when HyPE-GT is applied on the MNIST, CIFAR10, ogbg-molhiv, ogbg-moppa, ogbg-molpcba, and ogbg-code2. The performance of HyPE-GT on different readout methods applied on 6 datasets is elucidated in Figure 6.6. The Sum readout emerged as beneficial for MNIST but the Mean readout is more effective for CIFAR10. Similarly, ogbg-molhiv, ogbg-ppa, ogbg-molpcba, and ogbg-code2 exhibited optimal performances when the readout methods were Max, Sum, and Mean, respectively. The variations among the readout methods across all datasets are relatively lower, which emphasizes the resilience of HyPE-GT toward the choice of readout methods. The experiments also underscore the importance of choosing appropriate readout methods to obtain the best performance on the graph classification tasks. More results can be found in the Appendix 6.10.

6.7 Limitations

HyPE-GT excels at capturing intricate hierarchical relationships within graphs by generating learnable positional encodings in hyperbolic space. This approach is particularly effective when dealing with input graphs that possess inherent hierarchical structures, as hyperbolic space provides superior encoding capabilities compared to Euclidean space. However, not all input graphs exhibit hierarchical characteristics. In such cases, positional encodings learned in Euclidean space may be more appropriate. For instance, the performance of HyPE-GT on

PATTRN and CLUSTER datasets, which lack hierarchical components, shows slight degradation. Conversely, HyPE-GT performs commendably on the ogbg-molhiv and ogbg-molpcba datasets, which contain hierarchical structures. These results demonstrate that HyPE-GT is proficient in generating effective positional encodings that align well with hierarchical structures. Moreover, they underscore the framework's versatility and robustness in handling graphs with varying degrees of hierarchical complexity.

6.8 Conclusion

In this work, we proposed a novel framework called HyPE-GT to generate positional encodings in the hyperbolic space for Graph Transformers. Unlike the other existing methods, our framework can generate a set of positional encodings that offer diverse choices for solving downstream tasks. The generated PEs are learned either by HNN or HGCN-based architectures. The efficiency of HyPE-GT is also validated by performing several experiments on molecular graphs from benchmark datasets (Dwivedi et al., 2020) and OGB (Hu et al., 2020a) graph datasets, and achieving impressive performances on the datasets. We provided the results of an exhaustive ablation study to substantiate the importance of each component of HyPE-GT. We also re-purpose the positional encodings to integrate with node features to boost the performance of deep graph neural networks applied on Co-author and Co-purchase datasets. Exploring hyperbolic spaces to learn positional encodings may be a potential avenue for future directions. Also, further investigation is required on the effectiveness of positional encodings in deeper GNNs to boost performance.

6.9 Appendix A

6.9.1 Proofs

Lemma 6.1 Consider an n-dimensional Poincarê Ball $\mathbb{B}^n$ of unit radius and unit curvature. Let us assume that two points $x, y \in \mathbb{B}^n$ and their distance by using Poincarê metric is $d_{\mathbb{B}}(x, y)$. If we apply the Euclidean metric to them, the distance will be $d_E(x, y)$. Then $\forall k \in (0, 1)$, we have $d_{\mathbb{B}}(x, y) \geq 2k d_E(x, y)$ if $d_E(x, y) \in [0, \frac{\sqrt{1-k^2}}{k}]$.

Proof. Assume Poincarê Ball $\mathbb{B}^n = \{x \in \mathbb{R}^n, ||x|| \leq 1\}$ is equipped with Poincarê metric and

the distance between two points $x, y \in \mathbb{B}^n$ as following

$$d_{\mathbb{B}}(x,y) = \cosh^{-1}\left(1 + \frac{2||x-y||^2}{(1-||x||^2)(1-||y||^2)}\right) \tag{6.18}$$

The Euclidean norm between x, y is

$$d_E(x,y) = ||x-y|| \tag{6.19}$$

The following can be written

$$\begin{aligned} d_{\mathbb{B}}(x,y) &= \cosh^{-1}\left(1 + \frac{2||x-y||^2}{(1-||x||^2)(1-||y||^2)}\right) \\ &= 2\sinh^{-1}\left(\sqrt{\frac{\Delta(x,y)}{2}}\right) \end{aligned} \tag{6.20}$$

where

$$\Delta(x,y) = \frac{2||x-y||^2}{(1-||x||^2)(1-||y||^2)}$$

We have considered the unit radius of the Poincarê Ball, $||x||, ||y|| \leq 1$. then $(1-||x||^2)(1-||y||^2) \leq 1$. Therefore, we have

$$\Delta(x,y) \geq 2||x-y||^2 \tag{6.21}$$

We have $\frac{d}{dx}(\sinh^{-1}(x)) = \frac{1}{\sqrt{1+x^2}} > 0$ which implies that $\sinh^{-1}(x)$ is an increasing function. Therefore, we have,

$$\begin{aligned} &2\sinh^{-1}\left(\sqrt{\frac{\Delta(x,y)}{2}}\right) \geq 2\sinh^{-1}(||x-y||) \\ &d_{\mathbb{B}}(x,y) \geq 2\sinh^{-1}(||x-y||) \end{aligned} \tag{6.22}$$

We know that $\sinh^{-1}(z) = \ln\left(z + \sqrt{z^2+1}\right)$. Let us consider the following function $\forall k \in \mathbb{R}$

$$f(z) = \ln\left(z + \sqrt{z^2+1}\right) - kz \tag{6.23}$$

Differentiating w.r.t. z, we get

$$f'(z) = \frac{1}{\sqrt{1+x^2}} - k \tag{6.24}$$

Also $f(0) = 0$ and the function is increasing when $f'(z) \geq 0$

$$\begin{aligned} f'(z) &= \frac{1}{\sqrt{1+x^2}} - k \geq 0 \\ \frac{1}{\sqrt{1+x^2}} &\geq k \\ z &\leq \frac{\sqrt{1-k^2}}{k} \end{aligned} \tag{6.25}$$

If the above condition holds then $f(z)$ increases and non-negative $\forall z \in [0, \frac{\sqrt{1-k^2}}{k}]$. Then we have

$$\begin{aligned} &\ln\Big(z + \sqrt{z^2+1}\Big) - kz \geq 0 \\ &\ln\Big(z + \sqrt{z^2+1}\Big) \geq kz \\ &\sinh^{-1}(z) \geq kz \ \forall z \in [0, \frac{\sqrt{1-k^2}}{k}] \end{aligned} \tag{6.26}$$

Applying the inequality to the distance between points x, y and $\forall ||x-y|| \in [0, \frac{\sqrt{1-k^2}}{k}]$ we have,

$$\begin{aligned} &\sinh^{-1}(||x-y||) \geq k||x-y|| \\ &2\sinh^{-1}(||x-y||) \geq 2k||x-y|| \\ &d_{\mathbb{B}}(x,y) \geq 2k||x-y|| \end{aligned} \tag{6.27}$$

Therefore, the distance between any two points x, y increases when the Poincarê metric is applied compared to the scaled Euclidean distance under certain conditions. □

Lemma 6.2 Consider an n-dimensional Hyperboloid space $\mathbb{H}^n$ of unit radius and unit curvature. Let us assume that two points $x, y \in \mathbb{H}^n$ and their distance by using the Hyperboloid distance metric is $d_{\mathbb{H}}(x,y)$. If we apply the Euclidean metric to them, the distance will be $d_E(x,y)$. Then $\forall k \in [1, \infty)$, we have $d_{\mathbb{H}}(x,y) \geq \frac{k}{2} d_E(x,y)$ $\forall d_E(x,y) \in [1, \sqrt[4]{\frac{1+k^2}{k^2}}]$.

Proof. Let us define the Hyperboloid space in the following way

$$\mathbb{H}_1^n = \{x \in \mathbb{R}^{n+1} : \langle x, x\rangle_{\mathcal{L}} = -1, x_0 > 0\}, \tag{6.28}$$

where $\langle x, y\rangle_{\mathcal{L}}$ denotes Minkowski inner product is defined as follows

$$\langle x, y\rangle_{\mathcal{L}} = -x_0 y_0 + x_1 y_1 + \cdots + x_n y_n$$

The distance between x, y can be estimated as

$$\begin{aligned} d_{\mathbb{H}}(x, y) &= \cosh^{-1}(-\langle x, y\rangle_{\mathcal{L}}) \\ &= \cosh^{-1}(-(-x_0 y_0 + \sum_{i=1}^{n} x_i y_i)) \\ &= \cosh^{-1}(x_0 y_0 - \sum_{i=1}^{n} x_i y_i) \end{aligned} \tag{6.29}$$

For two points x, y, following the condition of $\langle x, x\rangle_{\mathcal{L}} = 1$, we have,

$$\begin{aligned} \langle x, x\rangle_{\mathcal{L}} &= -x_0^2 + x_1^2 + \cdots x_n^2 = -1 \\ \langle y, y\rangle_{\mathcal{L}} &= -y_0^2 + y_1^2 + \cdots y_n^2 = -1 \end{aligned} \tag{6.30}$$

The Euclidean distance between x, y can be expressed as

$$\begin{aligned} d_E(x, y) &= \sqrt{\sum_{i=0}^{n}(x_i - y_i)^2} \\ &= \sqrt{\sum_{i=0}^{n} x_i^2 + \sum_{i=0}^{n} y_i^2 - 2\sum_{i=0}^{d} x_i y_i} \\ &= \sqrt{x_0^2 + \sum_{i=1}^{n} x_i^2 + y_0^2 + \sum_{i=1}^{n} x_i^2 - 2(x_0 y_0 + \sum_{i=1}^{n} x_i y_i)} \end{aligned} \tag{6.31}$$

Using from Eq. 6.29 and Eq. 6.30, we have

$$d_E(x, y) = \sqrt{2x_0^2 + 2y_0^2 - 2 - 2(2x_0 y_0 - \cosh(d_{\mathbb{H}}(x, y)))}$$

$$d_E^2(x, y) = 2x_0^2 + 2y_0^2 - 2 - 4x_0 y_0 + 2\cosh(d_{\mathbb{H}}(x, y))$$

$$d_E^2(x, y) = 2((x_0 - y_0)^2 - 1) + 2\cosh(d_{\mathbb{H}}(x, y))$$

Assuming that $(x_0 - y_0)^2 \leq 1$, then we can express

$$\begin{aligned} 2\cosh(d_{\mathbb{H}}(x,y)) &\geq d_E^2(x,y) \\ d_{\mathbb{H}}(x,y) &\geq \cosh^{-1}\left(\frac{d_E^2(x,y)}{2}\right) \end{aligned} \tag{6.32}$$

Consider the function $f(z) = \cosh^{-1}(z) - kz$ where $\cosh^{-1}(z) = \ln\left(z + \sqrt{z^2-1}\right)$. Differentiating $f(z)$ w.r.t z , we get

$$\begin{aligned} f'(z) &= \frac{1}{\sqrt{z^2-1}} - k \geq 0 \\ \frac{1}{\sqrt{z^2-1}} &\geq k \\ z &\leq \frac{\sqrt{1+k^2}}{k} \end{aligned} \tag{6.33}$$

Also, $f(0) = 0$, and the function is increasing for the above condition. Therefore, we have

$$\begin{aligned} f(z) &= \cosh^{-1}(z) - kz \geq 0 \\ \cosh^{-1}(z) &\geq kz \;\forall z \in [1, \frac{\sqrt{1+k^2}}{k}] \end{aligned} \tag{6.34}$$

Applying the above inequality, finally, we have,

$$\begin{aligned} d_{\mathbb{H}}(x,y) &\geq \cosh^{-1}\left(\frac{d_E^2(x,y)}{2}\right) \geq k\frac{d_E^2(x,y)}{2} \\ d_{\mathbb{H}}(x,y) &\geq \frac{k}{2}d_E^2(x,y) \geq \frac{k}{2}d_E(x,y) \;\forall d_E^2(x,y) \in [1, \frac{\sqrt{1+k^2}}{k}] \\ d_{\mathbb{H}}(x,y) &\geq \frac{k}{2}d_E(x,y) \;\forall\, d_E(x,y) \in [1, \sqrt[4]{\frac{1+k^2}{k^2}}] \end{aligned} \tag{6.35}$$

Therefore, the distance between two points in the hyperboloid space will be greater than their scaled Euclidean distance under certain conditions. □

Theorem 6.1 Consider a pair of nodes 1 and 2 of a connected graph $\mathcal{G}$ whose degrees are d_1 and d_2 respectively. Their initialized positional encodings are $p_1, p_2 \in \mathbb{R}^d$. The Euclidean distance between them is estimated as $d_E(p_1, p_2)$. Suppose, p_1, p_2 are to be transformed by either HNN or HGCN with the underlying hyperbolic space as a n-dimensional Poincarê Ball $\mathbb{B}^n$ of unit radius and unit curvature, then we have the following:

1. **HNN** If the encodings are transformed by passing through an HNN of parameters Θ. The transformed encodings are respectively p_1^{hyp} and p_2^{hyp} whose distance is $d_{\mathbb{B}}(p_1^{\text{hyp}}, p_2^{\text{hyp}})$, then $\exists\, \Theta'$ such that $d_{\mathbb{B}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) \geq k'||\Theta'||_F d_E(p_1, p_2)$ for some $k' \in (0, 2)$ and $||\Theta'||_F \leq 1$.

2. **HGCN** If the encodings are transformed by passing through an HGCN of parameters Φ. then $\exists\, \Phi'$ with $||\Phi'||_F \leq 1$ such that $d_{\mathbb{B}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) \geq \frac{k'}{d}||\Phi'||_F d_E(p_1, p_2)$ where $d = \max\{d_1, d_2\}$ for some $k' \in (0, 2)$.

Proof. We will outline the complete proof for each of the parts.

HNN The Euclidean distance between two points x, y is estimated as $d_E(x, y) = ||p_1 - p_2||$. We want to compute the distance between p_1, p_2 in the $\mathbb{B}_c^n$. Therefore, we will apply the exponential map to project the points in the manifold space. The exponential map for $\mathbb{B}_c^n$ at 0 is defined as following,

$$\exp_c^0(v) = \tanh\big(\sqrt{c}||v||\big)\frac{v}{c||v||}, \tag{6.36}$$

where v is any point lying on the tangent space which resembles the locally linear space. As we considered the curvature to be 1, the exponential map can be reformulated as,

$$\exp_1^0(v) = \tanh(||v||)\frac{v}{||v||}, \tag{6.37}$$

From now on we will write as $\exp(v)$. The Möbius matrix-vector multiplication is defined as,

$$M^{\otimes_c}(x) = \frac{1}{\sqrt{c}}\tanh\left(\frac{||Mx||}{||x||}\tanh^{-1}(\sqrt{c}||x||)\right)\frac{Mx}{||Mx||}, \tag{6.38}$$

where $M \in \mathcal{M}_{m,n}(\mathbb{R})$ and $x \in \mathbb{B}_c^n$. Substituting $c = 1$, we get

$$M^{\otimes_1}(x) = \tanh\left(\frac{||Mx||}{||x||}\tanh^{-1}(||x||)\right)\frac{Mx}{||Mx||}, \tag{6.39}$$

If we pass the positional encodings through a hyperbolic neural network (HNN) with a trainable weight $\Theta \in \mathbb{R}^{d\times d'}$. Firstly, we mapped the encodings into the manifold space using the

exponential map.

$$\hat{p}_1 = \exp(p_1) = \tanh(||p_1||)\frac{p_1}{||p_1||}$$
$$\hat{p}_2 = \exp(p_2) = \tanh(||p_2||)\frac{p_2}{||p_2||}$$

Replacing the scalar terms $\zeta_1 = \frac{\tanh(||p_1||)}{||p_1||}$ and $\zeta_2 = \frac{\tanh(||p_2||)}{||p_2||}$. Then, we have $\hat{p}_1 = \zeta_1 p_1$ and $\hat{p}_2 = \zeta_2 p_2$.

The encodings are now mapped on the manifold. The encodings are transformed by passing through HNN, then applying Möbius matrix-vector formula, we have

$$p_1^{\text{hyp}} = \Theta^{\otimes_1}(\hat{p}_1) = \tanh\left(\frac{||\hat{p}_1\Theta||}{||\hat{p}_1||}\tanh^{-1}(||\hat{p}_1||)\right)\frac{\hat{p}_1\Theta}{||\hat{p}_1\Theta||}$$

Substituting $\hat{p}_1 = \zeta_1 p_1$

$$= \tanh\left(\frac{||p_1\Theta||}{||p_1||}\tanh^{-1}(||\zeta_1 p_1||)\right)\frac{p_1\Theta}{||p_1\Theta||}$$

Similarly, we can write

$$p_2^{\text{hyp}} = \Theta^{\otimes_1}(\hat{p}_2) = \tanh\left(\frac{||p_2\Theta||}{||p_2||}\tanh^{-1}(||\zeta_2 p_2||)\right)\frac{p_2\Theta}{||p_2\Theta||}$$

Replacing the scalar terms as $\eta_1 = \frac{\tanh^{-1}(||\zeta_1 p_1||)}{||p_1||}$ and $\eta_2 = \frac{\tanh^{-1}(||\zeta_2 p_2||)}{||p_2||}$, we have $p_1^{\text{hyp}} = \tanh(\eta_1||p_1\Theta||)\frac{p_1\Theta}{||p_1\Theta||}$ and $p_2^{\text{hyp}} = \tanh(\eta_2||p_2\Theta||)\frac{p_2\Theta}{||p_2\Theta||}$. Further substituting the scalar terms as $\omega_1 = \frac{\tanh(\eta_1||p_1\Theta||)}{||p_1\Theta||}$ and $\omega_2 = \frac{\tanh(\eta_2||p_2\Theta||)}{||p_2\Theta||}$. Therefore, we can write as,

$$p_1^{\text{hyp}} = \omega_1 p_1\Theta \qquad\qquad p_2^{\text{hyp}} = \omega_2 p_2\Theta \tag{6.40}$$

Therefore, the length of the geodesic on the manifold will be,

$$d_{\mathbb{B}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) = \cosh^{-1}\left(1 + \frac{2||p_1^{\text{hyp}} - p_2^{\text{hyp}}||^2}{(1-||p_1^{\text{hyp}}||^2)(1-||p_2^{\text{hyp}}||^2)}\right)$$
$$= \cosh^{-1}\left(1 + \frac{2||\omega_1 p_1\Theta - \omega_2 p_2\Theta||^2}{(1-||\omega_1 p_1\Theta||^2)(1-||\omega_2 p_2\Theta||^2)}\right)$$

We know $\frac{\tanh(z)}{z} \leq 1 \forall z \in \mathbb{R}$. The $\frac{\tanh^{-1}(tz)}{z} \leq 1 \ \forall |t| \leq 1, |z| \leq 1$. Therefore, we can say that $\zeta_1, \zeta_2 \leq 1$. We use this notion to have $\eta_1, \eta_2 \leq 1$. Finally, $\omega_1, \omega_2 \leq 1 \ \forall z \in \mathbb{R}$.

Applying the inequalities, $||\omega_1 p_1 \Theta|| \leq \omega_1 ||p_1|| ||\Theta||_F$. As we considered unit radius $||p_1|| \leq 1$ and assumed a bounded Frobenius norm of the HNN parameters $||\Theta||_F \leq 1$. Thus we have $||\omega_1 p_1 \Theta|| \leq 1$ and in a similar way $||\omega_2 p_2 \Theta|| \leq 1$ which implies $(1 - ||\omega_1 p_1 \Theta||^2)(1 - ||\omega_2 p_2 \Theta||^2) \leq 1$.

$$d_{\mathbb{B}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) = 2 \sinh^{-1} \left(\sqrt{\frac{\Delta(p_1^{\text{hyp}}, p_2^{\text{hyp}})}{2}} \right) \tag{6.41}$$

where

$$\Delta(p_1^{\text{hyp}}, p_2^{\text{hyp}}) = \frac{2||\omega_1 p_1 \Theta - \omega_2 p_2 \Theta||^2}{(1 - ||\omega_1 p_1 \Theta||^2)(1 - ||\omega_2 p_2 \Theta||^2)}$$

We have $\frac{d}{dx}(\sinh^{-1}(x)) = \frac{1}{\sqrt{1+x^2}} > 0$ which implies that $\sinh^{-1}(x)$ is an increasing function. Therefore, we have,

$$\begin{aligned} &2 \sinh^{-1} \left(\sqrt{\frac{\Delta(p_1^{\text{hyp}}, p_2^{\text{hyp}})}{2}} \right) \geq 2 \sinh^{-1}(||p_1^{\text{hyp}} - p_2^{\text{hyp}}||) \\ &d_{\mathbb{B}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) \geq 2 \sinh^{-1}(||p_1^{\text{hyp}} - p_2^{\text{hyp}}||) \end{aligned} \tag{6.42}$$

WLOG, we can assume $\omega_1 \geq \omega_2$ and apply properties of Euclidean norm. Then we have,

$$\begin{aligned} ||p_1^{\text{hyp}} - p_2^{\text{hyp}}|| &= ||\omega_1 p_1 \Theta - \omega_2 p_2 \Theta|| \\ &\geq (||\omega_1 p_1 \Theta|| - ||\omega_2 p_2 \Theta||) \\ &\geq \omega_2 (||p_1 \Theta|| - ||p_2 \Theta||) \end{aligned} \tag{6.43}$$

Again, we can write,

$$\omega_2 (||p_1 \Theta|| - ||p_2 \Theta||) \leq \omega_2 (||p_1 \Theta - p_2 \Theta||) \tag{6.44}$$

Using the properties of the vector norms, we have,

$$\omega_2 ||p_1 \Theta - p_2 \Theta|| \leq \omega_2 ||\Theta||_F ||p_1 - p_2|| \tag{6.45}$$

Combining Eqn. 6.44 and 6.45, we have,

$$\omega_2||\Theta||_F||p_1 - p_2|| \geq \omega_2(||p_1\Theta|| - ||p_2\Theta||) \tag{6.46}$$

Therefore, $\exists\ \Theta'$ with $||\Theta'||_F \leq 1$ such that,

$$|p_1^{\text{hyp}} - p_2^{\text{hyp}}|| \geq \omega_2||\Theta'||_F||p_1 - p_2|| \geq \omega_2(||p_1\Theta|| - ||p_2\Theta||) \tag{6.47}$$

Now combining with the Eq. 6.42 and using the notion from Theorem 1, we have,

$$\begin{aligned} 2\sinh^{-1}(||p_1^{\text{hyp}} - p_2^{\text{hyp}}||) &\geq 2k||p_1^{\text{hyp}} - p_2^{\text{hyp}}|| \\ \forall||p_1^{\text{hyp}} - p_2^{\text{hyp}}|| &\in [0, \frac{\sqrt{1-k^2}}{k}] \\ d_{\mathbb{B}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) &\geq 2k\omega_2||\Theta'||_F||p_1 - p_2|| \\ &= k'||\Theta'||_F||p_1 - p_2|| \\ &\text{where } k' = 2k\omega_2 \text{ with } k' \in (0, 2) \end{aligned} \tag{6.48}$$

HGCN If we employ an HGCN architecture with trainable parameters Φ, we need to incorporate normalized adjacency matrix $\tilde{A}$ into the positional encodings. The transformed encoding is mapped into the manifold's tangent space using the logarithmic map to enable multiplication with $\tilde{A}$. The logarithmic or simply log map at 0 position is defined as

$$\log_0^c(v) = \tanh^{-1}(\sqrt{c}||v||)\frac{v}{\sqrt{c}||v||} \tag{6.49}$$

Applying the log map with $c = 1$, the encodings are transformed into the tangent space as following

$$\begin{aligned} p_1^T &= \tanh^{-1}(||p_1^{\text{hyp}}||)\frac{p_1^{\text{hyp}}}{||p_1^{\text{hyp}}||} \\ p_2^T &= \tanh^{-1}(||p_2^{\text{hyp}}||)\frac{p_2^{\text{hyp}}}{||p_2^{\text{hyp}}||} \end{aligned} \tag{6.50}$$

Consider $\eta_1' = \frac{\tanh^{-1}(||p_1^{\text{hyp}}||)}{||p_1^{\text{hyp}}||}$ and $\eta_2' = \frac{\tanh^{-1}(||p_2^{\text{hyp}}||)}{||p_2^{\text{hyp}}||}$ with $\eta_1', \eta_2' \geq 1$. Therefore, we can express $p_1^T = \eta_1' p_1^{\text{hyp}}$ and $p_2^T = \eta_2' p_2^{\text{hyp}}$. The adjacency matrix is applied to the transformed

encodings which result in $p_1^A = \frac{1}{d_1} \sum_{j \in N(1) \cup p_1^T} p_j^T$ and $p_2^A = \frac{1}{d_2} \sum_{j \in N(2) \cup p_2^T} p_j^T$ where d_1, d_2 are the degrees for node 1 and 2 respectively with p_j^T are the corresponding neighbors' features. Suppose, $\exists\ C_1, C_2 \in \mathbb{R}$ such that,

$$||p_1^A|| \geq \frac{1}{d_1} C_1 ||p_1^T|| \qquad\qquad ||p_2^A|| \geq \frac{1}{d_2} C_2 ||p_2^T|| \tag{6.51}$$

From Eq 6.40, we have $||p_1^A|| \geq \frac{1}{d_1} C_1 \eta_1' ||p_1^{\text{hyp}}|| = \frac{1}{d_1} C_1 \eta_1' \omega_1 ||p_1 \Phi|| = \frac{1}{d_1} \omega_1' ||p_1 \Phi||$ where $\omega_1' = C_1 \eta_1' \omega_1$. Similarly, we have $||p_2^A|| \geq \frac{1}{d_2} \omega_2' ||p_2 \Phi||$ where $\omega_2' = C_2 \eta_2' \omega_2$. As $\omega_1, \omega_2 \leq 1$ with the assumption of

Again, we must apply the exponential map to revert the embeddings to the Poincarê Ball.

$$\begin{aligned} p_1^{\text{Ah}} &= \tanh\big(||p_1^A||\big) \frac{p_1^A}{||p_1^A||} \\ p_2^{\text{Ah}} &= \tanh\big(||p_2^A||\big) \frac{p_2^A}{||p_2^A||} \end{aligned} \tag{6.52}$$

Let us assume that $\tau_1 = \frac{\tanh\big(||p_1^A||\big)}{||p_1^A||}$ and $\tau_2 = \frac{\tanh\big(||p_2^A||\big)}{||p_2^A||}$ with $\tau_1, \tau_2 \leq 1$. Therefore, $p_1^{Ah} = \tau_1 p_1^A$ and $p_2^{Ah} = \tau_1 p_2^A$.

Similarly, we have $p_2^{Ah} = \frac{1}{d_2} \tau_2 p_2^A \geq \frac{1}{d_2} \tau_2 \omega_2' p_2 \Phi = \frac{1}{d_2} \tau_2' p_2 \Phi$ where $\tau_2' = \tau_2 \omega_2'$.

The distance between p_1^{Ah} and p_2^{Ah} is

$$\begin{aligned} d_{\mathbb{B}}(p_1^{Ah}, p_2^{Ah}) &= \cosh^{-1}\left(1 + \frac{2||p_1^{Ah} - p_2^{Ah}||^2}{(1 - ||p_1^{Ah}||^2)(1 - ||p_2^{Ah}||^2)}\right) \\ &= \cosh^{-1}\left(1 + \frac{2||\frac{1}{d_1}\tau_1' p_1 \Phi - \frac{1}{d_2}\tau_2' p_2 \Phi||^2}{(1 - ||\frac{1}{d_1}\tau_1' p_1 \Phi||^2)(1 - ||\frac{1}{d_2}\tau_2' p_2 \Phi||^2)}\right) \end{aligned} \tag{6.53}$$

We have

$$
\begin{aligned}
||p_1^{Ah}|| = \tau_1||p_1^A|| &= \frac{\tau_1}{d_1}||\sum_{j\in N(1)\cup p_1^T} p_j^T|| \\
&\leq \frac{\tau_1}{d_1}\sum_{j\in N(1)\cup p_1^T}||p_j^T|| \\
&\leq \frac{\tau_1}{d_1}\sum_{j\in N(1)\cup p_1^T}||p_j^T|| \\
\text{where } \eta_j' &= \frac{\tanh^{-1}(||p_j^{\text{hyp}}||)}{||p_j^{\text{hyp}}||} \\
&= \frac{\tau_1}{d_1}\sum_{j\in N(1)\cup p_1^T}\eta_j'||p_j^{\text{hyp}}|| \\
&\leq \frac{\tau_1}{d_1}\sum_{j\in N(1)\cup p_1^T}\eta_{\max}' = \tau_1\eta_{\max}'
\end{aligned}
\tag{6.54}
$$

Assuming $\tau_1\eta_{\max}' \leq 1$, then we have $||p_1^{Ah}|| \leq 1$. Similarly, $||p_2^{Ah}|| \leq 1$. Using these inequalities, we have $(1-||p_1^{Ah}||^2)(1-||p_2^{Ah}||^2) \leq 1$. We can express the following

$$
d_{\mathbb{B}}(p_1^{Ah}, p_2^{Ah}) \geq 2\sinh^{-1}(||p_1^{Ah} - p_2^{Ah}||) \tag{6.55}
$$

Now we can write the following,

$$
\begin{aligned}
||p_1^{Ah} - p_2^{Ah}|| &= ||\tau_1 p_1^A - \tau_2 p_2^A|| \\
&\geq \tau_1||p_1^A|| - \tau_2||p_2^A|| \\
&\geq \frac{1}{d_1}\tau_1\omega_1'||p_1\Phi|| - \frac{1}{d_2}\tau_2\omega_2'||p_2\Phi|| \\
&\geq ||\frac{\tau_1'}{d_1}p_1\Phi|| - ||\frac{\tau_2'}{d_2}p_2\Phi||,
\end{aligned}
\tag{6.56}
$$

where $\tau_1' = \tau_1\omega_1', \tau_2' = \tau_2\omega_2'$. Uisng the properties of Euclidean vector nomrs, we have,

$$
\begin{aligned}
||\frac{\tau_1'}{d_1}p_1\Phi - \frac{\tau_2'}{d_2}p_2\Phi|| &\geq ||\frac{\tau_1'}{d_1}p_1\Phi|| - ||\frac{\tau_2'}{d_2}p_2\Phi|| \\
||\frac{\tau_1'}{d_1}p_1\Phi - \frac{\tau_2'}{d_2}p_2\Phi|| &\leq ||\Phi||_F||\frac{\tau_1'}{d_1}p_1 - \frac{\tau_2'}{d_2}p_2||
\end{aligned}
\tag{6.57}
$$

Combining Eqn. 6.56 and 6.57, we have,

$$||\Phi||_F||\frac{\tau_1'}{d_1}p_1 - \frac{\tau_2'}{d_2}p_2|| \geq ||\frac{\tau_1'}{d_1}p_1\Phi|| - ||\frac{\tau_2'}{d_2}p_2\Phi|| \tag{6.58}$$

Therefore, $\exists\ \Phi'$ with $||\Phi'|| \leq 1$, such that,

$$||p_1^{Ah} - p_2^{Ah}|| \geq ||\Phi||_F||\frac{\tau_1'}{d_1}p_1 - \frac{\tau_2'}{d_2}p_2|| \geq ||\frac{\tau_1'}{d_1}p_1\Phi|| - ||\frac{\tau_2'}{d_2}p_2\Phi|| \tag{6.59}$$

WLOG, we can assume $\tau_1' \geq \tau_2'$, then we have,

$$||p_1^{Ah} - p_2^{Ah}|| \geq \tau_2'||\Phi||_F||\frac{p_1}{d_1} - \frac{p_2}{d_2}|| \tag{6.60}$$

If $d = \max\{d_1, d_2\}$, then we have,

$$\begin{aligned} ||p_1^{Ah} - p_2^{Ah}|| &\geq \tau_2'||\Phi||||\frac{p_1}{d_1} - \frac{p_2}{d_2}|| \\ &\geq \frac{\tau_2'||\Phi||_F}{d}||p_1 - p_2|| \end{aligned} \tag{6.61}$$

Therefore, from Lemma 1, we can express

$$\begin{aligned} d_{\mathbb{B}}(p_1^{Ah}, p_2^{Ah}) &\geq 2k||p_1^{Ah} - p_2^{Ah}||\ \forall k \in [0, \frac{\sqrt{1-k^2}}{k}] \\ &\geq 2k\frac{\tau_2'||\Phi||_F}{d}||p_1 - p_2|| \\ &= \frac{k'}{d}||\Phi||_F||p_1 - p_2||, \end{aligned} \tag{6.62}$$

where $k' = 2k\tau_2'$ with $k' \in (0, 2)$. The above inequality illustrates that the distance of the two points lying on the Poincarê Ball is greater than the distance of the same when lying on the Euclidean space under certain conditions. Furthermore, the inequality depends on the maximum degree of the pair of nodes in the graph.

If d increases, the distance on the Poincarê Ball also increases, which underscores the more distinctive positional encodings for the nodes having higher importance in the graph. □

Theorem 6.2 Consider a pair of nodes 1 and 2 of a connected graph $\mathcal{G}$ whose degrees are d_1 and d_2 respectively. Their initialized positional encodings are $p_1, p_2 \in \mathbb{R}^d$. The Euclidean

distance between them is estimated as $d_E(p_1, p_2)$. Suppose, p_1, p_2 are to be transformed y either HNN or HGCN with the underlying hyperbolic space as a n-dimensional Hyperboloid model $\mathbb{H}^n$ of unit radius and unit curvature, then we have the following:

1. **HNN** If the encodings are transformed by passing through an HNN of parameters Θ. The transformed encodings are respectively p_1^{hyp} and p_2^{hyp} whose distance is $d_{\mathbb{H}}(p_1^{\text{hyp}}, p_2^{\text{hyp}})$, then $\exists\, \Theta'$ such that $d_{\mathbb{H}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) \geq \frac{k'}{2}||\Theta'||_F d_E(p_1, p_2)$ for some $k' \in [1, \infty)$ and $||\Theta'||_F \leq 1$.

2. **HGCN** If the encodings are transformed by passing through an HGCN of parameters Φ. then there exists a Φ' with $||\Phi'||_F \leq 1$ such that $d_{\mathbb{B}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) \geq \frac{k'^{\prime}}{2d}||\Phi'||_F d_E(p_1, p_2)$ where $d = \max\{d_1, d_2\}$ for some $k' \in [1, \infty)$.

Proof. We will provide proof for each of the two parts.

HNN For any $v \in \mathcal{T}_0\mathbb{H}_1^n$ and $y \in \mathbb{H}_1^n$, then the exponential and logarithmic map of $\mathbb{H}_1^n$ at $x = 0$ with curvature $c = 1$ can be expressed as follows,

$$\begin{aligned} \exp_0^1(v) &= \sinh(||v||_{\mathcal{L}})\frac{v}{||v||_{\mathcal{L}}} \\ \log_0^1(y) &= \cosh^{-1}(1+\epsilon)\frac{y}{||y||_{\mathcal{L}}}, \end{aligned} \tag{6.63}$$

where ϵ is a significantly non-negative real quantity. If $y_0 = 0$ then $||y||_{\mathcal{L}}$ is equivalent to $||y||_2$. Consider two initialized positional encodings $p_1, p_2 \in \mathcal{T}_o\mathbb{H}_1^n$ lying in the tangent space of $x = 0$ which resembles locally to the Euclidean space. If an HNN transforms the encodings with trainable parameters Θ, then the transformed encodings can be represented as,

$$\hat{p}_1 = p_1\Theta \qquad\qquad \hat{p}_2 = p_2\Theta \tag{6.64}$$

After the transformation, the encodings are mapped to the Hyperboloid space by applying the exponential map.

$$\begin{aligned} p_1^{\text{hyp}} &= \exp(\hat{p}_1) = \sinh(||\hat{p}_1||)\frac{\hat{p}_1}{||\hat{p}_1||} \\ p_2^{\text{hyp}} &= \exp(\hat{p}_2) = \sinh(||\hat{p}_2||)\frac{\hat{p}_2}{||\hat{p}_2||} \end{aligned} \tag{6.65}$$

Assuming the scalar terms $\gamma_1 = \frac{\sinh(||\hat{p}_1||)}{||\hat{p}_1||}$ and $\gamma_2 = \frac{\sinh(||\hat{p}_2||)}{||\hat{p}_2||}$, we have $p_1^{\text{hyp}} = \gamma_1 p_1 \Theta$ and $p_2^{\text{hyp}} = \gamma_2 p_2 \Theta$. The distance between p_1^{hyp} and p_2^{hyp} is

$$d_{\mathbb{H}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) = \cosh^{-1}(-\langle p_1^{\text{hyp}}, p_2^{\text{hyp}} \rangle_{\mathcal{L}}) \tag{6.66}$$

From Lemma **??**, we have the following inequality

$$\begin{aligned} d_{\mathbb{H}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) &\geq \frac{k}{2} d_E(p_1^{\text{hyp}}, p_2^{\text{hyp}}) \\ &\qquad \forall\ d_E(p_1^{\text{hyp}}, p_2^{\text{hyp}}) \in [1, \sqrt[4]{\frac{1+k^2}{k^2}}] \\ &= \frac{k}{2} d_E(\gamma_1 \hat{p}_1, \gamma_2 \hat{p}_2) \\ &= \frac{k}{2} d_E(\gamma_1 p_1 \Theta, \gamma_2 p_2 \Theta) \\ &= \frac{k}{2} \|\gamma_1 p_1 \Theta - \gamma_2 p_2 \Theta\| \end{aligned} \tag{6.67}$$

Using the properties of Euclidean norm, we have,

$$\begin{aligned} \frac{k}{2} \|\gamma_1 p_1 \Theta - \gamma_2 p_2 \Theta\| &\leq \frac{k}{2} \|\Theta\|_F \|\gamma_1 p_1 - \gamma_2 p_2\| \\ \frac{k}{2} \|\gamma_1 p_1 \Theta - \gamma_2 p_2 \Theta\| &\geq \frac{k}{2} (||\gamma_1 p_1 \Theta|| - ||\gamma_2 p_2 \Theta||) \end{aligned} \tag{6.68}$$

Therefore, $\exists\ \Theta'$ such that $\|\Theta'\|_F \leq 1$ with satisfies the following

$$d_{\mathbb{H}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) \geq \frac{k}{2} \|\Theta'\|_F \|\gamma_1 p_1 - \gamma_2 p_2\| \geq \frac{k}{2} \|\gamma_1 p_1 \Theta - \gamma_2 p_2 \Theta\| \tag{6.69}$$

WLOG, we can assume $\gamma_1 \geq \gamma_2$, we have,

$$\begin{aligned} d_{\mathbb{H}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) &\geq \frac{k\gamma_2}{2} \|\Theta'\|_F \|p_1 - p_2\| \\ &= \frac{k'}{2} \|p_1 - p_2\|, \end{aligned} \tag{6.70}$$

where $k' = k\gamma_2$

HGCN If we want to transform the positional encodings with HGCN with trainable parameters Φ, then we need to first feed the encodings to the dense layer. Now applying Φ, the

transformed encodings can be expressed as,

$$\hat{p}_1 = p_1\Phi \qquad\qquad \hat{p}_2 = p_2\Phi \tag{6.71}$$

The adjacency matrix is applied to the transformed encodings which result in $p_1^A = \frac{1}{d_1}\sum_{j\in N(1)\cup\hat{p}_1}\hat{p}_j$ and $p_2^A = \frac{1}{d_2}\sum_{j\in N(2)\cup\hat{p}_2}\hat{p}_j$ where d_1, d_2 are the degrees for node 1 and 2 respectively with $\hat{p}_j$ are the corresponding neighbors' features. Now $\exists\ C_1, C_2$ such that,

$$||p_1^A|| \geq \frac{1}{d_1}C_1||\hat{p}_1|| \qquad\qquad ||p_2^A|| \geq \frac{1}{d_2}C_2||\hat{p}_2|| \tag{6.72}$$

Now the updated positional encodings are reverted to the Hyperboloid space by applying the following exponential map

$$\begin{aligned} p_1^{\text{hyp}} &= \exp(p_1^A) = \sinh\big(||p_1^A||\big)\frac{p_1^A}{||p_1^A||} \\ p_2^{\text{hyp}} &= \exp(p_2^A) = \sinh\big(||p_2^A||\big)\frac{p_2^A}{||p_2^A||} \end{aligned} \tag{6.73}$$

Replacing the scalar terms as $\alpha_1 = \frac{\sinh\left(||p_1^A||\right)}{||p_1^A||}$ and $\alpha_2 = \frac{\sinh\left(||p_2^A||\right)}{||p_2^A||}$, we have $p_1^{\text{hyp}} = \alpha_1 p_1^A$ and $p_2^{\text{hyp}} = \alpha_2 p_2^A$. Therefore, the distance between p_1^{hyp} and p_2^{hyp} is

$$d_{\mathbb{B}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) = \cosh^{-1}(-\langle p_1^{\text{hyp}}, p_2^{\text{hyp}}\rangle_{\mathcal{L}}). \tag{6.74}$$

From Theorem **??**, we have the following inequality

$$\begin{aligned} d_{\mathbb{H}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) &\geq \frac{k}{2} d_E(p_1^{\text{hyp}}, p_2^{\text{hyp}}) \\ &\qquad \forall\ d_E(p_1^{\text{hyp}}, p_2^{\text{hyp}}) \in [1, \sqrt[4]{\frac{1+k^2}{k^2}}] \\ &= \frac{k}{2} d_E(\alpha_1 p_1^A, \alpha_2 p_2^A) \\ &= \frac{k}{2} \|\alpha_1 p_1^A - \alpha_2 p_2^A\| \\ &\geq \frac{k}{2} (\alpha_1 ||p_1^A|| - \alpha_2 ||p_2^A||) \\ &\geq \frac{k}{2} (\frac{\alpha_1 C_1}{d_1} ||\hat{p_1}|| - \frac{\alpha_2 C_2}{d_2} ||\hat{p_2}||) \\ &= \frac{k}{2} \left(\frac{\alpha_1 C_1}{d_1} ||p_1 \Phi|| - \frac{\alpha_2 C_2}{d_2} ||p_2 \Phi|| \right) \end{aligned} \tag{6.75}$$

WLOG, we can assume $\alpha_1 C_1 \geq \alpha_2 C_2$, then

$$d_{\mathbb{H}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) \geq \frac{k \alpha_2 C_2}{2} \left(||\frac{p_1 \Phi}{d_1}|| - ||\frac{p_2 \Phi}{d_2}|| \right) \tag{6.76}$$

Using the properties of the Euclidean vector norm we have,

$$\begin{aligned} \|\frac{p_1 \Phi}{d_1} - \frac{p_2 \Phi}{d_2}\| &\leq \|\Phi\|_F \|\frac{p_1}{d_1} - \frac{p_2}{d_2}\| \\ \|\frac{p_1 \Phi}{d_1} - \frac{p_2 \Phi}{d_2}\| &\geq ||\frac{p_1 \Phi}{d_1}|| - ||\frac{p_2 \Phi}{d_2}|| \end{aligned} \tag{6.77}$$

Therefore, $\exists\ \Phi'$ with $\|\Phi'\|_F \leq 1$ such that

$$\begin{aligned} d_{\mathbb{H}}(p_1^{\text{hyp}}, p_2^{\text{hyp}}) &\geq \frac{k \alpha_2 C_2}{2} \|\Phi'\|_F \|\frac{p_1}{d_1} - \frac{p_2}{d_2}\| \\ &\geq \frac{k \alpha_2 C_2}{2d} \|\Phi'\|_F \|p_1 - p_2\| \\ &\qquad \text{where } d = \max\{d_1, d_2\} \\ &= \frac{k'}{2d} \|\Phi'\|_F \|p_1 - p_2\| \end{aligned} \tag{6.78}$$

Therefore, we have shown that an HGCN architecture exists such that the distance between two positional encodings increases compared to the same in Euclidean space. □

6.10 Appendix B

6.10.1 Details of the Datasets

The details of the datasets involved in the experiments are provided as follows,

Table 6.5: Details of the datasets from (Dwivedi et al., 2020) and (Hu et al., 2020a)

Name	#Graphs	Avg # Nodes	Avg # Edges	Task	Directed	Metric
PATTERN	14000	118.9	3,039.3	binary classif.	No	Accuracy
CLUSTER	12000	117.2	2,150.9	6-class classif.	No	Accuracy
MNIST	70,000	70.6	564.5	10-class classification	Yes	Accuracy
CIFAR10	60,000	117.6	941.1	10-class classification	Yes	Accuracy
ogbg-molhiv	41127	25.5	27.5	binary classif.	No	AUROC
ogbg-ppa	158,100	243.4	2,266.1	37-task classic.	No	Accuracy
ogbg-molpcba	437,929	26.0	28.1	128-task classif.	No	Avg. Precision
ogbg-code2	452,741	125.2	124.2	5 token sequence	No	F1 score

Table 6.6: Details of the Co-author and Co-purchase datasets

Dataset	Nodes	Edges	Features	Classes
Amazon photo	13752	491722	10	767
Amazon Computers	7650	238162	8	745
Coauthor CS	18333	81894	15	6805
Coauthor Physics	34493	495924	5	8415

PATTERN and CLUSTER are molecular datasets generated from Stochastic Block Model (Abbe, 2018). The prediction task here is an inductive node-level classification. In PATTERN the task is to identify which nodes in a graph belong to one of 100 different sub-graph patterns which were randomly generated with different SBM parameters. In CLUSTER, every graph is composed of 6 SBM-generated clusters, each drawn from the same distribution, with only a single node per cluster containing a unique cluster ID. Our target is predict the cluster ID of the nodes.

MNIST and CIFAR10 are generated from image classification datasets of similar names. Superpixel datasets are constructed by an 8 nearest-neighbor graph of SLIC superpixels for each image. The 10-class classification tasks and standard dataset splits follow the original image classification datasets, i.e., for MNIST 55K/5K/10K and CIFAR10 45K/5K/10K train/validation/test graphs.

ogbg-molhiv and ogbg-molpcba are molecular property prediction datasets designed by OGB from MoleculeNet. The molecules are represented by the nodes (atoms) and edges (bonds). The node and edge features are generated from a similar source which represents chemo-physical properties. The prediction task of ogbg-molhiv is the binary classification of the molecule's suitability for combating the replication of HIV. On the other hand, ogbg-molpcba, derived from PubChem BioAssay, is tasked to predict the results of 128 bioassays in multi-task binary classification.
setting.
ogbg-ppa (CC-0 license) consists of protein-protein association (PPA) networks derived from 1581 species categorized into 37 taxonomic groups. Nodes represent the proteins and edges are poised to encode the normalized level of 7 different associations between that pair of proteins. The target task is to classify to one of the 37 groups of the PPA network.

ogbg-code2 (MIT License) is comprised of abstract syntax trees (ASTs) constructed from the source code of functions written in Python. The task is to predict the first 5 subtokens of the original function's name. A small number of these ASTs are much larger than the average size in the dataset. Therefore, we truncated ASTs with over 1000 nodes and kept the first 1000 nodes according to their depth in the AST. The processing only impacted 2521 (0.5%) graphs in the entire dataset.

Co-authorship datasets Coauthor CS and Coauthor Physics are two co-authorship networks (Shchur et al., 2018). Nodes represent authors and edges exist between them if they co-authored a paper. The features of the nodes represent the keywords related to the paper of author. The label of each node denotes the field of study of the corresponding author.
Co-purchase datasets Amazon Computers and Amazon Photo are two co-purchase networks (Shchur et al., 2018) where each node denotes products, and an edge exists if two products are bought frequently. Node features denote the bag-of-words representation of the product reviews. Node labels indicate the product category.

6.10.2 Computational Resources

We run the experiments on the datasets with the standard train/validation/test splits. The mean and standard deviations are reported after 10 runs on multiple random seeds for each dataset. All experiments are done on a single GPU GeForce RTX 3090 with 24GB memory capacity.

6.10.3 Hyperparameter Details

In this section, we will describe the hyperparameters of every dataset employed for the experimentation. Refer to Tables 6.7, 6.8, 6.9, 6.10, and 6.14 for the hyperparameters of category-wise positional encodings for MNIST, CIFAR10, PATTERN, CLUSTER, ogbg-molhiv, ogbg-ppa, ogbg-molpcba, and ogbg-code2 datasets, respectively. The hyperparameters are adjusted from the initial setting, which are inspired by SAN (Kreuzer et al., 2021), GraphGPS (Rampášek et al., 2022), SAT (Chen et al., 2022a), and GraphGPS (Rampášek et al., 2022). The model parameters are optimized by Adam (Kingma and Ba, 2014) optimizer with the default settings. The learning rate is adjusted after the number of "patience" epochs.

The hyperparameters of the Co-author and Co-purchase datasets are provided in Table 6.15. The dimension of the positional vector is 128 when the eigenvectors of the Laplacian matrix for every network depth are initialized as PEs. The dimension of PE is fixed at 8 when PEs are initialized with RWPE.

6.10.4 More Results on Ablation Study

We conduct ablation studies on ogbg-molpcba and ogbg-code2 datasets from OGB. Varying different modules of HyPE-GT, we generate a diverse set of learnable hyperbolic positional encodings. Refer to Figure 6.7 for detailed visualization. The variation in the performances is recorded across 8 categories of PEs. The experiment underscores the utility of generating a diverse set of PEs for solving downstream tasks.

6.10.5 Comparative Study on Number of Parameters

We perform a comparative study on the parameters of the existing Graph Transformers like GraphTransformer (Dwivedi and Bresson, 2020), SAN (Kreuzer et al., 2021), SAT (Chen

Table 6.7: Hyperparameters for MNIST dataset for every category of PE generated in the experiments.

Hyperparameters / PE Category	MNIST							
	1	2	3	4	5	6	7	8
# HyPE-GT Layers	4							
# Head	8							
Hidden Dim	80							
Curvature	1.0							
Activation	ReLU							
# PE Layers	2							
Dropout	0.0							
Layernorm	False							
Batchnorm	True							
PE Dim	6							
Graph Pooling	Sum							
Batch size	128							
Init LR	0.001							
Epochs	1000							
Patience	10							
Weight Decay	0.0							

Table 6.8: Hyperparameters for CIFAR10 dataset for every category of PE generated in the experiments.

Hyperparameters / PE Category	CIFAR10							
	1	2	3	4	5	6	7	8
# HyPE-GT Layers	4							
# Head	8							
Hidden Dim	80							
Curvature	1.0							
Activation	ReLU							
# PE Layers	2							
Dropout	0.0							
Layernorm	False							
Batchnorm	True							
PE Dim	16							
Graph Pooling	Mean							
Batch size	128							
Init LR	0.001							
Epochs	1000							
Patience	10							
Weight Decay	0.0							

Table 6.9: Hyperparameters for PATTERN dataset for every category of PE generated in the experiments.

Hyperparameters / PE Category	PATTERN							
	1	2	3	4	5	6	7	8
# HyPE-GT Layers	10				10			
# Head	8				8			
Hidden Dim	80				80			
Curvature	1.0				1.0			
Activation	ReLU				ReLU			
# PE Layers	1				1			
Dropout	0.0				0.0			
Layernorm	False				False			
Batchnorm	True				True			
PE Dim	6				2			
Graph Pooling	Mean				Mean			
Batch size	26				26			
Init LR	0.0005				0.0003			
Epochs	1000				1000			
Patience	10				10			
Weight Decay	0.0				0.0			

Table 6.10: Hyperparameters for CLUSTER dataset for every category of PE generated in the experiments.

Hyperparameters / PE Category	CLUSTER							
	1	2	3	4	5	6	7	8
# HyPE-GT Layers	10				10			
# Head	8				8			
Hidden Dim	80				80			
Curvature	1.0				1.0			
Activation	ReLU				ReLU			
# PE Layers	4				2			
Dropout	0.0				0.0			
Layernorm	False				False			
Batchnorm	True				True			
PE Dim	6				16			
Graph Pooling	Mean				Mean			
Batch size	32				32			
Init LR	0.0005				0.0003			
Epochs	1000				1000			
Patience	10				10			
Weight Decay	0.0				0.0			

et al., 2022a), Graphormer (Ying et al., 2021b), EGT (Hussain et al., 2022), and GraphGPS (Rampášek et al., 2022) with our proposed method HyPE-GT. Refer to Table **??** for detailed information regarding the number of model parameters. The number of parameters from both variants is equal because the variants are structurally identical, but they differ only in the way PEs are incorporated. For PATTERN and CLUSTER, both our variants have several parameters comparable to GraphTransformer, SAN, and EGT. But as GraphGPS is

Table 6.11: Hyperparameters for ogbg-molhiv dataset for every category of PE generated in the experiments.

Hyperparameters / PE Category	ogbg-molhiv							
	1	2	3	4	5	6	7	8
# HyPE-GT Layers	10							
# Head	4							
Hidden Dim	64							
Curvature	1.0							
Activation	ReLU							
# PE Layers	2							
Dropout	0.01							
Layernorm	False							
Batchnorm	True							
PE Dim	32							
Graph Pooling	Max							
Batch size	64							
Init LR	0.0001							
Epochs	1000							
Patience	20							
Weight Decay	0.0							

Table 6.12: Hyperparameters for ogbg-ppa dataset for every category of PE generated in the experiments.

Hyperparameters / PE Category	ogbg-ppa							
	1	2	3	4	5	6	7	8
# HyPE-GT Layers	2							
# Head	2							
Hidden Dim	16							
Curvature	1.0							
Activation	ReLU							
# PE Layers	2							
Dropout	0.0							
Layernorm	False							
Batchnorm	True							
PE Dim	8							
Graph Pooling	Sum							
Batch size	16							
Init LR	0.0003							
Epochs	1000							
Patience	15							
Weight Decay	0.0							

Table 6.13: Hyperparameters for ogbg-molpcba dataset for every category of PE generated in the experiments.

Hyperparameters / PE Category	ogbg-molpcba							
	1	2	3	4	5	6	7	8
# HyPE-GT Layers	5							
# Head	4							
Hidden Dim	304							
Curvature	1.0							
Activation	ReLU							
# PE Layers	2							
Dropout	0.2							
Layernorm	False							
Batchnorm	True							
PE Dim	32							
Graph Pooling	Mean							
Batch size	512							
Init LR	0.0005							
Epochs	1000							
Patience	20							
Weight Decay	0.0							

Table 6.14: Hyperparameters for ogbg-code2 dataset for every category of PE generated in the experiments.

Hyperparameters / PE Category	ogbg-code2							
	1	2	3	4	5	6	7	8
# HyPE-GT Layers	4							
# Head	4							
Hidden Dim	16							
Curvature	1.0							
Activation	ReLU							
# PE Layers	1							
Dropout	0.2							
Layernorm	False							
Batchnorm	True							
PE Dim	8							
Graph Pooling	Mean							
Batch size	32							
Init LR	0.0001							
Epochs	1000							
Patience	20							
Weight Decay	0.0							

a linearized Transformer architecture. Therefore, it is desirable to have a lower number of parameters. However, the SAT has a much higher number of parameters. On the other side, our framework produces a higher number of parameters compared to EGT and GraphGPS (as the rest of the methods do not report the numbers). Still, HyPE-GT outperforms all methods. Lastly, our framework produces the lowest number of parameters compared to all SOTA approaches, and also achieves the best performance on the dataset. As ogbg-molhiv

Table 6.15: Hyperparameters for Co-author and Co-purchase datasets

Hyperparameters	Amazon photo	Amazon computers	Coauthor CS	Coauthor Physics
Learning rate	0.01	0.01	0.01	0.01
PE Dim	128/8	128/8	128/8	128/8
Hidden Dim	64	64	64	64
#PE Layers	2	2	2	2
Activation	ReLU	ReLU	ReLU	ReLU
Dropout	0.50	0.50	0.20	0.20
Curvature	1.0	1.0	1.0	1.0
weight decay	0.0005	0.0005	0.0005	0.0005
Training Epochs	500	500	500	500

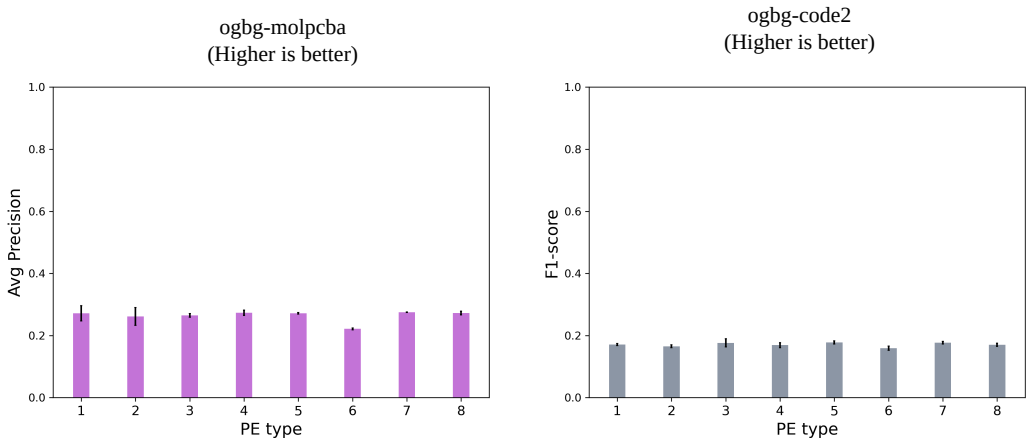


Figure 6.7: The performance of HyPE-GT on ogbg-molpcba and ogbg-code2 for 8 different categories of PEs is presented.

is one of the large-scale graphs, the performance of the framework is evidence of the efficacy of the hyperbolic positional encodings.

Table 6.16: A comparative study on the number of parameters of HyPE-GT with the other existing Graph Transformers.

Method / Data	Init PE	Hyperbolic Manifold	Hyperbolic NN	PATTERN	CLUSTER	MNIST	CIFAR10	ogbg-molhiv
GraphTransformer (Dwivedi and Bresson, 2020)				523146	522742	-	-	-
SAN (Kreuzer et al., 2021)				507,202	519,186	-	-	528265
Graphormer (Ying et al., 2021b)				-	-	-	-	47.0M
SAT (Chen et al., 2022a)				825,986	741,990	-	-	-
EGT (Hussain et al., 2022)				500000	500000	100000	100000	110.8M
GraphGPS (Rampášek et al., 2022)				337201	502054	115,394	112,726	558625
HyPE-GT (ours)	LapPE	Hyperboloid	HGCN	524022	524426	369390	371150	389441
			HNN	523382	524426	369390	369550	390465
		Poincare Ball	HGCN	523382	524426	369390	369550	388929
			HNN	523382	524666	369390	369550	388929
	RWPE	Hyperboloid	HGCN	523142	524666	368830	368990	390465
			HNN	523142	524666	368830	368990	390465
		Poincare Ball	HGCN	523142	524666	368830	369790	390465
			HNN	524426	524666	368830	369790	390465

Chapter 7

Future Works

Summary

In this chapter, we focus on evaluating the various contributions made by us in the previous chapters of this thesis. Specifically, we provide a summary highlighting the key attributes of our work and discuss their importance in further enriching the active research efforts focusing on deep learning on graphs. Moreover, we envision the future possibilities of our contribution, assuming they will pave the way for further research opportunities. Finally, as a concluding remark, we list the various open problems related to graph representation learning.

7.1 Future Possibilities

In this section, we describe how our contributions in this thesis can be further explored to open up new research opportunities.

- In chapter 2, we propose a novel architecture, RPE-GNN, that prevents the effects of oversmoothing. Our approach dynamically samples paths and estimates oath features, and subsequently, path features are employed to aggregate multi-hop features. As an immediate future work, we can enhance the prowess of RPE-GNN to make it applicable for large-scale graphs (Hu et al., 2020a). In our work, the path sampling is random. Thus, as a future work, an effective sampling strategy can be designed that may be suitable for heterophilic networks (Pei et al., 2020; Zhu et al., 2020).

- In the chapter 3, we propose a label-guided graph rewiring pre-processing framework, LGR-GNN, to tackle heterophilic networks. In this work, the key component is the two-

stage rewiring strategy. The proposed framework is specifically designed for undirected graphs. Therefore, the approach can be extended to directed graphs. One work, Dir-GNN (Rossi et al., 2024b), processes directed heterophilic graphs but does not pursue rewiring-based approaches. The design of the rewiring strategy for directed and large-scale graphs can be a potential avenue of research.

- In the chapter 4, we investigated the impacts on graph spectrum with the addition of self-loops and parallel edges. We have also shown the increase (decrease) in the eigenvalues of the normalized graph Laplacian when the number of parallel edges (self-loops) gradually increases. In our extensive experiments on 17 heterophilic graphs, we observed the performance trends when the number of self-loops or parallel edges varies. Based on these performance trends, the shape of the spectrum as well as the parity of the frequencies can be identified. Our strategy avert the computationally expensive eigen-decomposition algorithm to gain insights into the graph spectrum. Therefore, designing more strategies hinged on the GNNs to substitute traditional costly algorithms like matrix rank computation for large dimensions can be an immediate extension. GNNs can further be employed to learn various graph algorithms, which can be connected with Neural Algorithmic Reasoning (NAR) (Veličković and Blundell, 2021).

- In the chapter 5, we propose a model-agnostic solution, CAMP, to address oversquashing in the GNNs. Our primary target is to process and store the aggregated messages in the fixed-dimensional feature vectors at different time stamps. This approach leads us to design an asynchronous message passing framework. CAMP considers high centrality nodes as primary sources for information bottlenecks and allows those nodes to propagate information to the lower centrality nodes at different time-stamps. As future work, we can design a strategy that learn when messages are to be processed and updated asynchronously. Furthermore, the proposed method is mostly appropriate for small-scale molecular graphs where centrality estimation is computationally feasible. Studying the effects of oversquashing in the large-scale graphs can be a potential research direction. Additionally, we can design novel techniques to find bottleneck nodes that will be compatible with large-scale graphs.

- In chapter 6, we propose a novel and learnable positional encoding HyPE that operates

in the hyperbolic space for the Graph Transformer. HyPE can be flexibly incorporated into all variants of GTs. HyPE excels in capturing hierarchical patterns embedded in the molecular graphs. As a future work, we can pursue the design of positional encodings for geometric graphs. We have also demonstrated that hyperbolic positional encodings improve the performance of multi-layered MPNNs, preventing oversmoothing. Thus, designing effective positional encodings to mitigate oversmoothing for large graphs can be an important research avenue.

7.2 Open problems

In this section, we discuss some open problems related to graph representation learning, which are likely to introduce new avenues of research.

- The studies in the thesis mostly revolve around static graphs. The issues like heterophily, oversmoothing, and oversquashing can be further extended to the dynamic or time-evolving graphs. Sm recent works in Temporal Graph Benchmark (TGB) (Shirzadkhani et al., 2024), extend the existing GNNs to be compatible with the temporal graphs.

- The graph transformers suffer from the quadratic time complexity for computing the self-attention matrix. GTs also suffer from computing learnable positional encodings for the input graph. As the future direction, the immediate extension can be designing efficient sparse attention modules to culminate in a linear time complexity of the attention module. Additionally, the future work can propose a more advanced graph transformer that refrains from estimating the overhead of positional encodings.

- Designing foundation models on graphs emerges as a profound challenge due to insufficient graph datasets. The training of the prevailing LLMs enjoys the advantages of abundant textual data. In a striking contrast, such a volume of graph data is not available for training. Thus, to align with the current trends, we may attempt to develop the Graph Foundation Model (GFM) by generating adequate graph data.

- Solving combinatorial problems may face an exponential order of time complexity, like the Traveling Salesman Problem (TSP) or detecting a Hamiltonian cycle. Very few

works like (Cappart et al., 2023; Jin et al., 2024) explore the capability of GNNs to solve NP-hard problems. Thus, exploring future possibilities of addressing NP-hard problems by applying GNNs can be an effective area of future work.

List of Publications

K. Bose and S. Das. Can graph neural networks go deeper without over-smoothing? yes, with a randomized path exploration! *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(5):1595–1604, 2023. doi: 10.1109/TETCI.2023.3249255.

K. Bose and S. Das. Learning from heterophilic graphs: A spectral theory perspective on the impact of self-loops and parallel edges. *IEEE Transactions on Artificial Intelligence*, pages 1–15, 2026. doi: 10.1109/TAI.2026.3664026.

K. Bose and S. Das. Asynchronous message passing for addressing oversquashing in graph neural networks. *Under review*, January 2026.

K. Bose and S. Das. Hype-gt: where graph transformers meet hyperbolic positional encodings. *Under review*, March 2026.

K. Bose, S. Banerjee, and S. Das. Can graph neural networks tackle heterophily? yes, with a label-guided graph rewiring approach! *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15, 2025. doi: 10.1109/TNNLS.2025.3565108.

References

E. Abbe. Community detection and stochastic block models: Recent developments. *Journal of Machine Learning Research*, 18(177):1–86, 2018. URL http://jmlr.org/papers/v18/16-480.html. 181

R. Abboud, R. Dimitrov, and I. I. Ceylan. Shortest path networks for graph property prediction. In *Learning on Graphs Conference*, pages 5–1. PMLR, 2022.

S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. Ver Steeg, and A. Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019. 9

N. Alon and V. D. Milman. Eigenvalues, expanders and superconcentrators. In *25th Annual Symposium onFoundations of Computer Science, 1984.*, pages 320–322. IEEE, 1984.

U. Alon and E. Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020. 11, 17, 60, 118, 121, 141

A. Arnaiz-Rodríguez, A. Begga, F. Escolano, and N. Oliver. Diffwire: Inductive graph rewiring via the lovász bound, 2022. *URL https://arxiv. org/abs/2206.07369*, 2. 13, 118

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 14, 146, 158

P. K. Banerjee, K. Karhadkar, Y. G. Wang, U. Alon, and G. Montúfar. Oversquashing in gnns through the lens of information contraction and graph expansion. In *2022 58th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1–8. IEEE, 2022. 12

F. Barbero, A. Velingker, A. Saberi, M. Bronstein, and F. Di Giovanni. Locality-aware graph-rewiring in gnns. *arXiv preprint arXiv:2310.01668*, 2023. 13, 118, 128

D. Beaini, S. Passaro, V. Létourneau, W. Hamilton, G. Corso, and P. Liò. Directional graph networks. In *International Conference on Machine Learning*, pages 748–758. PMLR, 2021.

W. Bi, L. Du, Q. Fu, Y. Wang, S. Han, and D. Zhang. Make heterophily graphs better fit gnn: A graph rewiring approach. *arXiv preprint arXiv:2209.08264*, 2022a.

W. Bi, L. Du, Q. Fu, Y. Wang, S. Han, and D. Zhang. Make heterophily graphs better fit gnn: A graph rewiring approach. *arXiv preprint arXiv:2209.08264*, 2022b. 47, 60

F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi. Graph neural networks with convolutional arma filters. *IEEE transactions on pattern analysis and machine intelligence*, 44(7): 3496–3507, 2021. 81

M. Black, Z. Wan, A. Nayyeri, and Y. Wang. Understanding oversquashing in gnns through the lens of effective resistance. In *International Conference on Machine Learning*, pages 2528–2547. PMLR, 2023a. 12, 17, 118, 133

M. Black, Z. Wan, A. Nayyeri, and Y. Wang. Understanding oversquashing in gnns through the lens of effective resistance. In *International Conference on Machine Learning*, pages 2528–2547. PMLR, 2023b.

D. Bo, X. Wang, C. Shi, and H. Shen. Beyond low-frequency information in graph convolutional networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 3950–3957, 2021. 8, 81

D. Bo, C. Shi, L. Wang, and R. Liao. Specformer: Spectral graph neural networks meet transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=0pdSt3oyJa1. 155

C. Bodnar, F. Di Giovanni, B. Chamberlain, P. Lio, and M. Bronstein. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns. *Advances in Neural Information Processing Systems*, 35:18527–18541, 2022. 6

A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann. Netgan: Generating graphs via random walks. In *International Conference on Machine Learning*, pages 610–619. PMLR, 2018. 25

S. P. Borgatti and D. S. Halgin. Analyzing affiliation networks. *The Sage handbook of social network analysis*, 1:417–433, 2011. 123

G. Bouritsas, F. Frasca, S. Zafeiriou, and M. M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668, 2022. 155

X. Bresson and T. Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017. 151

S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=F72ximsx7C1.

M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. 2, 24

C. Cai and Y. Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020a. 7

C. Cai and Y. Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020b. 118, 121

Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023. 190

I. Chami, Z. Ying, C. Ré, and J. Leskovec. Hyperbolic graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019. 144

S. Chanpuriya and C. Musco. Simplified graph convolution with heterophily. *arXiv preprint arXiv:2202.04139*, 2022. 8

D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445, 2020a. 6, 26

D. Chen, L. O'Bray, and K. Borgwardt. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, pages 3469–3489. PMLR, 2022a. 15, 17, 141, 144, 151, 183, 186

G. Chen and J. Zhang. Preventing over-smoothing for hypergraph neural networks. *arXiv preprint arXiv:2203.17159*, 2022. 5

H. Chen, W. Huang, Y. Xu, F. Sun, and Z. Li. Graph unfolding networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1981–1984, 2020b.

H. Chen, Z. Deng, Y. Xu, and Z. Li. Non-recursive graph convolutional networks. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3205–3209. IEEE, 2021. 5

J. Chen, K. Gao, G. Li, and K. He. Nagphormer: A tokenized graph transformer for node classification in large graphs. *arXiv preprint arXiv:2206.04910*, 2022b.

J. Chen, S. Chen, J. Gao, Z. Huang, J. Zhang, and J. Pu. Exploiting neighbor effect: Conv-agnostic gnn framework for graphs with heterophily. *IEEE Transactions on Neural Networks and Learning Systems*, 2023a. 11

J. Chen, T. Liao, C. Chen, and Z. Zheng. Improving message-passing gnns by asynchronous aggregation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 228–238, 2024a. 12

J. Chen, C. Liu, K. Gao, G. Li, and K. He. Nagphormer+: A tokenized graph transformer with neighborhood augmentation for node classification in large graphs. *IEEE Transactions on Big Data*, 2024b.

J. Chen, H. Liu, J. Hopcroft, and K. He. Leveraging contrastive learning for enhanced node representations in tokenized graph transformers. *Advances in Neural Information Processing Systems*, 37:85824–85845, 2024c.

M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pages 1725–1735. PMLR, 2020c. 3, 6, 8, 17, 32, 39, 50, 60, 118, 160

Z. Chen, F. Chen, L. Zhang, T. Ji, K. Fu, L. Zhao, F. Chen, L. Wu, C. Aggarwal, and C.-T. Lu. Bridging the gap between spatial and spectral domains: A unified framework for graph neural networks. *ACM Computing Surveys*, 56(5):1–42, 2023b. 80

Z. Chen, Z. Lin, S. Chen, Y. Polyanskiy, and P. Rigollet. Residual connections provably mitigate oversmoothing in graph neural networks. *arXiv e-prints*, pages arXiv–2501, 2025. 7

E. Chien, J. Peng, P. Li, and O. Milenkovic. Adaptive universal generalized pagerank graph neural network. *arXiv preprint arXiv:2006.07988*, 2020. 5, 8, 47, 60

J. Choi, S. Park, H. Wi, S.-B. Cho, and N. Park. PANDA: Expanded width-aware message passing beyond rewiring. In *Forty-first International Conference on Machine Learning*, 2024a. URL https://openreview.net/forum?id=J1NIXxiDbu. 13, 17, 118

Y. Y. Choi, S. W. Park, M. Lee, and Y. Woo. Topology-informed graph transformer. *arXiv preprint arXiv:2402.02005*, 2024b. 151

F. R. Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997. 136

B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of operations research*, 153:235–256, 2007.

G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020. 3, 118, 151

E. Dai, S. Zhou, Z. Guo, and S. Wang. Label-wise graph convolutional network for heterophilic graphs. In *Learning on Graphs Conference*, pages 26–1. PMLR, 2022. 10

A. Deac, M. Lackenby, and P. Veličković. Expander graph propagation. In *Learning on Graphs Conference*, pages 38–1. PMLR, 2022. 12

M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016. 80

F. Di Giovanni, J. Rowbottom, B. P. Chamberlain, T. Markovich, and M. M. Bronstein. Graph neural networks as gradient flows: understanding graph convolutions via energy. *arXiv preprint arXiv:2206.10991*, 2022a.

F. Di Giovanni, J. Rowbottom, B. P. Chamberlain, T. Markovich, and M. M. Bronstein. Graph neural networks as gradient flows: understanding graph convolutions via energy. *arXiv preprint arXiv:2206.10991*, 2022b. 6

F. Di Giovanni, L. Giusti, F. Barbero, G. Luise, P. Lio, and M. M. Bronstein. On over-squashing in message passing neural networks: The impact of width, depth, and topology. In *International Conference on Machine Learning*, pages 7865–7885. PMLR, 2023a.

F. Di Giovanni, L. Giusti, F. Barbero, G. Luise, P. Lio, and M. M. Bronstein. On over-squashing in message passing neural networks: The impact of width, depth, and topology. In *International Conference on Machine Learning*, pages 7865–7885. PMLR, 2023b. 11, 13, 121, 133

F. Di Giovanni, T. K. Rusch, M. M. Bronstein, A. Deac, M. Lackenby, S. Mishra, and P. Veličković. How does over-squashing affect the power of gnns? *arXiv preprint arXiv:2306.03589*, 2023c. 13

Y. Ding, A. Orvieto, B. He, and T. Hofmann. Recurrent distance-encoding neural networks for graph representation learning, 2024. URL https://openreview.net/forum?id=lNIj5FdXsC. 151

Y. Dong, K. Ding, B. Jalaian, S. Ji, and J. Li. Adagnn: Graph neural networks with adaptive frequency response filter. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 392–401, 2021. 81

D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *arXiv preprint arXiv:1509.09292*, 2015. 25

V. P. Dwivedi and X. Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020. 13, 15, 17, 118, 143, 145, 151, 155, 183, 186

V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. 2020. xvi, 118, 151, 156, 165, 181

V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021. 15, 17, 141, 149

V. P. Dwivedi, L. Rampášek, M. Galkin, A. Parviz, G. Wolf, A. T. Luu, and D. Beaini. Long range graph benchmark. *Advances in Neural Information Processing Systems*, 35: 22326–22340, 2022. 3

M. Eliasof, F. Frasca, B. Bevilacqua, E. Treister, G. Chechik, and H. Maron. Graph positional encoding via random feature propagation. In *International Conference on Machine Learning*, pages 9202–9223. PMLR, 2023. 16

B. Epping, A. René, M. Helias, and M. T. Schaub. Graph neural networks do not always oversmooth. *arXiv preprint arXiv:2406.02269*, 2024. 7

L. Faber and R. Wattenhofer. Asynchronous message passing: A new framework for learning in graphs.

B. Fatemi, L. El Asri, and S. M. Kazemi. Slaps: Self-supervision improves structure learning for graph neural networks. *Advances in Neural Information Processing Systems*, 34:22667–22681, 2021.

L. Fesser and M. Weber. Mitigating over-smoothing and over-squashing using augmentations of forman-ricci curvature. In *Learning on Graphs Conference*, pages 19–1. PMLR, 2024. 12

M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 35

B. Finkelshtein, X. Huang, M. Bronstein, and u. u. Ceylan. Cooperative graph neural networks. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024. 11

L. Franceschi, M. Niepert, M. Pontil, and X. He. Learning discrete structures for graph neural networks. In *International conference on machine learning*, pages 1972–1982. PMLR, 2019.

F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020. 39, 47

L. Freeman. A set of measures of centrality based on betweenness.(1977). *Sociometry*, 40 (35-41), 1977. 123

L. C. Freeman et al. Centrality in social networks: Conceptual clarification. *Social network: critical concepts in sociology. Londres: Routledge*, 1:238–263, 2002. 123

R. B. Gabrielsson, M. Yurochkin, and J. Solomon. Rewiring with positional encodings for graph neural networks. *Transactions on Machine Learning Research*, 2023.

O. Ganea, G. Bécigneul, and T. Hofmann. Hyperbolic neural networks. *Advances in neural information processing systems*, 31, 2018. 144, 151

Z. Gao, D. Dong, C. Tan, J. Xia, B. Hu, and S. Z. Li. A graph is worth k words: Euclideanizing graph using pure transformer. *arXiv preprint arXiv:2402.02464*, 2024.

J. Gasteiger, S. Weißenberger, and S. Günnemann. Diffusion improves graph learning. *Advances in neural information processing systems*, 32, 2019.

A. Ghosh, S. Boyd, and A. Saberi. Minimizing effective resistance of a graph. *SIAM review*, 50(1):37–66, 2008. 133

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017a. 25

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017b. 2, 47

J. H. Giraldo, K. Skianis, T. Bouwmans, and F. D. Malliaros. On the trade-off between over-smoothing and over-squashing in deep graph neural networks. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 566–576, 2023. 12

K.-I. Goh, B. Kahng, and D. Kim. Universal behavior of load distribution in scale-free networks. *Physical review letters*, 87(27):278701, 2001. 123

A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016. 25

J. Guo, L. Du, W. Bi, Q. Fu, X. Ma, X. Chen, S. Han, D. Zhang, and Y. Zhang. Homophily-oriented heterogeneous graph rewiring. In *Proceedings of the ACM Web Conference 2023*, pages 511–522, 2023.

K. Guo, K. Zhou, X. Hu, Y. Li, Y. Chang, and X. Wang. Orthogonal graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3996–4004, 2022. 5

B. Gutteridge, X. Dong, M. M. Bronstein, and F. Di Giovanni. Drew: Dynamically rewired message passing with delay. In *International Conference on Machine Learning*, pages 12252–12267. PMLR, 2023. 12, 60

W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017. 3, 25, 26, 27, 33, 37, 47, 77, 122

H. Han, J. Li, W. Huang, X. Tang, H. Lu, C. Luo, H. Liu, and J. Tang. Node-wise filtering in graph neural networks: A mixture of experts approach. *arXiv preprint arXiv:2406.03464*, 2024. 9

A. Hasanzadeh, E. Hajiramezanali, S. Boluki, M. Zhou, N. Duffield, K. Narayanan, and X. Qian. Bayesian graph neural networks with adaptive connection sampling. In *International conference on machine learning*, pages 4094–4104. PMLR, 2020. 6

D. He, C. Liang, H. Liu, M. Wen, P. Jiao, and Z. Feng. Block modeling-guided graph convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 4022–4029, 2022. 10, 17, 47, 49, 50, 77

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a. 6

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016b. 25

W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020a. xvi, 156, 165, 181, 187

Z. Hu, Y. Dong, K. Wang, and Y. Sun. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*, pages 2704–2710, 2020b. 144

K. Huang, Y. G. Wang, M. Li, et al. How universal polynomial bases enhance spectral graph neural networks: Heterophily, over-smoothing, and over-squashing. *arXiv preprint arXiv:2405.12474*, 2024a. 8

Y. Huang, W. Lu, J. Robinson, Y. Yang, M. Zhang, S. Jegelka, and P. Li. On the stability of expressive positional encodings for graphs. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=xAqcJ9XoTf. 16

Y. Huang, H. P. Wang, and P. Li. What are good positional encodings for directed graphs? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=s4Wm71LFK4. 16

M. S. Hussain, M. J. Zaki, and D. Subramanian. Global self-attention as a replacement for graph convolution. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 655–665, 2022. 144, 151, 184, 186

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015. 14, 146, 157

M. Jiang, G. Liu, Y. Su, and X. Wu. Gcn-sl: Graph convolutional networks with structure learning for graphs under heterophily. *arXiv preprint arXiv:2105.13795*, 2021.

D. Jin, Z. Yu, C. Huo, R. Wang, X. Wang, D. He, and J. Han. Universal graph convolutional networks. *Advances in Neural Information Processing Systems*, 34:10654–10664, 2021a. 9, 47

D. Jin, R. Wang, M. Ge, D. He, X. Li, W. Lin, and W. Zhang. Raw-gnn: Random walk aggregation based graph neural network. *arXiv preprint arXiv:2206.13953*, 2022. 9, 17

W. Jin, T. Derr, Y. Wang, Y. Ma, Z. Liu, and J. Tang. Node similarity preserving graph convolutional networks. In *Proceedings of the 14th ACM international conference on web search and data mining*, pages 148–156, 2021b. 9

Y. Jin, X. Yan, S. Liu, and X. Wang. A unified framework for combinatorial optimization based on graph neural networks. *arXiv preprint arXiv:2406.13125*, 2024. 190

C. Kanatsoulis, E. Choi, S. Jegelka, J. Leskovec, and A. Ribeiro. Learning efficient positional encodings with graph neural networks. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=AWg2tkbydO. 16

K. Karhadkar, P. K. Banerjee, and G. Montúfar. Fosr: First-order spectral rewiring for addressing oversquashing in gnns. *arXiv preprint arXiv:2210.11790*, 2022. 12, 17, 60, 113, 115, 118, 128, 135

N. Keriven. Not too little, not too much: a theoretical analysis of graph (over) smoothing. *Advances in Neural Information Processing Systems*, 35:2268–2281, 2022. 7

V. Khrulkov, L. Mirvakhabova, E. Ustinova, I. Oseledets, and V. Lempitsky. Hyperbolic image embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6418–6428, 2020. 144

J. Kim, D. Nguyen, S. Min, S. Cho, M. Lee, H. Lee, and S. Hong. Pure transformers are powerful graph learners. *Advances in Neural Information Processing Systems*, 35:14582–14595, 2022. 144

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 40, 183

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 3, 25, 26, 27, 33, 35, 36, 37, 47, 60, 77, 118, 130, 151, 160

J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018. 5, 26, 36, 39, 60

J. Klicpera, S. Weißenberger, and S. Günnemann. Diffusion improves graph learning. *Advances in Neural Information Processing Systems*, 32:13354–13366, 2019. 5, 26, 29, 34, 36

M. Koutrouli, E. Karatzas, D. Paez-Espino, and G. A. Pavlopoulos. A guide to conquer the biological network era using graph theory. *Frontiers in bioengineering and biotechnology*, 8:34, 2020. 25

D. Kreuzer, D. Beaini, W. Hamilton, V. Létourneau, and P. Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021. 15, 141, 144, 151, 155, 183, 186

G. Li, M. Muller, A. Thabet, and B. Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9267–9276, 2019. 26

G. Li, C. Xiong, A. Thabet, and B. Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020a. 6, 118, 155

H. Li, C. Li, J. Zhang, Y. Ouyang, and W. Rong. Addressing over-squashing in gnns with graph rewiring and ordered neurons. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2024.

P. Li, Y. Wang, H. Wang, and J. Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems*, 33:4465–4478, 2020b. 148

Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018. 4, 25, 141, 152

X. Li, R. Zhu, Y. Cheng, C. Shan, S. Luo, D. Li, and W. Qian. Finding global homophily in graph neural networks when meeting heterophily. In *International Conference on Machine Learning*, pages 13242–13256. PMLR, 2022. 10

L. Liang, F. Bu, Z. Song, Z. Xu, S. Pan, and K. Shin. Mitigating over-squashing in graph neural networks by spectrum-preserving sparsification. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=NiMu23k0Ym. 13

D. Lim, F. Hohne, X. Li, S. L. Huang, V. Gupta, O. Bhalerao, and S. N. Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in Neural Information Processing Systems*, 34:20887–20902, 2021. xv, 88, 90

D. Lim, J. D. Robinson, L. Zhao, T. Smidt, S. Sra, H. Maron, and S. Jegelka. Sign and basis invariant networks for spectral graph representation learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Q-UHqMorzil. 16

M. Liu, H. Gao, and S. Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 338–348, 2020. 6, 36, 37

M. Liu, Z. Wang, and S. Ji. Non-local graph neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 44(12):10270–10276, 2021. 10

Q. Liu, M. Nickel, and D. Kiela. Hyperbolic graph neural networks. *Advances in neural information processing systems*, 32, 2019a. 144

Y. Liu, C. Zhou, S. Pan, J. Wu, Z. Li, H. Chen, and P. Zhang. Curvdrop: A ricci curvature based approach to prevent graph neural networks from over-smoothing and over-squashing. In *Proceedings of the ACM Web Conference 2023*, pages 221–230, 2023. 13

Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, and Y. Qi. Geniepath: Graph neural networks with adaptive receptive paths. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4424–4431, 2019b. 5, 17

S. Luan, C. Hua, Q. Lu, J. Zhu, M. Zhao, S. Zhang, X.-W. Chang, and D. Precup. Revisiting heterophily for graph neural networks. *Advances in neural information processing systems*, 35:1362–1375, 2022. 8, 47

L. Ma, C. Lin, D. Lim, A. Romero-Soriano, P. K. Dokania, M. Coates, P. Torr, and S.-N. Lim. Graph inductive biases in transformers without message passing. In *International Conference on Machine Learning*, pages 23321–23337. PMLR, 2023. 155

J. Masci, E. Rodolà, D. Boscaini, M. M. Bronstein, and H. Li. Geometric deep learning. In *SIGGRAPH ASIA 2016 Courses*, pages 1–50. 2016. 2, 24

S. K. Maurya, X. Liu, and T. Murata. Improving graph neural networks with simple architecture design. *arXiv preprint arXiv:2105.07634*, 2021. 9

G. Mialon, D. Chen, M. Selosse, and J. Mairal. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021. 144, 148

F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. 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*, pages 5115–5124, 2017. 37

C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019. 141

C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. Tu-dataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020. 3, 131

K. Nguyen, N. M. Hieu, V. D. Nguyen, N. Ho, S. Osher, and T. M. Nguyen. Revisiting over-smoothing and over-squashing using ollivier-ricci curvature. In *International Conference on Machine Learning*, pages 25956–25979. PMLR, 2023. 12

G. Nikolentzos and M. Vazirgiannis. Random walk graph neural networks. *Advances in Neural Information Processing Systems*, 33:16211–16222, 2020.

H. Nt and T. Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019. 77, 80

K. Oono and T. Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019. 7, 25, 121

A. Ortega. *Introduction to graph signal processing*. Cambridge University Press, 2022. 80

A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, 2018. 80

L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. 5, 8, 123

G. A. Pavlopoulos, M. Secrier, C. N. Moschopoulos, T. G. Soldatos, S. Kossida, J. Aerts, R. Schneider, and P. G. Bagos. Using graph theory to analyze biological networks. *BioData mining*, 4(1):1–27, 2011. 25

H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020. xi, xv, 7, 9, 17, 35, 39, 58, 59, 85, 88, 90, 187

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

P. Pham, Q.-T. Bui, N. T. Nguyen, R. Kozma, P. S. Yu, and B. Vo. Topological data analysis in graph neural networks: Surveys and perspectives. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–19, 2025. doi: 10.1109/TNNLS.2024.3520147. 47

O. Platonov, D. Kuznedelev, M. Diskin, A. Babenko, and L. Prokhorenkova. A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=tJbbQfw-5wv. xiv, xv, 58, 59, 60, 61, 63, 73, 74, 88, 91

C. Qian, A. Manolache, K. Ahmed, Z. Zeng, G. V. d. Broeck, M. Niepert, and C. Morris. Probabilistically rewired message-passing neural networks. *arXiv preprint arXiv:2310.02156*, 2023. 12

L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022. 15, 144, 151, 155, 183, 184, 186

Y. Rong, W. Huang, T. Xu, and J. Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019. 6, 26

Y. Rong, Y. Bian, T. Xu, W. Xie, Y. Wei, W. Huang, and J. Huang. Self-supervised graph transformer on large-scale molecular data. *Advances in Neural Information Processing Systems*, 33:12559–12571, 2020.

E. Rossi, B. Charpentier, F. Di Giovanni, F. Frasca, S. Günnemann, and M. M. Bronstein. Edge directionality improves learning on heterophilic graphs. In *Learning on Graphs Conference*, pages 25–1. PMLR, 2024a.

E. Rossi, B. Charpentier, F. Di Giovanni, F. Frasca, S. Günnemann, and M. M. Bronstein. Edge directionality improves learning on heterophilic graphs. In *Learning on graphs conference*, pages 25–1. PMLR, 2024b. 10, 188

T. K. Rusch, B. P. Chamberlain, M. W. Mahoney, M. M. Bronstein, and S. Mishra. Gradient gating for deep multi-rate learning on graphs. *arXiv preprint arXiv:2210.00513*, 2022a. 6

T. K. Rusch, B. P. Chamberlain, M. W. Mahoney, M. M. Bronstein, and S. Mishra. Gradient gating for deep multi-rate learning on graphs. *arXiv preprint arXiv:2210.00513*, 2022b.

T. K. Rusch, M. M. Bronstein, and S. Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023. 4

K. Sancak, Z. Hua, J. Fang, Y. Xie, A. Malevich, B. Long, M. F. Balin, and Ü. V. Çatalyürek. A scalable and effective alternative to graph transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 20255–20263, 2025. 155

A. Sandryhaila and J. M. Moura. Discrete signal processing on graphs: Frequency analysis. *IEEE Transactions on signal processing*, 62(12):3042–3054, 2014. 80

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008. 2, 25, 47, 77, 118

M. Scholkemper, X. Wu, A. Jadbabaie, and M. T. Schaub. Residual connections and normalization can provably prevent oversmoothing in gnns. *arXiv preprint arXiv:2406.02997*, 2024. 6

P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008. 34

O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018. 34, 35, 156, 182

H. Shirzad, A. Velingker, B. Venkatachalam, D. J. Sutherland, and A. K. Sinop. Exphormer: Sparse transformers for graphs. In *International Conference on Machine Learning*, pages 31613–31632. PMLR, 2023. 151, 155

R. Shirzadkhani, S. Huang, E. Kooshafar, R. Rabbany, and F. Poursafaei. Temporal graph analysis with tgx. *arXiv preprint arXiv:2402.03651*, 2024. 189

Y. Song, C. Zhou, X. Wang, and Z. Lin. Ordered gnn: Ordering message passing to deal with heterophily and over-smoothing. *arXiv preprint arXiv:2302.01524*, 2023. 9

O. Stretcu, K. Viswanathan, D. Movshovitz-Attias, E. Platanios, S. Ravi, and A. Tomkins. Graph agreement models for semi-supervised learning. *Advances in Neural Information Processing Systems*, 32, 2019.

H. Sun, X. Li, Z. Wu, D. Su, R.-H. Li, and G. Wang. Breaking the entanglement of homophily and heterophily in semi-supervised node classification. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 2379–2392. IEEE, 2024. 10

Q. Sun, J. Li, H. Yuan, X. Fu, H. Peng, C. Ji, Q. Li, and P. S. Yu. Position-aware structure learning for graph topology-imbalance by relieving under-reaching and over-squashing. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 1848–1857, 2022. 13

Y. Sun, H. Deng, Y. Yang, C. Wang, J. Xu, R. Huang, L. Cao, Y. Wang, and L. Chen. Beyond homophily: Structure-aware path aggregation graph neural network. 9

S. Suresh, V. Budde, J. Neville, P. Li, and J. Ma. Breaking the limit of graph neural networks by improving the assortativity of graphs with local mixing patterns. *arXiv preprint arXiv:2106.06586*, 2021. 10, 77

S. A. Tailor, F. L. Opolka, P. Lio, and N. D. Lane. Do we need anisotropic graph neural networks? *arXiv preprint arXiv:2104.01481*, 2021. 81

J. Toenshoff, M. Ritzert, H. Wolf, and M. Grohe. Graph learning with 1d convolutions on random walks. *arXiv preprint arXiv:2102.08786*, 2021.

J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint arXiv:2111.14522*, 2021a. 11, 121

J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint arXiv:2111.14522*, 2021b. 12, 17, 60, 118

D. Tortorella and A. Micheli. Leave graphs alone: Addressing over-squashing without rewiring. *arXiv preprint arXiv:2212.06538*, 2022. 12

N. Tremblay, P. Gonçalves, and P. Borgnat. Design of graph filters and filterbanks. In *Cooperative and Graph Signal Processing*, pages 299–324. Elsevier, 2018. 80

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf. 13, 145

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 3, 25, 26, 27, 33, 36, 37, 47, 50, 60, 77, 118, 143, 151

P. Veličković and C. Blundell. Neural algorithmic reasoning. *Patterns*, 2(7):100273, July 2021. ISSN 2666-3899. doi: 10.1016/j.patter.2021.100273. URL http://dx.doi.org/10.1016/j.patter.2021.100273. 188

C. Wang, O. Tsepa, J. Ma, and B. Wang. Graph-mamba: Towards long-range graph sequence modeling with selective state spaces. *arXiv preprint arXiv:2402.00789*, 2024. 151

G. Wang, R. Ying, J. Huang, and J. Leskovec. Improving graph attention networks with large margin-based constraints. *arXiv preprint arXiv:1910.11945*, 2019. 32

H. Wang, H. Yin, M. Zhang, and P. Li. Equivariant and stable positional encoding for more powerful graph neural networks. In *International Conference on Learning Representations*, 2022a. URL https://openreview.net/forum?id=e95i1IHcWj. 16

T. Wang, D. Jin, R. Wang, D. He, and Y. Huang. Powerful graph convolutional networks with adaptive propagation mechanism for homophily and heterophily. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 4210–4218, 2022b. 10, 17, 47, 49, 50, 77

Y. Wang and T. Derr. Tree decomposed graph neural network. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 2040–2049, 2021. 9

F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019. 36, 47, 77, 100, 118

L. Wu, C. Tan, Z. Liu, Z. Gao, H. Lin, and S. Z. Li. Learning to augment graph structure for both homophily and heterophily graphs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 3–18. Springer, 2023a.

X. Wu, A. Ajorlou, Z. Wu, and A. Jadbabaie. Demystifying oversmoothing in attention-based graph neural networks. *Advances in Neural Information Processing Systems*, 36: 35084–35106, 2023b. 7

Z. Wu, P. Jain, M. Wright, A. Mirhoseini, J. E. Gonzalez, and I. Stoica. Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems*, 34:13266–13279, 2021. 144, 155

Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang. Beyond low-pass filtering: Graph convolutional networks with automatic filtering. *IEEE Transactions on Knowledge and Data Engineering*, 35(7):6687–6697, 2022. 81

X. Xie, Z. Kang, and W. Chen. Robust graph structure learning under heterophily. *arXiv preprint arXiv:2403.03659*, 2024.

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018a. 47, 130, 141, 151

K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462. PMLR, 2018b. 6, 17, 26, 27, 36, 160

Y. Yan, M. Hashemi, K. Swersky, Y. Yang, and D. Koutra. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 1287–1292. IEEE, 2022. 10, 47

C. Yang, R. Wang, S. Yao, S. Liu, and T. Abdelzaher. Revisiting over-smoothing in deep gcns. *arXiv preprint arXiv:2003.13663*, 2020. 6

M. Yang, R. Wang, Y. Shen, H. Qi, and B. Yin. Breaking the expression bottleneck of graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 2022a. 155

M. Yang, M. Zhou, Z. Li, J. Liu, L. Pan, H. Xiong, and I. King. Hyperbolic graph neural networks: A review of methods and applications. *arXiv preprint arXiv:2202.13852*, 2022b.

T. Yang, Y. Wang, Z. Yue, Y. Yang, Y. Tong, and J. Bai. Graph pointer neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8832–8839, 2022c. 10, 47

Y. Yang, X. Wang, M. Song, J. Yuan, and D. Tao. Spagan: Shortest path graph attention network. *arXiv preprint arXiv:2101.03464*, 2021. 6

Z. Yang, W. Cohen, and R. Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016. 25

C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu. Do transformers really perform badly for graph representation? *Advances in neural information processing systems*, 34:28877–28888, 2021a.

C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021b. 15, 141, 144, 148, 151, 184, 186

S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019. 144

H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019. 123

J. Zhang, H. Zhang, C. Xia, and L. Sun. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020a.

J. Zhang, H. Zhang, C. Xia, and L. Sun. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020b.

L. Zhang and H. Lu. A feature-importance-aware and robust aggregator for gcn. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1813–1822, 2020.

M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 25

Y. Zhang, X. Wang, C. Shi, X. Jiang, and Y. Ye. Hyperbolic graph attention network. *IEEE Transactions on Big Data*, 8(6):1690–1701, 2021. 144

Z. Zhang, Q. Liu, Q. Hu, and C.-K. Lee. Hierarchical graph transformer with adaptive node sampling. *arXiv preprint arXiv:2210.03930*, 2022.

L. Zhao and L. Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019. 6, 17, 26

Z. Zhong, S. Ivanov, and J. Pang. Simplifying node classification on heterophilous graphs with compatible label propagation. *arXiv preprint arXiv:2205.09389*, 2022a. 10

Z. Zhong, S. Ivanov, and J. Pang. Simplifying node classification on heterophilous graphs with compatible label propagation. *Transactions on Machine Learning Research (TMLR)*, 2022b.

D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. *Advances in neural information processing systems*, 16, 2003.

K. Zhou, X. Huang, Y. Li, D. Zha, R. Chen, and X. Hu. Towards deeper graph neural networks with differentiable group normalization. *arXiv preprint arXiv:2006.06972*, 2020. 6, 13, 17, 26, 148, 151

K. Zhou, Y. Dong, K. Wang, W. S. Lee, B. Hooi, H. Xu, and J. Feng. Understanding and resolving performance degradation in deep graph convolutional networks. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 2728–2737, 2021a.

K. Zhou, X. Huang, D. Zha, R. Chen, L. Li, S.-H. Choi, and X. Hu. Dirichlet energy constrained learning for deep graph neural networks. *Advances in Neural Information Processing Systems*, 34:21834–21846, 2021b. 6

H. Zhu and P. Koniusz. Simple spectral graph convolution. In *International conference on learning representations*, 2021. 81

J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020. 7, 9, 17, 47, 50, 60, 77, 187

J. Zhu, R. A. Rossi, A. Rao, T. Mai, N. Lipka, N. K. Ahmed, and D. Koutra. Graph neural networks with heterophily. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11168–11176, 2021a. 10, 47, 77

Y. Zhu, W. Xu, J. Zhang, Y. Du, J. Zhang, Q. Liu, C. Yang, and S. Wu. A survey on graph structure learning: Progress and opportunities. *arXiv preprint arXiv:2103.03036*, 2021b.