Ca' Foscari
University
of Venice

Ph.D. Degree
in Computer Science

Cycle XXXVII

Final Thesis

Advanced Representation Learning Techniques For Graph Neural Networks

Supervisor
Professor Marcello Pelillo

Co-supervisor
Professor Thilo Stadelmann

Graduand
Waqar Ali
Matriculation Number: 956676

Academic Year 2024/2025

Abstract

Over the past decade, deep learning models have become a key driver of modern artificial intelligence, enabling remarkable advancements in analyzing and interpreting complex data across various fields, such as computer vision, speech recognition, natural language processing, and beyond. These models are mainly developed for regular grid-structured data like sequences and images; however, not all forms of data conform to grid-like structures. A graph is a more general data structure that powerfully represents entities and their relationships, which is natural for fields such as social networks, biology, and chemistry. Modern deep learning models face significant challenges when applied to graph data due to the inherent complexity and non-euclidean nature of these graphs. Recently, Graph Neural Networks (GNNs) have been developed as a powerful tool for capturing the intricate structures and relationships within graph data. In this thesis, we delve into the various aspects that can enhance the performance of GNNs for standard graph learning tasks (e.g., graph and node classification) by introducing advanced graph representation learning techniques such as graph pooling, graph augmentation, community-based message passing, and graph rewiring.

First, we focus on graph pooling, an essential GNN building block, and design three novel graph pooling methods for learning graph representations. These methods reduce graphs' size and complexity while preserving essential structural information, enabling GNN models to achieve superior performance in graph classification tasks. Second, we propose advanced graph augmentation techniques to generate weakly labelled data samples, improving the generalization and robustness of GNN models. These techniques include the Node-Dropping Augmentation strategy, which selectively removes less impor-

tant nodes based on their degree and a structure learning method to reconnect isolated nodes by learning attention-based relationships.

Third, we develop an advanced message passing framework designed to address the challenges of heterophilic graphs, where connected nodes may have dissimilar features or labels. This improved message passing method enhances the GNN ability to capture diverse node attributes, leading to more accurate node classification outcomes. Lastly, we present a novel graph rewiring method to overcome the over-squashing problem in GNN architectures. This method strategically adds edges to ensure effective communication and information flow within the graph, preserving long-range dependencies and improving overall model performance.

Acknowledgments

Above all, I am deeply thankful to Allah, the Most Merciful and Compassionate, for His unending mercy, guidance, and wisdom during this journey. I sincerely acknowledge that I have the courage, perseverance, and knowledge to complete my thesis because of His boundless kindness and blessings. His divine will has been my guiding light, and I am profoundly aware of my complete reliance on His support in every step of this endeavor.

I would like to express my sincere gratitude to my supervisor, Professor Marcello Pelillo, whose vast knowledge, continuous support, and insightful guidance have been pivotal in the successful completion of this thesis. His constant support, thoughtful feedback, and encouragement have been invaluable throughout my Ph.D. journey. Additionally, our informal conversations, often revolving around shared interests in culture, religious talk, and the joys of desi Pakistani food (chicken biryani), have been particularly memorable, and I will always cherish them. I am also profoundly grateful to my co-supervisor, Professor Thilo Stadelmann, for his expert advice, collaborative spirit, and the countless opportunities he provided me for learning and professional development, especially during my internship at the Zurich University of Applied Sciences, where his support and encouragement greatly enriched my research journey and personal growth, laying a strong foundation for future collaborations in applied research settings. Thanks, Professor Thilo, for reviewing my thesis's advanced draft and providing detailed feedback that greatly helped me to refine and improve it.

I would like to express my deepest gratitude to my family for their unwavering support, without which none of this would have been possible. To my late mother, who always dreamed of seeing me earn a Ph.D., I dedicate this achievement to her memory and

enduring inspiration. A special note of gratitude also goes to my wife, whose endless patience, understanding, and unwavering support have guided me through the most challenging moments of this journey. Your love, sacrifices, and belief in me have made this accomplishment possible, and I am profoundly grateful to have you by my side.

My research stay at the University of Alicante, Spain, marked a pivotal chapter in my academic journey. I am deeply grateful for this opportunity to work under the supervision of Professor Francisco Escolano Ruiz, whose guidance, encouragement, and collaborative atmosphere were invaluable to my research. Additionally, I am thankful to Ahmed Begga for his collaboration and insightful discussions, which significantly contributed to our research projects.

I would like to thank my colleagues and friends at Ca' Foscari University for their unwavering support throughout my Ph.D. journey. Special thanks go to Anees, Zubair, Waqas, Anwar, Waseem, Federico, Alberto, and Guglielmo for their encouragement and companionship. I am especially grateful to Sebastiano, my primary collaborator during this journey, for the countless hours we spent together discussing ideas, coding, debugging, and writing papers, especially during intense deadlines. I would also like to thank Nicolla Miotello for his constant assistance with all the administrative matters I encountered during my doctorate. Your support has been invaluable to me.

In sum, this thesis represents my efforts and the unwavering support, guidance, and encouragement of everyone acknowledged above. I am deeply thankful to all who have contributed to my Ph.D. journey and am extremely grateful for your belief in me every step of the way.

Contents

Chapter 1

Introduction

"Data is the new science. Big Data holds the answers."

— Pat Gelsinger

1.1 Motivation

In the era of big data, the ability to derive meaningful insights from vast and complex datasets has become increasingly essential. Deep learning has emerged as a transformative technology for extracting significant value from massive datasets, obtaining unprecedented levels of accuracy and efficiency in numerous applications, such as face recognition from images, machine translation from text, and speech recognition from audio. Unlike traditional machine learning approaches, deep learning models excel in automatically learning features from large-scale data, driven by the surge in computational power and the exponential growth of available data.

Modern deep learning architectures are specially designed for data with regular structures, such as grids and sequences. For instance, Convolutional Neural Networks (CNNs) [60, 76] are tailored for image data arranged in grids, while Recurrent Neural Networks (RNNs) [33, 61] and Transformers [123] are designed for sequence data. This structural regularity simplifies the implementation of these models and contributes to their outstanding performance.

Nevertheless, many real-world entities, such as molecular compounds, brain structures, and social networks, can be naturally represented as graphs, which consist of nodes and edges capturing complex relationships and interactions. These graphs do not attach to the regular structure assumed by traditional deep learning models, posing unique challenges for data representation and processing. This thesis focuses primarily on Graph Neural Networks (GNNs) [72], a class of deep learning models, which is specifically designed to learn the relational and structural information inherent in graph data and shown significant promise across a diverse range of applications, including molecular structures, bioinformatics, physics, knowledge graphs, brain networks, Natural Language Processing (NLP), recommendation systems, Computer Vision (CV), social sciences, and more. Figure 1.1 shows the difference between the convolution operation of CNNs and the graph convolution operation of GNNs.

Despite the rapid progress and outstanding results attained by GNNs, the field of graph representation learning remains in its early stages. Further research is essential to unlock the full potential of these groundbreaking deep learning approaches. The main motivation of this thesis is to enhance the performance of GNNs in graph learning tasks, such as node and graph classification, by developing advanced graph representation learning techniques. Many components commonly used to construct GNNs often lack the necessary expressivity for addressing more advanced tasks because of their poor graph representations or limited design choices. For instance, the pooling operation is a key component of GNN that aggregates node and edge information to simplify large-scale graphs, enhancing GNNs' performance in real-world applications [143, 50]. However, the graph pooling methods are inconsistently addressed in the literature. This thesis identifies significant design flaws in several commonly used pooling layers in GNN architectures and proposes three novel approaches to address these issues, detailed in Chapters 3, 4 and 5.

Additionally, the message passing mechanism, a core component of GNNs, iteratively updates node representations by aggregating information from neighboring nodes. However, the standard message passing methods struggle to capture long-range interactions, especially in applications where such interactions are crucial for performance or dealing with

heterophilic (i.e., connected nodes have dissimilar labels). This prompts the redesign of typical representation learning architectures, addressing key aspects while opening new opportunities to solve interesting problems such as graph and node classification. Furthermore, the scarcity of labelled data poses a significant challenge in many graph classification tasks, such as predicting molecular properties, where obtaining labelled examples is often complex and resource-intensive. This lack of sufficient labelled data hinders the ability of GNNs to attain a promising prediction performance, highlighting the need for innovative techniques to enhance data availability and model robustness.

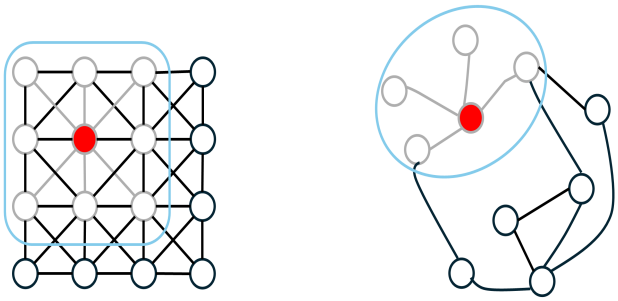


Fig. 1.1: *Shows the difference between traditional convolution operations on grid-structured data (left) and graph convolution operations on non-grid, irregular data structures (right). Similar to a graph, each pixel in the diagram is considered a node, and the filter window determines neighbors. The convolution operation captures the features of the red node along its neighbors. The neighbor nodes are ordered and fixed in size. Meanwhile, the graph convolution operation takes the features of the red node along its neighbors, which are unordered and variable in size. This highlights the adaptability of GNNs in processing diverse data types beyond regular grids.*

1.2 Problem Statement

The ultimate goal of enhancing graph representation learning techniques, as discussed in the previous section, requires addressing several key issues inherent in current GNN architectures and their components. GNNs must be able to efficiently and effectively handle the complexities of real-world graph data, which are often large-scale, heterogeneous, and dynamically evolving.

This gives rise to the following specific problems in GNNs for graph and node-level predictions:

1. **Complex structures and scalability:** As the size and complexity of graphs

increase, GNNs face computational challenges, leading to scalability issues that can impede performance in real-world applications. Simplifying large graphs by reducing their size while retaining critical graph features is necessary for performing efficient downstream tasks, such as graph classification.

2. **Generalization and robustness:** GNNs often struggle to generalize across different datasets and maintain robustness against perturbations, which is essential for their reliability in real-world applications. Achieving consistent performance under varying conditions and across diverse datasets remains a key challenge, highlighting the need for methods such as graph augmentation to enhance the model's generalization and robustness.

3. **Over-squashing problem:** In deeper GNN architectures, important information can become bottlenecked, leading to the loss of crucial details and impairing effective communication between distant nodes. This is an over-squashing problem, which impedes GNNs' ability to capture long-range dependencies.

4. **Heterophilic graphs:** Many real-world graphs exhibit heterophily, where connected nodes have dissimilar features or labels. Traditional GNN models, which often assume homophily, struggle to perform well on such graphs, resulting in poor classification performance.

By addressing the above challenges in the problem statement, this thesis aims to advance the field of representation learning, designing more efficient and effective graph representation learning techniques, including novel graph pooling methods to improve scalability and efficiency by reducing the size and complexity of graphs while preserving their essential structural information, advanced augmentation techniques to enhance the generalization and robustness of GNNs against diverse and perturbed datasets, graph rewiring methods to tackle the over-squashing problem and ensure effective communication and information flow within deep GNN architectures, and advanced message passing strategy to handle heterophilic graphs, thereby improving classification performance on

graphs where connected nodes have dissimilar features or labels. The proposed methods are expected to have significant implications across a wide range of applications, enhancing the applicability and performance of GNNs across various domains.

1.3 Contribution

Given the context and challenges outlined above, this thesis makes the following contributions to enhance the performance of GNNs:

- A novel graph pooling method, Quasi-CliquePool, is proposed to address the problem of finding overlapping nodes between two cliques. The experimental results demonstrate that combining the Quasi-CliquePool with existing GNN architectures yields an average improvement of 2% accuracy on four out of six graph classification benchmarks compared to the state-of-the-art. This contribution is explained in Chapter 3.

- We identify the limitations in existing graph clustering pooling methods and develop a novel Dominant Set clustering-based Multi-View Graph Pooling method (DSMVPool), which uses edge weights to extract the clusters without relying on a predefined cluster ratio. Furthermore, we design an attention-fusion-view convolution layer that refines the graph representations by integrating different sources of information, including local topology, coarser graph structure, and node features. Comprehensive experiments are performed on four different kinds of graph classification benchmarks, including CV, chemical, biological, and social networks, demonstrating superior performance against the state-of-the-art. This contribution is presented in Chapter 4.

- A novel pooling layer, Hierarchical Glocal Attention Pooling (HGLA-Pool), is introduced to address the limitations in existing clique pooling by sequentially integrating global structural properties of the graph with local node properties. This method designs a dynamic fusion technique that leverages both graph structural

information and node features to identify and rank the most relevant global structures (cliques). Additionally, a multi-attention LocalPool is developed to capture local node properties. Experimental results demonstrate that this method reduces the complexity and improves the representation of the graphs, consistently showing superior performance on seven diverse and challenging graph classification benchmarks. This contribution is presented in Chapter 5.

- In this contribution, we study the residual skip connections, attention mechanism, and Mixup augmentation methods for addressing over-smoothing issues in GNNs and enhancing their regularization in training. We design a Residual Attention Augmentation (RAA) methodology using skip connections with a multi-head attention strategy and Mixup augmentation method. This methodology shows superior performance on node-level prediction tasks. This contribution is outlined in Chapter 6.

- This contribution focuses on generating new data samples (graphs) to improve the generalization and robustness of the GNNs. We introduce the Node-Dropping Augmentation (NDAUG) method that selectively removes nodes with lower importance based on their degree while maintaining key topological motif structures. In the case of isolated nodes, we develop a structure learning method to reconnect these isolated nodes by learning attention-based relationships between nodes. Experiments demonstrate that combining the proposed NDAUG with existing GNN models yields an average improvement of 3% accuracy on eight graph classification benchmarks compared to the state-of-the-art baselines. This contribution is outlined in Chapter 7.

- A GNN architecture has been improved by incorporating valuable insights from community detection algorithms for addressing the heterophily issue (dissimilar labels among connected nodes). We developed an expressive and interpretable GNN model that can adapt to diverse graph properties and improve performance on various node classification tasks. This contribution is outlined in Chapter 8.

- We propose a novel graph rewiring method to improve communication within graphs by adding a linear number of edges locally to encourage community structures and globally to facilitate long-range connections. Our rewiring method generates a new optimized graph, and we utilize it as an augmented graph to increase the training size of the dataset, thereby improving the generalization and robustness of the GNN. This contribution is outlined in Chapter 9.

1.4 Thesis Structure

The rest of this thesis is structured as follows:

Chapter 2 introduces the mathematical notations and fundamental concepts related to graph theory used in the thesis. This chapter also focuses on the relevant background of GNNs (e.g., message passing mechanism, basic tasks in graph learning, graph pooling strategies), which is required to understand all other chapters. Additionally, it provides valuable literature related to graph augmentation methods and graph rewiring techniques for over-squashing problems. Chapters 3, 4, and 5 introduce the works related to graph pooling for GNNs. Specifically, Chapter 3 generalizes the notion of the clique to extract dense incomplete subgraphs as quasi-cliques, Chapter 4 proposes a novel graph pooling approach that can refine the graph representations by integrating different sources of information, including local topology, coarser graph structure, and node features, and Chapter 5 discusses the limitations of the existing clique-based pooling operations and proposes a novel pooling method called Hierarchical Glocal Attention Pooling. These three chapters address the first key issue of the problem statement.

Chapter 6 and Chapter 7 address the second key challenge of the problem statement. Chapter 6 focuses on the residual skip connections, attention mechanism, and Mixup augmentation methods for GNNs, and Chapter 7 introduces a novel node-dropping augmentation method to enhance the robustness and generalization of GNNs and test it on different domains, such as molecular graphs, bioinformatics, and social networks, for graph classification tasks.

Chapter 8 improves the message passing aggregation procedure of the GNN model for heterophily graph structures (dissimilar labels among connected nodes) by incorporating valuable insights from community detection algorithms. This chapter addresses the fourth part of the problem statement.

Chapter 9 addresses the second and third part of the problem statement. We develop a novel graph rewiring method called Inductive Spectral Theory (IST) that improves graph communication by optimizing its topology. Our method especially learns eigenfunctions that are reactive to graph labels and adds a linear number of edges locally and globally to encourage community structures and facilitate long-range connections.

Finally, Chapter 10 concludes this thesis and explains promising future work directions.

1.5 Publications

This thesis is primarily based on the following papers written during doctorate studies:

- **Waqar Ali**, Sebastiano Vascon, Thilo Stadelmann, and Marcello Pelillo. Quasi-cliquepool: Hierarchical graph pooling for graph classification. In Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, pp. 544-552. 2023.

- **Waqar Ali**, Sebastiano Vascon, Thilo Stadelmann, and Marcello Pelillo. Dominant Set Multi-View Graph Pooling for Graph Classification. Neural Networks Journal 2024 (under review).

- **Waqar Ali**, Sebastiano Vascon, Thilo Stadelmann, and Marcello Pelillo. Hierarchical Glocal Attention Pooling for Graph Classification. Pattern Recognition Letters Journal 2024.

- Muhammad Affan Abbas, **Waqar Ali**, Florentin Smarandache, Sultan S Alshamrani, Muhammad Ahsan Raza, Abdullah Alshehri, Mubashir Ali. Residual Attention Augmentation Graph Neural Network for Improved Node Classification.

Engineering, Technology & Applied Science Research (ETASR), 14, no. 2 (2024): 13238-13242.

- **Waqar Ali**, Sebastiano Vascon, Thilo Stadelmann, and Marcello Pelillo. Topology-Aware Node Dropping Augmentation for Graph Classification. European Symposium on Artificial Neural Networks (ESANN, 2025) (under review).

- Ahmed Begga, **Waqar Ali**, Gabriel Niculescu, Francisco Escolano, Thilo Stadelmann, and Marcello Pelillo. Community-Hop: Enhancing Node Classification through Community Preference. Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition and Structural and Syntactic Pattern Recognition (S+SSPR 2024).

- **Waqar Ali**, Ahmed Begga, Francisco Escolano, Sebastiano Vascon, Thilo Stadelmann, and Marcello Pelillo. Inductive Spectral Theory: Learnable Local-to-Global Spectral Rewiring in GNNs. In Proceedings of the 39th Annual AAAI Conference on Artificial Intelligence 2024 (under review).

Chapter 2

Background

"Mankind invented a system to cope with the fact that we are so intrinsically lousy at manipulating numbers. It's called the graph."

— Charlie Munger

2.1 Preamble

This chapter provides relevant background knowledge related to several topics (e.g., GNNs, graph pooling operations, graph augmentation methods and over-squashing issues in GNNs) and structures them into the larger context of graph representation learning to understand the thesis's contributions.

First, we introduce mathematical notations and basic graph theory definitions that will be used in this thesis. Section 2.3 introduces graph neural networks, including message passing mechanisms, different GNN architectures, and prediction tasks in graph learning. In section 2.4, we briefly explain graph pooling methods, which reduce the graph size for graph classification tasks. Section 2.5 gives an overview of graph augmentation methods, which increase the size of the datasets and enhance the generalization of GNN training. Lastly, we describe graph rewiring methods to mitigate over-squashing issues in GNNs and improve communication in graphs by introducing new edges on the bottleneck.

2.2 Definition and Notations

This section defines the essential concepts and mathematical notations that will be utilized in this thesis.

Graph: A graph is defined as a tuple of two sets $G = (V, E)$, where $V = \{1, ..., N\}$ is set of the nodes with cardinality $|V| = N$, and $E \subseteq V \times V$ is the set of edges.

Nodes: We represent nodes by scalar indices, such as nodes 1 and i. It is essential to understand that the numbering of nodes does not imply any specific order since V is generally an unordered set.

Edges: Edges are defined as pairs of nodes (i, j). In the case of directed edges, the sequence of the pair specifies the direction, indicating that edge (i, j) points from node i to node j. For undirected edges, we consider edges as $(i, j) \in E \Leftrightarrow (j, i) \in E$. In this thesis, we only focus on graphs with undirected edges.

Features: In this thesis, we also consider graph features, where each node and optionally each edge is associated with a feature vector. For instance, social network graphs may contain node features such as age, gender, and occupation, while edge features may include the nature of the relationship between individuals, such as being friends or coworkers. we represent the node feature vector as $x_i \in \mathbb{R}^d$, and similarly, we denote the edge feature vector as $e_{ij} \in \mathbb{R}^d$.

Feature matrices: We use $X \in \mathbb{R}^{N \times d}$ to represent all node features of a graph with N as the total number of nodes and d as the feature space's dimension. Similarly, we represent the edge feature matrix as $E \in \mathbb{R}^{N \times N \times d}$, but it is less common.

Adjacency matrix: An adjacency matrix can represent the connection between nodes of G as $A \in \{0, 1\}^{N \times N}$ with $A_{ij} = 1$ if $(i, j) \in E$ and $A_{i,j} = 0$ otherwise. For weighted graphs, the adjacency matrix can be represented as $A \in \mathbb{R}^{N \times N}$, where $A_{ij} > 0$ indicates the weight of the connection between node i and node j, and $A_{ij} = 0$ if there is no edge between them.

2.3 Graph Neural Networks

In recent years, GNNs have drawn considerable attention and become a powerful tool for analyzing graph-structured data, extending traditional neural networks to operate directly on graphs by learning structural and featural relationships between nodes to improve prediction performance. The foundational research on GNNs dates back to the Gori et al. [53] works and is further elaborated by Scarselli et al. [111]. These early studies aimed to design recurrent graph neural networks (RecGNNs) capable of processing graph-structured data. They learn a target node's representation by iteratively propagating neighbour information until a stable fixed point is reached. Over the following years, researchers developed several GNN models, laying the foundation for graph-structured data and demonstrating promising results in a variety of applications, such as social network analysis, molecular property prediction, recommendation systems, and natural language processing.

This section introduces the fundamental building blocks of GNNs. A GNN is a sequence of differentiable operations that take a graph as input and output an embedding representation vector for the entire graph or its nodes. In the following, we explain how the message passing mechanism generates the node embedding, the different models of GNNs and the common tasks in graph learning.

2.3.1 Node Embeddings in Graph Neural Networks

GNN models aim to generate node embeddings, also known as node representations, that contain information derived from both node features and the graph topology. Most of the GNN architectures utilize a neural message passing mechanism, where nodes exchange messages and update them using different aggregation functions (e.g., sum, max, average) or neural networks [138].

Message Passing Mechanism

The core idea of GNN is to iteratively update node representations by aggregating information from their neighbours, a process known as message passing. This mechanism allows GNNs to utilize each node's local neighbourhood information by passing messages through the graph to refine and update their representations. So, GNNs can effectively capture the graph's structural information and the features associated with each node, enabling them to learn rich and expressive node representations that incorporate local and global contexts. The message passing strategy in GNNs is similar to the convolution operation in CNNs, but it is specifically designed for graph-based data. Specifically, at each iteration of the message passing in a GNN, the embedding $h_i^{(k)}$ of each node $i \in V$ is updated based on the aggregated information from its neighbourhood $N(j)$. This procedure can be illustrated mathematically as follows:

$$\mathbf{h}_i^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_i^{(k)}, \text{AGGREGATE}^{(k)}\left(\left\{\mathbf{h}_j^{(k)} \mid j \in \mathcal{N}(i)\right\}\right)\right) \tag{2.1}$$

$$\mathbf{h}_i^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_i^{(k)}, \mathbf{m}_{\mathcal{N}(i)}^{(k)}\right) \tag{2.2}$$

where Aggregate and Update are functions (i.e., sum, mean, max or neural networks) and $\mathbf{m}_{\mathcal{N}(i)}^{(k)}$ represents the "message" that is aggregated from i's graph neighbourhood $N(i)$. The aggregate function takes the set of embeddings of the nodes in i's graph neighbourhood $N(i)$ as input and generates a message $\mathbf{m}_{\mathcal{N}(i)}^{(k)}$ based on this aggregated neighbourhood information. Then, the update function generates the updated embedding by combining the message $\mathbf{m}_{\mathcal{N}(i)}^{(k)}$ with the previous embedding h_i of node i. The input feature set for all the nodes at initial embeddings when $k = 0$ such as $\mathbf{h}_i^{(0)} = \mathbf{x}_i, \forall i \in V$. The following equation can be used to define the embeddings for each node after K number of GNN message passing layers:

$$\mathbf{z}_i = \mathbf{h}_i^K, \forall i \in V \tag{2.3}$$

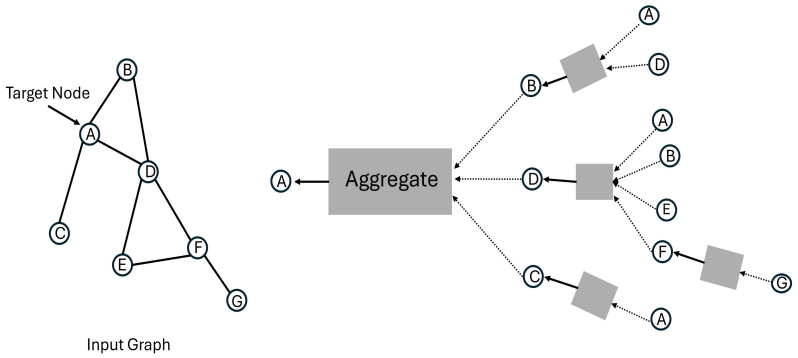


Fig. 2.1: *Abstract view of how a single node aggregates messages from its local neighborhood in a GNN. Consider node A, a target node that collects messages from its neighboring nodes (i.e., B, C, and D). The messages that A receives from these neighbors are aggregates of information gathered from their respective neighborhoods, creating a multi-layered information flow. This visualization illustrates a three-layer message passing architecture, where each layer enables the nodes to incorporate progressively more distant information.*

Figure 2.1 is an abstract representation of how a single node aggregates messages from its local neighborhood.

2.3.2 Graph Neural Network Architectures

There are many ways to design the message passing layer depending on the nature of tasks. In the following, we briefly describe the different GNN architectures.

Graph Convolutional Networks (GCNs)

This is the most popular baseline GNN model, employing the Kipf [72] normalized aggregation with the self-loop update approach. The GCN model defines the message passing function using the combination of non-linearities and linear transformations:

$$\mathbf{H}^{(k+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\mathbf{H}^{(k)}\mathbf{W}^{(k)}\right) \tag{2.4}$$

where σ is the row-wise non-linear activation function, $\tilde{A} = A + I$ is the adjacency matrix with added self-loops, $\tilde{D}$ is the corresponding degree matrix, and $\mathbf{H}^{(k)}$ is the matrix of node features at layer k.

Graph Attention Networks (GATs)

This model is proposed by Veličković et al. [125], which utilizes attention mechanisms to weigh the importance of neighboring nodes differently. The attention coefficients are computed as:

$$e_{ij} = \text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j]) \tag{2.5}$$

where $\mathbf{a}$ is a learnable weight vector, and $\|$ denotes concatenation. The normalized attention coefficients are then used to aggregate the node features:

$$\mathbf{h}_i^{(k+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} \mathbf{W}\mathbf{h}_j^{(k)}\right) \tag{2.6}$$

where $\alpha_{ij} = \text{softmax}(e_{ij})$.

GraphSAGE

Hamilton et al. [56] introduced Graph Sample and Aggregate (GraphSAGE), which generates node embeddings by sampling and aggregating features from a node's local neighborhood. The GraphSAGE framework can be described as:

$$\mathbf{h}_i^{(k+1)} = \sigma\left(\mathbf{W}^{(k)} \cdot \text{AGGREGATE}^{(k)}\left(\{\mathbf{h}_j^{(k)} : j \in \text{Sample}(\mathcal{N}(i))\}\right)\right) \tag{2.7}$$

Graph Isomorphism Networks (GINs)

Introduced by Xu et al. [138], GINs aim to distinguish different graph structures more effectively by employing a sum aggregation function. The layer-wise propagation rule for GINs is:

$$\mathbf{h}_i^{(k+1)} = \text{MLP}^{(k)}\left((1 + \epsilon^{(k)})\mathbf{h}_i^{(k)} + \sum_{j \in \mathcal{N}(i)} \mathbf{h}_j^{(k)}\right) \tag{2.8}$$

where $\text{MLP}^{(k)}$ is a multi-layer perceptron, $\epsilon^{(k)}$ is a learnable parameter or a fixed scalar, $\mathbf{h}_i^{(k)}$ is the hidden state of node i at layer k, and $\mathcal{N}(i)$ denotes the set of neighbors of node i. All of the above-mentioned GNN variants showcase their adaptability to different types of data to solve a wide range of prediction tasks using graph-structured data. The principal difference among these variants lies in their output representations.

2.3.3 Tasks in Graph Learning

There are three common prediction tasks in graph learning, each highlighting the versatility of GNNs.

Node Classification

In this type of task, the GNN aims to predict the label of individual nodes within a graph by learning the representation of each node. For instance, classifying users in a social network based on their roles or attributes or predicting traffic on different roads. In classical deep learning, an example of node-learning is the task of image segmentation, in which each pixel is classified as belonging to a particular object or region. Generally, the node classification task can be represented as:

$$\hat{y}_i = \text{softmax}(\mathbf{h}_i^{(K)})$$

where $\hat{y}_i$ is the predicted label of node i.

Graph Classification

In graph-level learning, the goal of the GNN is to assign a label to an entire graph. This is useful in domains such as chemistry, where graphs can represent molecules and the task is to predict properties like toxicity or activity. For graph classification, GNNs can combine convolutional layers with pooling operations, like in CNNs, and use a readout layer for mapping the graph to a vector representation. The graph classification task can

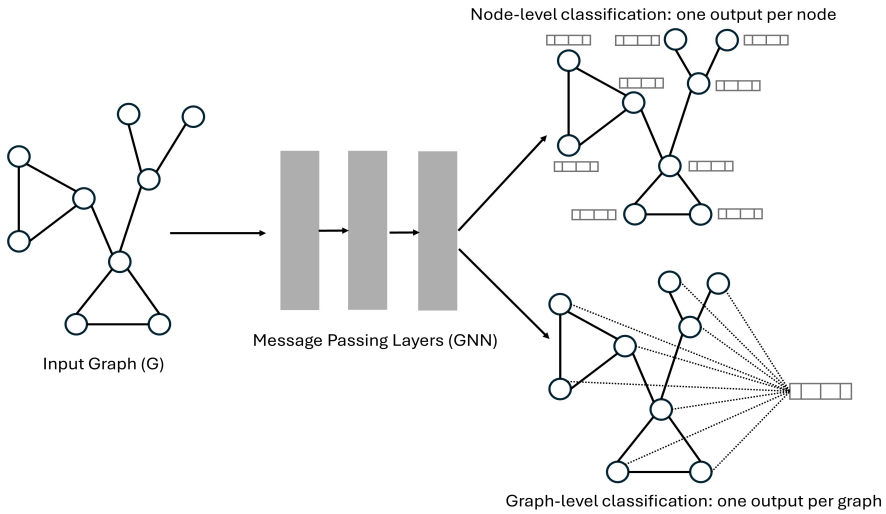


Fig. 2.2: *Structural representation of graph-level and node-level classification tasks. The stacking of message passing layers as a GNN outputs a single node embedding or a prediction for each node at the node level and a single prediction for the entire graph at the graph level.*

be expressed as:

$$\hat{y}_G = \text{softmax}(\mathbf{h}_G)$$

where $\hat{y}_G$ is the predicted label of the graph.

Edge Prediction

Another relevant use of GNNs is edge prediction, which involves determining the existence or type of edges between nodes. It is also known as link prediction, and this can be structured as a specific case of node-level prediction, such as predicting node representations that are similar for nodes that should be connected by an edge. This task is essential for applications like link prediction in social networks, where the goal is to predict potential future connections. The edge prediction can be formalized as:

$$\hat{y}_{ij} = \sigma(\mathbf{h}_i^{(K)} \cdot \mathbf{h}_j^{(K)})$$

where $\hat{y}_{ij}$ is the predicted probability of an edge existing between nodes i and j. Figure 2.2 shows the difference between the graph and node-level prediction tasks settings.

This thesis focuses on graph classification and node classification tasks. The Figure 2.3

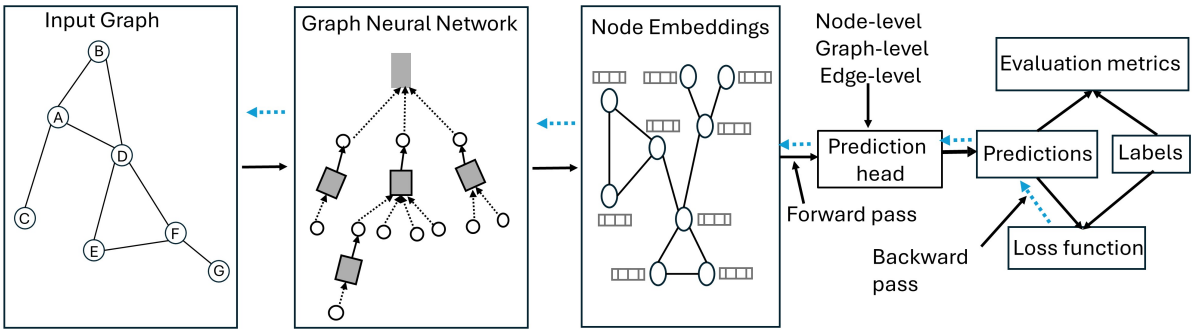


Fig. 2.3: *The training pipeline for graph learning. In the first step, GNN receives a graph as input and generates node embeddings. Next, the prediction head uses these embeddings and performs predictions for node-level, graph-level, or edge-level tasks. The loss function calculates the loss and compares it to true labels. The loss is then used in the backward pass to update the model parameters. Evaluation metrics assess the model's performance throughout the training process.*

illustrates how a GNN model processes an input graph to generate node embeddings and make predictions.

2.4 Graph Pooling Operations

Graph Pooling methods are an essential component of GNNs, designed to reduce the number of nodes in a graph. Pooling methods provide two primary benefits: they reduce computational requirements and generate a more abstract representation of the graph by mapping the nodes or subgraphs into a compact representation, similar to the role of pooling layers in CNNs. Therefore, studying and improving graph pooling methods is significant to enhancing GNN performance across various domains, driving the progress of graph learning.

In general, pooling operations are defined as functions that transform a graph $G = (V, E)$ into a smaller graph $G' = (V', E')$ where $|V'| < |V|$. This process is also known as graph coarsening. A detailed description of the pooling operation will be presented in Chapters 3, 4, and 5 as part of this thesis's contributions. This section provides an overview of the diverse pooling techniques found in the literature.

Earlier works used only graph coarsening methods without neural networks. For example, spectral clustering methods achieve coarsened graphs using eigendecomposition

[131]. However, the eigendecomposition procedure was not good in terms of time complexity. Dhillon et al. [38] first time proposed a Graclus method to extract the clusters of given graphs without eigenvectors. Graclus method used the concept of mathematical equivalence between a general weighted kernel k-means objective and a general spectral clustering objective. This technique improves the several weighted graph clustering objectives, including ratio cut and normalized cut. Even in recent GNN models [18, 104], Graclus is used as a pooling module.

Recently, many pooling techniques have been proposed based on different design principles and requirements to extract global and hierarchical structural information in the graphs. Global property-based pooling methods usually adopts summation or average operations to integrate the embeddings of all nodes, resulting in a single vector representation for the entire graph. Gilmer et al. [51] introduced a message passing scheme based on a general framework for graph classification and obtained the entire graph classification using the Set2Set model. In [147], the authors proposed the SortPooling method to keep much more node information and learn from the global graph topology. This method sorts the node embeddings according to the graph structural roles and then feeds these sorted embeddings to the next layers.

Global pooling is the most effective way to reduce the size of the graph. However, these methods ignore the hierarchical graph information, which is important for capturing the structural information of graphs. The principal goal of hierarchical pooling approaches is to build a technique that uses graph topology or node feature information to learn the node representation hierarchically. In this regard, Ying et al. [143] proposed the first Differentiable Pooling (DiffPool) method that learns a hierarchical clustering of graph nodes, generating a coarsened graph representation by assigning nodes to clusters. This method can be used with various GNN architectures in an end-to-end fashion. Recently, authors introduced a clique-based graph pooling method [84], partitioning the graph into cliques (complete subgraphs). The nodes within each clique are then merged to form supernodes and aggregate node features within each supernode. This method effectively reduces the graph's size while preserving the local connectivity structure. Furthermore,

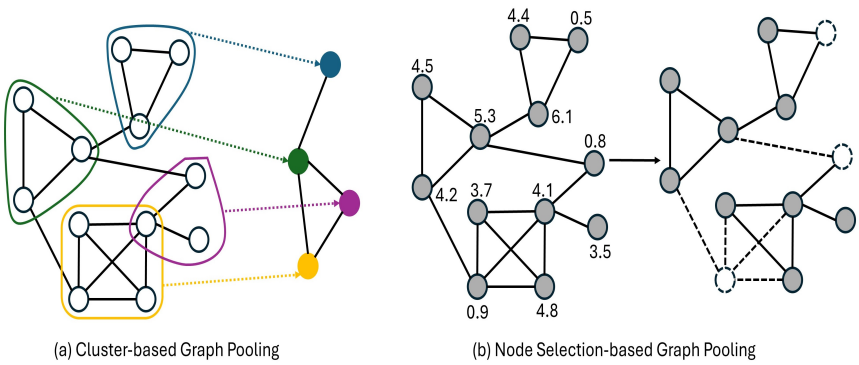


Fig. 2.4: *The diagram illustrates two distinct approaches to graph pooling: cluster-based graph pooling (a) and node selection-based graph pooling (b). In cluster-based pooling, nodes are grouped into clusters based on connectivity, and each cluster is represented as a single node, generating a pooled graph. In node selection-based pooling, nodes are scored based on importance, with high-scoring nodes retained and low-scoring nodes removed, resulting in a pooled sparser graph that maintains essential structural features.*

the k-plex pooling method [9] generalizes clique pooling by allowing for k-plexes, which are subgraphs where each node is connected to all but k nodes within the subgraph. This method provides more flexibility than strict cliques, extracting more complex and less densely connected structures.

The cluster-based pooling methods are very effective in practice but have been criticized for their high memory consumption [26]. To overcome these challenges, several studies have proposed a range of sparse operators collectively referred to as node-selection graph pooling methods. Figure 2.4 shows how cluster and node-selection pooling methods process an input graph. These methods utilize learnable transformation functions to project node features into a scoring vector, and then this scoring vector is used to generate a pooled graph by selecting the most important nodes. Hongyang et al. [50] developed a Topk-k Pooling method that selects a fixed number of nodes based on their importance scores, typically computed using a learnable projection vector. Nodes with the Top-k highest scores are selected, and their features are passed to the next layer. In SAGPool [77], the authors introduced self-attention mechanisms to determine the importance of each node. Nodes with higher attention scores are selected for the pooled graph. Jinheon et al. [140] proposed Multistructure Attention Convolutional (MAC) pooling that incorporates dual-node scoring strategies to obtain the importance of nodes. Similar to node-selection methods but focusing on edges rather than nodes, Edge Contraction

Pooling [40, 39] computes a score for the edges incident to each node. Based on these computed scores, edges are then iteratively contracted, meaning their endpoints are merged into a single node.

2.5 Graph Augmentation

In recent years, GNNs have achieved remarkable advancements in graph representation learning, excelling in graph and node classification tasks [72, 125]. However, a major challenge remains the limited availability of labelled datasets for many graph classification problems [144]. For example, in the domain of molecular property prediction, GNNs are extensively utilized, but acquiring labeled molecular data often requires intricate and labor-intensive laboratory processes. Consequently, the scarcity of sufficiently labeled samples hinders the ability of GNNs to achieve promising prediction performance.

Data augmentation methods are known for their efficiency and effectiveness in generating new weak-labeled synthetic data samples from the existing training data, providing a straightforward and cost-efficient method to enhance the generalization of a deep model. This strategy is preferred over resource-intensive methods like gathering extra real data or significantly changing the model architecture or training algorithms.

Data augmentation methods have been demonstrated to be helpful in CV and NLP [101]. Effective data augmentation approaches for CV include image flipping, noise injection, and cutout [37]. Additionally, methods like Generative Adversarial Networks [52] and Auto-Encoder [71] also contribute to generating new samples by learning the distribution of data. However, applying such techniques to graphs is more complex due to their non-Euclidean nature, where nodes are irregularly connected by edges [153, 41], presenting unique challenges in augmentation.

Recent works have focused on developing graph augmentation by revising the graph's structures and manipulating node features for node-level and graph-level prediction tasks [126, 73]. Structure-based augmentation alters the graph's topology to create diverse

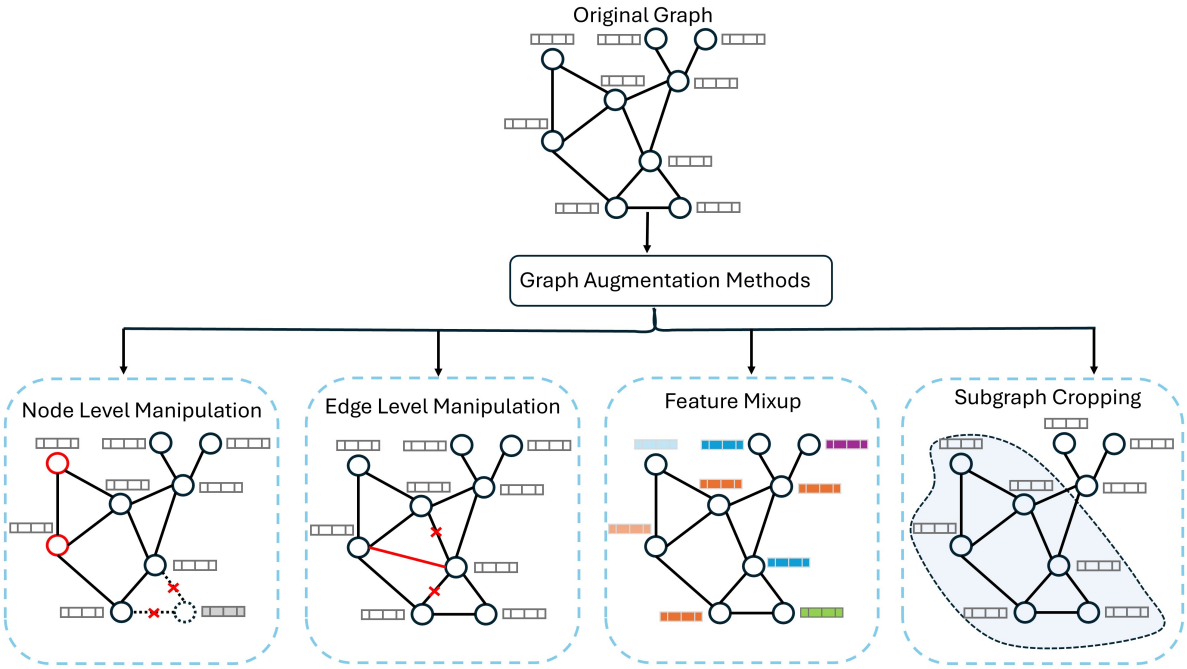


Fig. 2.5: *An outline of different graph augmentation methods for graph representation learning.*

graph structures and the Figure 2.5 shows both structure and feature-based graph augmentation methods. These methods can include edge and node perturbation, where edges and nodes are randomly added or removed while maintaining the overall structure. For example, DropEdge [106] employs a random method to remove a uniform portion of edges and generate augmented graphs to enhance the robustness of the GNN model during test time inference, AdaEdge [28] methodology uses an iterative process to add (or remove) edges connecting nodes that are predicted to have similar (or different) labels with a high level of probability, and DropNode [144] randomly deletes a certain portion of nodes from the original graph, resulting in the generation of augmented graphs. The DropNode augmentation operation can disconnect closely related nodes in the augmented graph, which can lead to a loss of the graph's overall structural information [132].

Feature-based augmentation methods manipulate the node features to create new training samples. Researchers have recently developed Mixup augmentation methods [59, 95, 1] for graph augmentation, which generates augmented graphs by interpolating the features of node pairs. This method helps the model to generalize better by exposing it to intermediate feature representations that lie between the original nodes. In the last, subgraph-

based augmentation involves generating subgraphs from the original graph. The authors in [132] developed a GraphCrop method, which generates various cropped-augmented graphs using a node-centric strategy.

2.6 Over Squashing

As mentioned in section 2.3, message passing is a key component of a GNN model, where node features are iteratively updated by aggregating information from neighboring nodes and generating the node embedding for nodes [51]. Further, this node embedding output is used to perform various tasks like graph and node classification.
However, the GNN's message passing mechanism faces significant challenges, particularly in practical applications that require capturing long-range interactions. One prominent issue is over-smoothing, where node features become indistinguishable as the number of layers increases [20]. This convergence of features limits the depth of GNNs, thereby restricting their ability to capture complex relationships within the data.

Another critical issue is over-squashing, first identified by [6]. This problem arises as the depth of a GNN increases, requiring information from potentially exponentially expanding receptive fields to be processed simultaneously at each message passing step. This scenario creates a bottleneck, where an exponential amount of information is compressed into fixed-size node vectors, leading to over-squashing [6] (see the Figure 2.6). As a result, GNNs may fail at tasks that depend on long-range interactions. Oversquashing tends to occur when the GNN has sufficient layers to encompass the entire graph (large receptive fields) but not enough to manage and process all the relationships between nodes effectively. One of the prevalent strategies to address this issue is graph rewiring, which aims to alter the graph's edges to reduce structural bottlenecks [121]. Chapter 9 will present a detailed description of graph rewiring methods and possible solutions as part of this thesis's contributions. This section provides an overview of the diverse graph rewiring methods in the literature.

One of the prevalent strategies to address this issue is graph rewiring, which aims to

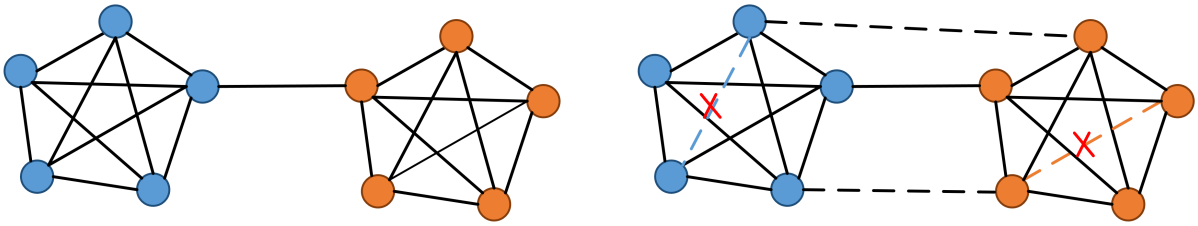


Fig. 2.6: *This diagram depicts the over-squashing problem in GNNs and a potential solution through graph rewiring methods. The over-squashing problem is illustrated on the left, where information from distant nodes becomes bottlenecked, impairing effective communication. On the right, the graph rewiring solution is shown, where additional edges are added to the graph to alleviate bottlenecks, thus improving the flow of information and enhancing the GNN's ability to capture long-range dependencies.*

alter the graph's edges to reduce structural bottlenecks [121]. A common approach to rewiring includes making the graph fully connected or applying transformer architectures to form attention-based connections between nodes [75, 102]. These spatial rewiring techniques often use a k-hop neighborhood strategy connecting each node to others [2]. However, these methods can sometimes neglect the graph's inherent structural features and may suffer from high computational costs and issues with noisy attention weights [6]. Recent studies have explored leveraging graph-theoretic metrics such as spectral gap, commute time, and effective resistance to adjust edges or their weights [7, 66, 19]. These adjustments aim to diminish bottlenecks and enhance information flow across the graph.

2.7 Conclusion

This chapter covered the fundamental principles of GNNs, including generating node embedding through a message passing mechanism, various GNN architectures and different graph learning tasks. We also explored the limitations in existing graph learning approaches, which have driven the development of more advanced graph representation learning techniques, such as pooling methods, enhanced message passing mechanisms, novel augmentation strategies, and graph rewiring techniques to improve graph representations. The concepts and methods described in this chapter will serve as the basis for the novel methods and applications explored in this thesis to enhance further the capabilities of GNNs.

Chapter 3

Quasi-Clique Pooling for Graph Neural Networks

"Patterns are not just the domain of design but the essence of understanding complexity."

— Dan Cederholm

3.1 Preamble

In this chapter, we implemented a novel Quasi-CliquePool method to overcome the limitations of existing clique pooling. The research and findings discussed in this chapter are based on the following paper:

- **Waqar Ali**, Sebastiano Vascon, Thilo Stadelmann, and Marcello Pelillo. "Quasi-cliquepool: Hierarchical graph pooling for graph classification." In Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, pp. 544-552. 2023 [4].

The author has the following contributions:

- **Developing** the overall framework of the algorithm.
- **Writing** the majority of the code.

- **Performing** all the experiments.
- **Writing** a significant part of the paper.

3.2 Introduction

In recent years, CNN models have demonstrated outstanding performance in various challenging tasks in the fields of image processing, video processing, natural language interpretation, and beyond [90, 55]. These tasks typically represent data in euclidean space, whereas a large amount of data exists in non-euclidean domains, such as chemical molecules, biological and social networks, which can usually be represented as graphs [34]. Therefore, attempts have been made to successfully generalize CNN models to operate on graph data, leading to GNNs.

GNNs have been implemented for various kinds of graphs and have achieved state-of-the-art performance for many graph-related tasks, such as classifying nodes, classifying graphs, and predicting links [150]. These findings demonstrate that GNNs are effective at node-level and graph-level representations. On the other hand, pooling approaches are demonstrated to be efficient and effective in many natural language processing [74] and image-related tasks such as text and image classification [78]. It is thus natural to investigate these techniques also for graph data [77]. The researchers generalize the CNN pooling methods on graphs to reduce the size of nodes for graph-level prediction. For example, studies have extended the global average or sum pooling operations to graph models by averaging or summing all node features [116]. But, such pooling methods are not able to capture the hierarchical graph structure and may lose important features [143].

Several advanced graph pooling techniques, like DiffPool [143], Top-k pooling [50], and SortPool [77], have been developed to overcome the shortcomings of global pooling and have achieved promising results on graph classification tasks. Furthermore, in [9, 84], the authors proposed topology-based clique and k-plex hierarchical pooling methods for

graph classification.

However, the pooling techniques mentioned above have room for improvement. For instance, the DiffPool method produces a dense adjacency matrix due to its differentiable nature. It requires hyper parameterization in the form of a prior on the number of clusters or nodes allowed to be pruned. Top-k pooling introduced a new gPool method to overcome this issue [50]. Based on the scalar projection values of the nodes, the gPool method selects the top-k nodes to reduce the size of the graph. But, this method ignores the topological structure of the graph. In [84], the authors proposed a clique pooling method for graph classification using topological information. Still, clique pooling is much more restricted and less flexible than k-plex pooling because it is limited to hard graph partitions [9]; hence, k-plex pooling has achieved good results compared to clique pooling. However, the k-plex pooling depends on the k number of adjacent nodes. The method proposed in this chapter is close to clique and k-plex pooling methods.

This chapter links pooling operators and two graph theoretical concepts: clique and quasi-clique. The former performs a hard partitioning between nodes, where each node is connected up to one cluster. The latter ones provide flexible partitioning for a clique, which relaxes the definition of a clique to a quasi-clique to extract dense, incomplete subgraphs within a large graph. In this chapter, we propose *Quasi-CliquePool* (cf. Figure 3.1), a novel pooling technique to learn a hierarchical representation of a graph to address the limitations mentioned above. The proposed method uses the Replicator Dynamics (RD) algorithm to extract a subset of nodes (maximal clique) iteratively. The RD is a dynamical system that, at convergence, provides the likelihood of participation of each node into a cluster; the higher the value, the central the node.

In this chapter, we introduce a new soft peel-off strategy to find the low-participating nodes in a converged RD. Such nodes, being not central, lie on the border of the cluster. Hence they are good candidates for being the link with other clusters in the graph. Those nodes are allowed to get extracted again with other clusters, while the central ones belong only to one cluster. Having nodes that belong to multiple clusters inevitably extends the maximal clique concept to a quasi-clique due to missing edges. Nodes that

remain isolated during the pooling operations are removed from the graph. Details are discussed in Section 3.5. We make a coarsened graph based on extracted cliques in the last step. Overall, this chpater makes the following contributions:

- We propose a novel graph pooling method, Quasi-CliquePool, based on the concept of quasi-clique. The proposed graph pooling method can be integrated into various GNN architectures.
- We also introduce a new soft peel-off strategy to find the overlapping cluster/clique nodes of a given graph during the pooling procedure.
- We conduct comprehensive experiments and show that Quasi-CliquePool improves an average accuracy by 2% in four out of six graph classification benchmarks compared to state-of-the-art pooling methods.

3.3 Related work

This section briefly explains the multiple GNN models and graph pooling methods for graph classifications.

3.3.1 Graph Neural Network Models

GNN models have drawn considerable attention due to their excellent performance on various tasks in the graph learning representation domain. Recently, several GNN architectures have been developed, including architectures inspired by CNNs, recursive graph networks [111], recurrent graph networks [81], and line graph neural networks [25]. Gilmer et al. [51] proposed "a neural message passing" framework for graph data, and most of the above approaches fit within this framework. In this message-passing framework, the GNN model computed the node representations directly from their neighbour nodes' features using a differentiable aggregation and propagation function. In [57], the authors present a comprehensive review of recent advancements in this domain, and [22] makes connections to spectral graph convolutions.

3.3.2 Graph Pooling Methods

The pooling methods reduce the graph size using node dropping or node pruning for graph classification. There are three categories of graph pooling: topology, global and hierarchical pooling [77].

Topology based pooling

Earlier works used only graph coarsening methods without neural networks. For example, spectral clustering methods achieve coarsened graphs using eigendecomposition [131]. However, the eigendecomposition procedure was not good in terms of time complexity. Dhillon et al. [38] first time proposed a Graclus method to extract the clusters of given graphs without eigenvectors. Graclus method used the concept of mathematical equivalence between a general weighted kernel k-means objective and a general spectral clustering objective. This technique improves the several weighted graph clustering objectives, including ratio cut and normalized cut. Even in recent GNN models [104, 18], Graclus is used as a pooling module.

Global pooling

Global pooling methods use neural networks or summation functions to pool all the node representations in each layer. Gilmer et al. [51] introduced a message-passing scheme based on a general framework for graph classification and obtained the entire graph classification using the Set2Set model. In [147], the authors proposed the SortPooling method to keep much more node information and learn from the global graph topology. This method sorts the nodes embeddings according to the graph structural roles and then feed these sorted embedding to the next layers.

Hierarchical pooling

Global pooling is the most effective way to reduce the size of the graph. However, these methods ignore the hierarchical graph information, which is important for capturing the structural information of graphs. The principal goal of hierarchical pooling approaches

is to build a technique that uses graph topology or node feature information to learn the node representation hierarchically. In this regard, Ying et al. [143] proposed the first hierarchical DiffPool method for classifying graphs. This method can be used with various GNN architectures in an end-to-end fashion. It used the learning assignment matrix that contains the probability values of nodes in layer L and then assigned these values to clusters in the next layer $L+1$. Due to its differentiable essence, its application produces dense adjacency matrices. In [50], the authors developed the Top-k pooling to overcome this issue by learning a project vector. But, this method ignores the topological structure of the graph. Enxhell et al. [84] introduced a clique-based hierarchical pooling method for graph classification. Clique pooling is much more restrictive because it is limited to hard graph partitions. In [9], the authors improved the simple clique pooling method and proposed a k-plex pool method for graph classification. To further improve pooling methods, this chapter proposes a novel Quasi-CliquePool method that can use topological information to yield hierarchical representations.

3.4 Limitations in Existing Graph Poolings

Despite the advancements in graph pooling methods, several limitations still hinder their effectiveness in accurately capturing graph structures. Table 3.1 summarizes the current graph pooling methods based on their four desirable clustering properties: 1) Hierarchical pooling—Global pooling methods ignore the hierarchical structure information in the graphs during pooling operation however, hierarchical pooling methods extract the hierarchical information. Our quasi-pooling can extract hierarchical information from graphs and can be combined with various GNN architectures, 2) Adaptive—we can distinguish pooling methods based on the k number of nodes of the pooled graph. The k-plex [9] is a fixed pooling method because it depends on the apriori k number of adjacent nodes. Our pooling method is adaptive because it is not dependent on any k number of adjacent nodes, 3) Topology pooling—in terms of graph clustering, topology structure-based aggregation is crucial for pooling operations. DiffPool [143] and Top-k [50] only used

Table 3.1: *Related work in terms of four desirable graph pooling properties outlined in Section 3.3. Methods are divided into hierarchical pooling, adaptive, topology pooling, and overlapping nodes.*

Methods	Hierarchical	Adaptive	Topology	Overlapping Nodes
Graclus [131]	✗	✔	✔	✗
TopK-Pool [50]	✔	✔	✗	✗
SAGPool [77]	✔	✔	✔	✗
DiffPool [143]	✔	✗	✗	✔
SortPool [147]	✗	✔	✔	✗
CliquePool [84]	✔	✔	✔	✗
K-plexPool [9]	✔	✗	✔	✔
Quasi-CliquePool	✔	✔	✔	✔

the node features to perform pooling and ignored the topological structure. The proposed method uses topological structure information to perform pooling operations, and aggregates node features using element-wise sum or max functions, and 4) Overlapping nodes—one node may belong to multiple clusters. It can be observed from Table 3.1 that the proposed pooling method is different from other partitioning-based graph coarsening approaches [84, 9] because Quasi-CliquePool extracts overlapping nodes during pooling operations to preserve the topological structure. Clique pooling, on the other hand, forces a split between two nodes, which destroys the topological relationship between the two nodes. The k-plex pooling finds the overlapping nodes, but it depends on the k-fixed number of adjacent nodes.

3.5 Quasi-Clique Graph Pooling

In this section, we explain the mechanism of Quasi-CliquePool and show how it is implemented in a GNN architecture for graph classification. Section 3.5.1 briefly describe the background for RD with the quasi and maximal cliques. In section 3.5.2, we explain how the quasi-clique method can be used to coarsen the graph. Finally, section 3.5.3 explains the Quasi-CliquePool algorithm to extract the quasi-cliques and maximal cliques.

3.5.1 Replicator Dynamics, Maximal and Quasi Clique

Replicator Equations (REs) are a class of dynamical systems developed to model the evolution of animal behaviour using tools and principles of game theory. The REs have recently been applied with significant success to solve the maximal clique and related problems [100]. This approach formulates the maximal clique problem into a standard quadratic assignment program based on a well-known result from graph theory. We introduce some notations and definitions to represent this concept formally. Initially, the data to be clustered are represented as a graph $G = (V, E)$ with no self-loop, where V and E are sets of vertices and edges, respectively. In our case, the vertices correspond to the graph nodes, and the edges represent the neighbouring relationship between two nodes. We compute the adjacency matrix of G, which is the $N \times N$ non-negative symmetric matrix $A = (A_{ij})$ defined as follows:

$$a_{ij} = \begin{cases} 1, & if(i,j) \in E, \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

The degree of a vertex $i \in V$ relative to a subset of vertices C, denoted by $deg_C(i)$, is the number of vertices in C adjacent to it, that is,

$$deg_C(i) = \sum_{j \in C} a_{ij}. \tag{3.2}$$

When $C = V$ we obtain the standard degree notion, in which case we shall write $deg(i)$ instead of $deg_V(j)$. A subset C of vertices in G is called a clique if all its vertices are mutually adjacent. A clique is said to be maximal if it is not contained in any larger clique, while Quasi-clique is a dense incomplete subgraph of a graph that relaxes the clique constraints. In [100], a one-to-one correspondence between stable points of the RD, local maximizers of a standard quadratic assignment problem and maximal clique is provided. It is thus sufficient to reach an equilibrium point of the RD to get a maximal clique. We used this algorithm in our implementation to extract the maximal and quasi-

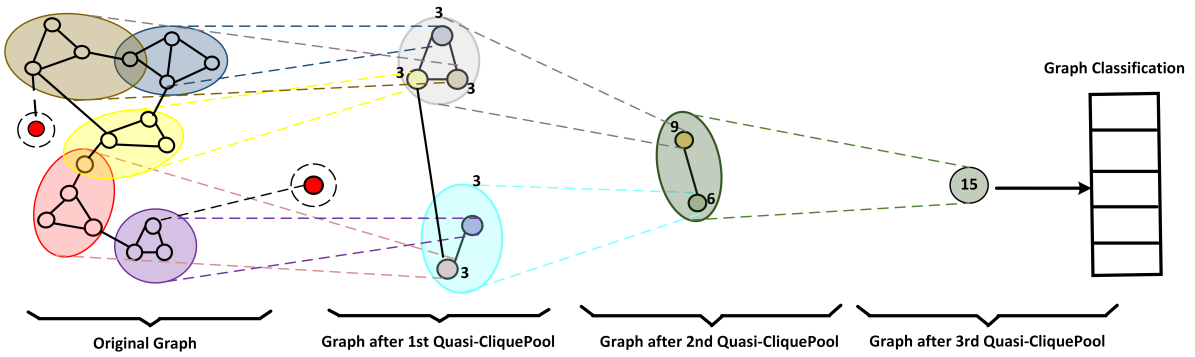


Fig. 3.1: *An example of the Quasi-CliquePool method for a graph. We run a GNN model at each hierarchical layer to obtain embeddings of nodes. After that, apply a Quasi-CliquePool on learned embeddings to get the maximal and quasi cliques and transfer each clique into new nodes in the coarsened graph G′. This process is repeated for n layers, and the final output representation is used to classify the graph. The red nodes are isolated nodes and are removed during the pooling operations. Numbers in bold font represent the number of agglomerated nodes (the highly participating ones) from the previous layer. The nodes are connected in coarsened graph based on node and edge-sharing among cliques.*

cliques. Figure 3.2 shows the graphical representation of how the RD method obtains the maximal cliques and quasi-clique using a density curve. The discrete replicator dynamic is defined as:

$$x_i^{(t+1)} = x_i^{(t)} \frac{(Sx^{(t)})_i}{(x^{(t)})' S(x^{(t)})} \tag{3.3}$$

the Equation 3.3 (for $i = 1, ..., n$) corresponds to the discrete-time version of first-order replicator equations. The RD is a continuous optimization technique that, at convergence, provides the degree of centrality for each node in a cluster; the higher the degree, the more central the node. In [100], the authors introduced an effective strategy (Peel-off) to perform a hard partition of the given data into coherent clusters using RD, with the following steps: 1) find the most participating nodes based on a predefined threshold to obtain a cluster. (i.e., a maximal clique), 2) remove those selected nodes from the similarity graph, and 3) reiterate steps 1 and 2 on the remaining nodes until all nodes have been clustered. The peel-off strategy considers only the highly participating nodes due to the predefined threshold in the RD convergence; however, the low-participating nodes that lie on the border of the cluster might be a good candidate to link with other clusters in the graph. These nodes can be extracted again with other clusters at RD convergence, while the central ones belong only to one cluster. Hence we introduce a new

soft peel-off strategy, a threshold, to find these low participating nodes in a converged RD. Figure 3.3 shows that the central nodes belong only to one cluster, while the border nodes can be linked to other clusters. The run-time complexity of RD is $O(K|V|^2)$, where V is the number of vertices in the graph and K is the number of iterations. A detailed explanation of how we used the RD algorithm and soft peel-off strategy in our implementation is mentioned in section 3.5.3.

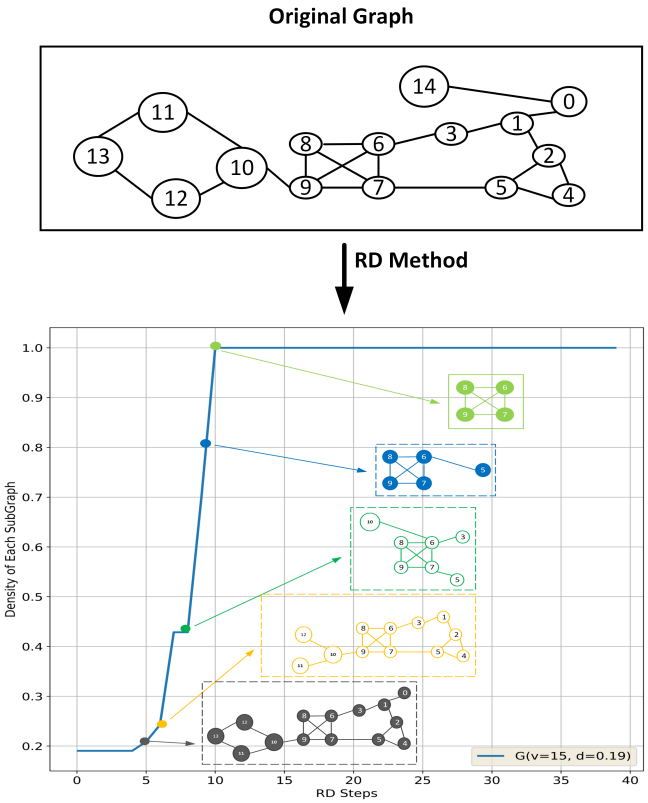


Fig. 3.2: *Illustrations of the RD method to obtain the quasi-cliques and maximal clique. Given a graph with 15 nodes and an initial density* ($d = 0.19$). *We apply the RD method to this graph and map the obtained quasi-cliques and maximal clique onto the density curve at each iteration of the RD. The quasi-cliques and maximal clique are shown in the dotted line and solid line boxes, respectively. It can be seen that the RD method returns the quasi-cliques at each step; however, it returns the maximal clique at convergence when it reaches density* $d = 1.0$.

3.5.2 Graph Coarsening with Quasi-CliquePool

The proposed Quasi-CliquePool method computes the cliques $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ of the input graph (V, E, α, β), and returns a coarsened graph $(V', E', \alpha', \beta')$, such as:

$$V' = V(G') = \{v'_1, v'_2, v'_3, ..., v'_k\}, \tag{3.4}$$

$$E' = E(G') = \{\{v'_i, v'_j\} \mid E(G[C_i, C_j]) \neq \emptyset\}, \tag{3.5}$$

where node v'_i represents the coarsened version of C_i and $E'\{v'_i, v'_j\}$ represents coarsened edge that exists iff there is at least one edge in original graph G connecting a node of C_i with a node of C_j. The node features function $\alpha' : V' \rightarrow \mathbb{R}^d$ aggregates the features that belong to the same clique C_i. We considered the features of maximal clique-based nodes for aggregation since these nodes are highly connected. For relabeling the nodes and edges in the coarsened graph, we defined node features in the following way:

$$\alpha'(v_i)' = \Phi(\{\alpha(v_i) | v_i \in C_i\}), \tag{3.6}$$

$$\beta'(\{v'_i, v'_j\}) = \psi(\{\beta(e) | e \in E(G[C_i, C_j])\}), \tag{3.7}$$

where ϕ and ψ represent the aggregation functions. The element-wise max or sum is a common aggregator function for node features [138]. We used element-wise max or sum aggregators for node features. Our approach is different from other partitioning-based graph coarsening methods [84] because a node may belong to multiple cliques in the proposed method. On the other hand, CliquePool [84] performs a hard partition between nodes, which destroys the topological structural relationship in the cliques[1]. Figure 3.1 illustrates the framework of Quasi-CliquePool. Concretely, we view a GraphConv layer followed by a Quasi-cliquePool layer as a module and name it Quasi-cliquePool GraphConv layer for convenience. The Quasi-cliquePool GraphConv layer takes a graph as an input and outputs a new pooled graph with a new feature matrix and adjacency matrix. Then, the pooled graph is fed into the next Quasi-cliquePool GraphConv layer and ap-

[1]We do not consider edge attributes in this work.

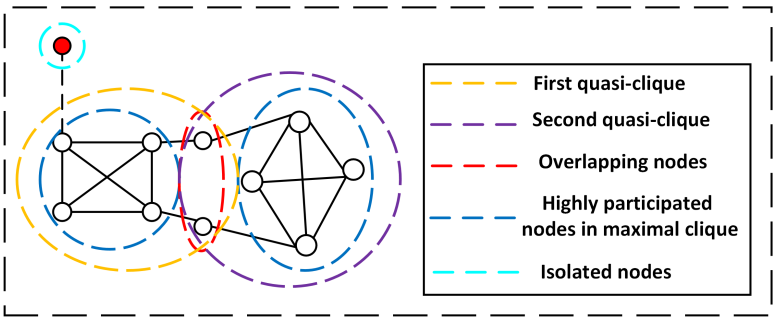


Fig. 3.3: *An example illustrates the clustering process using Quasi-CliquePool. In the first step, the RD method extracts a yellow quasi-cluster with six nodes: four highly participated and two overlapping nodes. The red one is an isolated node removed during the pooling procedure. The overlapping nodes are selected from the yellow cluster using the* soft peel-off *strategy* σ*. We removed only the highly participated nodes from the similarity matrix and iterated the RD method to extract the second purple cluster with overlapping nodes.*

plies a readout function on it. In the last step, we get the final graph representation by adding the embeddings of each graph layer, and a multi-layer perceptron model is applied to this representation to classify the graph.

3.5.3 Quasi-CliquePool Algorithm

In this section, we propose a Quasi-CliquePool algorithm for graph classification, whose pseudocode is shown in Algorithm 1, that extracts the quasi-cliques and maximal cliques of a given graph. Algorithm 1 performs two main tasks: 1) partitions the graph and assigns the nodes of the original graph G to k different cliques, and 2) each clique is transformed into a new node in the coarsened graph G'. The core of our method is inspired by the original RD algorithm for the maximal clique of Pelillo et al. [100]. Algorithm 1 receives a graph $A \in \{0,1\}^{N\times N}$ and $X \in \mathbb{R}^{N\times d}$ as input and returns the list of all possible quasi-cliques C. In the first step, we used the Euclidean pairwise distance method, and the gaussian kernel [146] function to build the similarity matrix of the given graph (Equation 3.8). The kernel function basically tells the model how similar two data points are (x_i, x_j). The affinity between a pair of points can be defined as

$$S_{ij} = \exp\left(\frac{-d^2(x_i, x_j)}{\rho_i \rho_j}\right), \tag{3.8}$$

where $d(x_i, x_j)$ is the euclidean distance between the vectors x_i and x_j and ρ_i ρ_j are the local scaling parameters computed with [146]. In the next step, the RD method uses this similarity matrix as an input and provides a characteristic vector X_N that contains the probability value of the participation of each node in a cluster; the higher the value, the central the nodes as a maximal clique, and the lower the value, the border the nodes. Then a γ threshold is applied to the characteristic vector X_N to extract the highly participating nodes as a maximal clique. Here we are interested in extracting the low participating nodes that lie on the clique's border because these nodes might be good candidates for overlapping nodes with other cliques in the graph. We introduced a new soft peel-off strategy, a filter operation that finds border (overlapping) nodes using a σ threshold and assigns them again in characteristic vector x for the next convergence of the RD. So in this way, Algorithm 1 iteratively selects all the possible quasi-cliques by extracting the nodes in set U. Nodes that remain isolated during the pooling operations are removed from the graph. Moreover, the aggregation procedure of the node features (Equation 3.6) contributes similarly to the respective clusters, and element-wise sum or average readout functions are used to aggregate the node features in the coarsened graph. In the next pooling layer, we transformed each clique into a new node and connected two cliques (Equation 3.7) if they have a common edge or node in the original graph G. We illustrate our proposed Quasi-CliquePool in Figure 3.1, where we performed quasi-clique pooling on a graph with 19 nodes and obtained a new graph G' at the first layer of Quasi-CliquePool with 5 nodes.

Algorithm 1: Quasi-CliquePool

Input: Given a graph G as $A \in \{0,1\}^{N \times N}$ and $X \in \mathbb{R}^{N \times d}$

Output: List of all possible Quasi-Cliques $\mathcal{C}$ of G

$\mathcal{C} \leftarrow \emptyset$;

$S \leftarrow$ build a similarity matrix (Equation 3.8);

$U \leftarrow V(G)$;

while $U \neq 0$ **do**

  $X_N \leftarrow \mathrm{RD}(S)$; `// characteristics vector` X_N `(Equation 3.3)`

  $C_i \leftarrow \mathrm{filter}(X_N, \gamma)$; `// extracts the quasi-clique`

  $L \leftarrow \mathrm{filter}(C_i, \sigma)$; `// selects the low participated nodes quasi-clique (overlapping nodes) from` C_i

  $\mathcal{C} \leftarrow \mathcal{C} \cup C_i$;

  $C_i \leftarrow C_i \setminus L$; `// remove the selected lowest value nodes from` C_i

  Remove C_i from the similarity matrix S;

$\mathcal{C} = \{C_1, \ldots, C_k\}$;

3.6 Result and Discussion

This section evaluates the superiority and effectiveness of Quasi-CliquePool in comparison to other contemporary graph-based classification approaches. Section 3.6.1 Section provides a brief overview of the datasets utilized in the research. Section 3.6.2 explains the baseline methods used to compare the results and configuration of Quasi-CliquePool and baseline methods. In section 3.6.3, we compare Quasi-CliquePool results with state-of-the-art graph classification approaches. Finally, section 3.6.4 presents the ablation study.

3.6.1 Dataset Setup

To verify the performance of Quasi-CliquePool in learning complex hierarchical graph structures in different domains, it tested on a variety of large benchmarks that are com-

monly used in graph classification tasks [67]. This study used bio-informatics datasets including DD [47], Protein and Enzymes [21, 42], and the molecule datasets Mutag [105], NCI-1, and NCI-109 [129]. All datasets are retrieved from the TU-Dortmund collection [92]. Table 3.2 shows the statistics of the datasets, most of the datasets are relatively large-scale with different sizes of graphs and hence suitable for evaluating deep graph models.

Table 3.2: *Statistics of datasets.*

Datasets	Total Graph	Average Nodes	Average Edges	Classes
ENZYMES [42]	600	32.63	62.14	6
PROTEINS [21]	1,113	39.06	72.82	2
D&D [47]	1,178	284.32	715.66	2
NCI-1 [129]	4,110	29.87	32.30	2
NCI-109 [129]	4,127	29.68	32.13	2
MUTAG [105]	188	17.93	19.79	2

3.6.2 Baselines and Experimental Settings

To compare the performance of graph classification, we consider GNNs-based baselines combined with different state-of-the-art pooling methods. The next section briefly describes these baseline methods with experimental settings.

Graph Neural Network Methods. This chapter considers three GNN architectures to test the proposed Quasi-CliquePool method. (1) GCN [72] is a convolutional neural network that learns node representations by aggregating and propagating information from neighbours. (2) GraphSage [56] introduces the inductive framework, which calculates node embedding by aggregating and sampling features from local neighbours. (3) GraphConve [91] proposed k-dimensional GNNs that can take high-order graph structures and are useful in analysing social networks and molecule graphs.

Competitors. This chapter compares Quasi-CliquePool with five state-of-the-art hierarchical pooling techniques: Top-k pooling [50], SAGEpool [77], DiffPool [143], clique

pooling [84], and k-plex pooling [9] and two global pooling methods: Sort pooling[147] and Graclus[131]. For all baselines and quasi-clique poling, we employed the same hyper-parameter search strategy. The hyperparameters are summarized in Table 3.3.

For all the pooling and GNN baselines, we consider the accuracy scores reported by the original authors. In cases where baseline techniques did not publish require classification scores, we used the original authors' code (if available) with the same hyperparameters setting mentioned in the original papers. In our experiments, we used the GraphConv architecture for Quasi-CliquePool, since we achieved effective and superior performance with this architecture compared to the standard graph convolutional model. We used the Graphconv "add" variant, and after each Graphconv layer, we added a Quasi-CliquePool layer. A global readout function is applied after every layer of Quasi-CliquePool with dropout (ratio 0.5). A total of three Quasi-CliquePool layers are used for the datasets. Next, the ReLu activation function is applied after every convolutional layer. A softmax function is used to classify the graph in the last step. We randomly split each dataset into three parts: 80 percent for training, 10 percent for the validation set, and 10 percent for the testing set. We repeated this random splitting process 10 times to get more stable performance and reported the average performance. We used PyTorch to implement the Quasi-CliquePool and the Adam optimizer to optimize the model. Table 3.3 shows the hyperparameter list. For all GNNs and pooling baselines, we used the official PyTorch published code.[2]

3.6.3 Performance on Graph Classification

Table 3.4 demonstrates the test accuracy of Quasi-CliquePool on bio-informatics and molecules benchmarks. To summarize the results, we have the following observations:

- First, we can observe from the results that Quasi-CliquePool consistently outperforms other baselines in most datasets. For example, in the molecule datasets, quasi-clique pooling achieves 1.10% improvement compared to the k-plex best baseline in NCI-1, which is 10.13% improvement over a Graph Convolutional Network

[2] $https://github.com/pyg-team/pytorch\_geometric/tree/master/benchmark/kernel$

Table 3.3: *The hyper-parameters setting. We applied three graph convolutional layers and three quasi-clique pooling layers. The pooling ratio is used only for Top-k pooling, SAGPool, and DiffPool.*

Model	**Hyper-Parameters**	**Values**
All	GNNs	GCN, GraphSAGE, GraphConv
	layers	2, 3
	learning rate	1e-2, 1e-3, 1e-4
	weight decay	1e-2, 1e-3, 1e-4, 1e-5
	TopK-Pool, SAG-Pool	0.8
DiffPool	pooling ratio	0.25
RD	ϵ, γ	2.0e-4, 1.0e-7
Quasi-CliquePool	σ	1.876e-03

(GCN) with no hierarchical pooling mechanism. Quasi-CliquePool also achieves 2.76% improvement over the k-plex pooling in the Proteins dataset, and the overall improvement is 8.67% over the GCN model.

- In most of the datasets, our quasi-clique pooling method obtains better performance than both hierarchical and global pooling methods. Quasi-CliquePool almost achieves 2.5% overall improvement over hierarchical baselines, including DiffPool, clique, k-plex, and Top k pooling in bio-informatics datasets. At the same time, the quasi-clique method almost achieves 8% improvement over global pooling approaches (Graclus and Sortpooling).

- Being consistent with existing studies' findings [85, 151], we can see from Table 3.4 that GNN architectures without pooling modules are not able to achieve promising results because they ignore hierarchical graph information while summarizing the node representations globally. So this also proves that GNNs need a graph pooling layer for graph classification tasks.

- We also note that the hierarchical-based pooling methods achieved relatively better results than global methods, which further demonstrates the effectiveness of the hierarchical pooling operations. Both the SAGPool and Top-k pooling methods

Table 3.4: *Test accuracy on the classification of molecules and bio-informatics benchmarks. The bold and underlined text highlight the highest and second-highest accuracy scores, respectively.*

Classification	Baselines	ENZYMES	PROTEINS	D&D	NCI-1	NCI-109	MUTAG
GNNs	GCN [72]	28.68%	70.01%	71.42%	70.24%	68.33%	72.30%
	GraphSAGE [56]	31.73%	71.37%	71.70%	73.36%	72.30%	74.08%
	GraphConv [91]	33.71%	72.43%	72.31%	74.70%	73.22%	77.97%
Pooling	Graclus [131]	28.51%	71.35%	72.45%	74.25%	72.32%	76.64%
	TopK-Pool [50]	31.64%	<u>77.25</u>%	**82.43**%	73.30%	72.30%	74.75%
	SAGPool [77]	32.68%	70.04%	76.19%	74.18%	<u>74.04</u>%	77.72%
	DiffPool [143]	**62.53**%	76.25%	<u>80.64</u>%	76.40%	74.29%	81.09%
	SortPool [147]	-	75.54%	79.37%	74.48%	72.31%	<u>84.12</u>%
	CliquePool [84]	42.17%	73.86%	74.88%	78.83%	-	-
	K-plexPool [9]	43.33%	75.92%	77.76%	<u>79.01</u>%	-	-
Proposed	Quasi-CliquePool	<u>45.01%</u>	**78.68%**	75.30%	**80.11%**	**76.30%**	**84.88**%

perform poorly on the ENZYMES dataset. The possible reason may be limited training examples per class, resulting in overfitting in GNN. However, DiffPool achieves superior performance in the ENZYMES dataset, and the proposed Quasi-CliquePool achieves the second-best accuracy in this dataset. In addition, the Top-Kpool obtains the best performance on the D&D dataset, and DiffPool obtains the second-best performance. The quasi-clique pooling method performs badly on the D&D dataset because it has very large, noisy, and sparse graphs.

- In comparison to the K-Plex and clique pooling methods, our method achieves the best performance on all six datasets, as shown in Table 3.4. Such observations demonstrate the overlapping nodes information in graphs is useful for graph pooling. And overall, our Quasi-CliquePool performs best on four out of six datasets.

3.6.4 Ablation Study

In this section, we conducted the experiments to verify the performance of the proposed method by varying the sensitivity of many significant hyperparameters. Next, we integrate our pooling method into various GNN architectures to investigate its effect. We

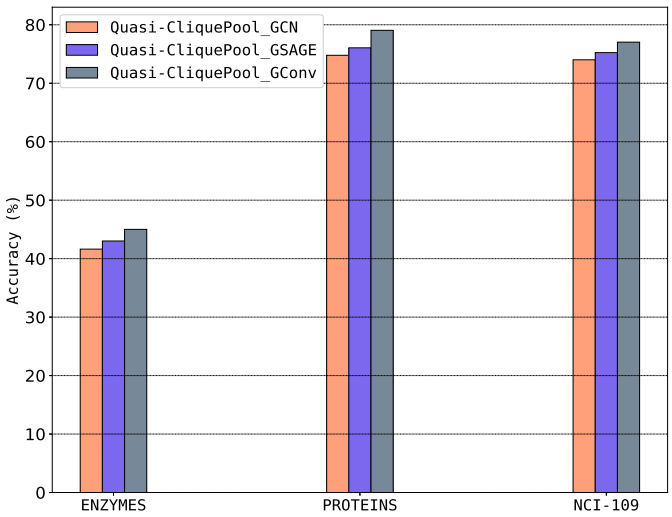


Fig. 3.4: *Quasi-CliquePool performance with different GNN architectures.*

also investigate the performance of quasi-clique pooing with different readout functions.

Quasi-CliquePool and Graph Neural Network Architectures

As previously mentioned, the proposed Quasi-CliquePool can integrate into various GNNs architectures. We integrate Quasi-CliquePool into the three most widely used graph convolutional models, including GCN, GraphSAGE, and GraphConv. These models test on three datasets (Protein, Enzymes, NCI-109), which cover large and small graph datasets with multiple classes. Figure 3.4 shows the performance of the three Quasi-CliquePool variants. It can see the Quasi-CliquePool_GraphConv achieves the highest accuracy on all three datasets, specifically on the Protein dataset. One can also see that the performance of Quasi-CliquePool_GraphConv on the Enzymes dataset is also better than other variants, so it shows our proposed model can get good results on multi-classes datasets.

Hyper-Parameter Analysis

This section further investigates the sensitivity of some important hyper-parameters on Quasi-CliquePool. In detail, we investigate how the GNN layers L and graph representation dimension d affect graph classification results. As shown in Figure 3.5, Quasi-

CliquePool obtains the highest accuracy when setting k = 3 and d = 64, respectively. It can be observed that the accuracy presents a slight increase trend with the dimension d increasing in both datasets. This is because the higher dimensional representation space makes classification tasks easy. One can also see that when we increase the neural network layers, the accuracy is also increasing, but too large layers L will hurt the model's performance due to over-smoothing [79].

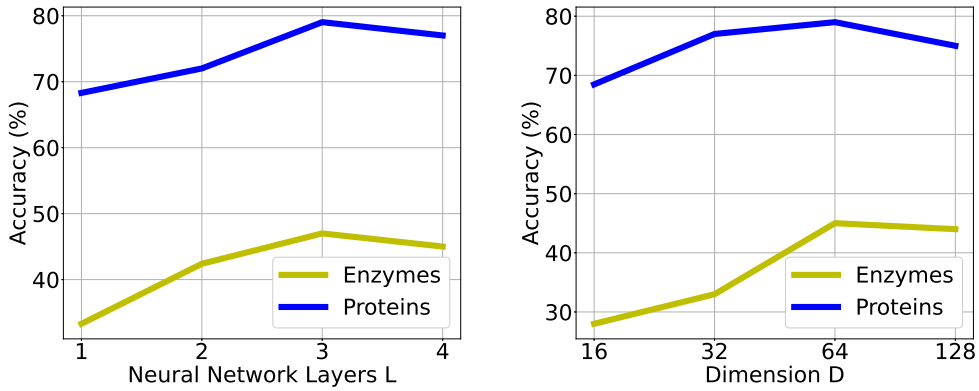


Fig. 3.5: *Test accuracy curves on Protein and Enzymes with different values of L and d*

Readout Functions

To investigate how the different readout functions affect the performance, we used three readout functions, *sum, avg, and max* to aggregate node features, which are denoted as Quasi-CliquePool-sum, Quasi-CliquePool-avg, and Quasi-CliquePool-max, respectively. These readout-based quasi-clique models evaluate on three datasets (Protein, Enzymes, NCI-109). As shown in Table 3.5, Quasi-CliquePool-Max and Quasi-CliquePool-Avg achieve better performance than Quasi-CliquePool-Sum. Quasi-CliquePool-max obtains the highest accuracy than Quasi-CliquePool-Avg, which is consistent with existing research work [9, 84]. This observation highlights the significance of readout functions in aggregating nodes in pooling operations.

Table 3.5: *Quasi-CliquePool performance with various readout functions.*

Readout Functions	**ENZYMES**	**PROTEINS**	**NCI-109**
Quasi-CliquePool-Sum	43.20%	75.89%	78.15%
Quasi-CliquePool-Max	**45.01**%	**78.68**%	**80.11**%
Quasi-CliquePool-Avg	44.01%	77.30%	77.30%

3.7 Conclusion

In this chapter, we designed a novel graph pooling technique, Quasi-CliquePool, for graph classification using the RD algorithm. The proposed Quasi-CliquePool method has the following specifications: it exploits the topological structure of the graph, extracts the complex hierarchical structure of graphs, does not require a-priori knowledge of the hierarchy, and can be integrated into several GNN architectures. This chapter also introduced a soft peel-off strategy to find the overlapping nodes of the graph in the clustering procedure. To demonstrate the superiority of Quasi-CliquePool for graph classification, it tested on six datasets from two domains: molecules and bio-informatics. The proposed method obtained the best results in four datasets, demonstrating our model's effectiveness.

Chapter 4

Dominant Sets: A Multi-View Approach to Graph Pooling

"Authentic perspicuity comes not from a single view but from the intersection of viewpoints, each uncovering a segment of the immense puzzle."

— Unknown

4.1 Preamble

In this chapter, we introduced a novel Dominant Set-based clustering pooling method that analyses the graph's overall architecture and connectivity patterns, finds all potential clusters using edge weight information, and generates a coarser graph view. Additionally, we designed a fusion-view attention layer to fuse the different sources of information, which allows our pooling method to extract and integrate global and local structures and node features simultaneously. The research and findings discussed in this chapter are based on the following paper:

- **Waqar Ali**, Sebastiano Vascon, Thilo Stadelmann, and Marcello Pelillo. "Dominant Set Multi-View Graph Pooling for Graph Classification"; Submitted to Neural Networks Journal 2024 (under review).

The author has the following contributions:

- **Developing** the overall framework of the algorithm.
- **Writing** the majority of the code.
- **Performing** the overall experiments.
- **Writing** a considerable part of the paper.

4.2 Introduction

GNNs have recently revolutionized the analysis of graph-structured data by leveraging message passing mechanisms to aggregate neighborhood information, generating node embeddings, which are used to perform various graph-related tasks from node classification to link prediction [157, 103]. In the context of graph classification, graph pooling is an essential operation in GNNs for learning the representation of an entire graph. It maps the nodes or subgraphs into a compact representation, highlighting significant graph structures while improving computational efficiency [94]. Early graph pooling methods use sum or average aggregation functions to generate graph-level representations for GNNs [12, 116]. Despite being straightforward, these methods often overlook the contextual and hierarchical information within graphs. To address these limitations, recent studies have introduced hierarchical pooling methods that preserve graph substructures through local and global topological information [77, 30, 84]. These hierarchical methods provide a more comprehensive graph representation by clustering or selecting informative nodes layer by layer.

Node cluster pooling methods capture the connectivity patterns of the entire graph by grouping similar nodes into clusters and transferring each cluster into a single node to generate a coarsened graph. Ying et al. [143] developed the first DiffPool that learns a soft cluster assignment matrix for nodes using the GNN, which contains the probability values of nodes being assigned to clusters. DiffPool mainly focuses on the node features to extract the clusters and depends on a predefined cluster ratio. Similarly, the clique pooling method [84] targets global topological structures by identifying all max-

imal cliques within the graph. Methods [4] and [9] further improve clique pooling to extract the overlapping nodes between two cliques. In node selection methods, the goal is to create a pooled graph by learning the significance scores of each node and then selecting a subset of nodes with high scores [140, 77, 30]. For example, Gao et al. [50] proposed a Top-k pooling, which employs scalar projection values of node features to select the most important nodes. SAGPool [77] further enhanced the performance of Top-k pooling using self-attention weights, and MAC [140] used attention weights with a convolutional neural network to evaluate the importance of nodes. Furthermore, MVPool [152] employs a multi-view scoring strategy and uses the attention mechanism for integrating the different views to generate more robust subgraphs. However, existing cluster pooling approaches mainly perform analysis on unweighted graphs, overlooking the nuanced dynamics of weighted graphs. Edge weights in the graphs represent the similarity between nodes, which is significant for analyzing the graph's hierarchical structural information to perform downstream classification tasks. Additionally, most of the traditional graph cluster methods often require a predefined cluster ratio to guide the pooling process. Furthermore, the above-mentioned pooling methods usually fail to integrate the graph's comprehensive multi-view contextual information, resulting in generating less robust graph representations.

To address the challenges in existing graph pooling, this chapter introduces a novel Dominant Set Multi-View Pooling method (DSMVPool), which enhances the performance of graph classification tasks by fusing different contextual information, including both global and local topological information, as well as node features and edge weights. Specifically, we develop a node cluster method using the dominant set concept [98] to capture the graph's global topological information. Unlike existing graph cluster pooling approaches, our dominant set utilizes edge-weighted graphs and finds all potential clusters without depending on a predefined cluster ratio, generating a more robust graph coarsening representation $G_{coarser}$. We also generate two pooled views of the input graph G_{local} and $G_{feature}$ by extracting the most important nodes based on the graph's local topological information and node features, respectively. Furthermore, we design a fusion-view atten-

tion convolution layer to fuse $G_{coarser}$ with G_{local} and $G_{feature}$. Ultimately, we generate the final pooled graph by aggregating the fused graph representations. Specifically, our contribution is four-fold:

- We propose a novel Dominant Set Multi-View Graph Pooling method, which simultaneously captures and integrates local topological information, coarser graph structures, and node features.
- For the first time, we use the Dominant Set clustering method to develop graph pooling. This has the advantage of exploiting edge weights (neglected by most of the pooling methods) and avoiding an a-priori fixed number of clusters, resulting in a more expressive graph coarsening.
- Furthermore, we design an attention-fusion-view convolution layer that refines the graph representations by integrating the coarser graph with local topological structures and node features-based pooled graphs.
- We conduct extensive experiments showing that DSMVPool enhances average accuracy by 1.31%, 1.51%, 0.62%, and 1.14% in chemical molecules, social networks, bio-informatics, and computer vision-based graph classification benchmarks, respectively, compared to the state-of-the-art pooling approaches.

4.3 Related Work

The current graph pooling methods are classified into two types based on their design strategy: global and hierarchical pooling.

Global Pooling Approaches: Global pooling usually adopts summation or average operations to integrate the embeddings of all nodes, resulting in a single vector representation for the entire graph. In Set2Set [127], the authors use the long short-term memory model to aggregate the node embeddings and perform global pooling. The DGCNN [147] model first sorts the node embeddings and subsequently generates the graph representation by combining certain node embeddings. Graph topological-based pooling procedures

are introduced in [145, 104], where Graclus and graph coarsening techniques are used as pooling modules [62]. Global approaches perform pooling operations based only on node attributes, potentially resulting in the loss of hierarchical information.

Hierarchical Pooling Approaches: Hierarchical graph pooling methods aim to learn a hierarchical representation via building hierarchical GNNs. Based on designing properties, the hierarchical methods can be grouped into two main classes: sparse node selection and node cluster. The node selection pooling algorithms calculate the importance of nodes based on their features or the structural information of the graph and retain the nodes with the highest scores. For example, Top-k pooling [50], SAGPool [77], and AttPool [63] select the most significant nodes based on node attributes or attention scores to form a pooled graph for the next input layer. Jinheon et al. [140] proposed MAC pooling that incorporates multiple strategies to calculate the importance of nodes and use an attention mechanism to update node representations. MVPool [152] further improves the node ranking using different contextual graph information and develops a structure learning method to refine the graph structure for the pooled graph. These methods are more effective as they only need to calculate an important score for each node. However, they do not perform node aggregation and neglect to consider the graph topological information during the pooling operation. EdgePool [85] generates a pooled graph by integrating the edges of a given graph. It lacks flexibility as it can only reduce the number of nodes by half with each iteration.

The graph cluster method globally assigns all nodes to a number of clusters but requires setting the number of clusters and then obtaining the basis vector of each cluster. This strategy better ensures the completeness of feature information due to the basis vectors of each cluster containing all node information. In [143], the authors proposed DiffPool to learn a soft assignment matrix using graph neural networks and mapping nodes to a set of clusters. In [84], the authors introduced clique-based graph pooling to capture the overall topological structures of the network effectively. This is achieved by dividing the graph into its possible cliques. Methods [4, 9] enhance the clique pooling technique to retrieve the overlapping nodes shared by two cliques. Further, MuchPool [43] combines

Table 4.1: *Comparison of graph pooling methods.*

Properties	TopkPool [50]	SAGPool [77]	DiffPool [143]	CliquePool [84]	MuchPool [43]	**Our**
Sparse	✔	✔	✗	✗	✔	✔
Node aggregation	✗	✗	✔	✔	✔	✔
Graph clustering	✗	✗	✔	✔	✔	✔
Flexible number of clusters	✗	✗	✗	✗	✗	✔
Edge weights support	✗	✗	✗	✗	✗	✔

DiffPool with Top-k pooling to capture graph local and global topological information and node features. However, most current approaches only use a limited amount of contextual information from graphs, disregarding collaboration and input from many sources of information. Furthermore, the node cluster pooling methods stated above only focus on unweighted graphs, which hinders their ability to adapt the weighted graphs during the clustering implementation process. This limitation may constrain the potential use cases for pooling operations. Thus, this chapter introduces a novel dominant set multi-view pooling and uses edge weights to extract the clusters in the weighted graphs without relying on a predefined cluster ratio. Furthermore, our method designs a attention-fusion-view convolution layer to fuse the coarser graph with pooled graphs to generate more robust graph representations for graph classification tasks. Table 4.1 compares the attributes of the above graph pooling operators with DSMVPool. The table shows that our technique consists of all properties.

4.4 Proposed Methodology

The proposed DSMVPool mainly uses three views to perform graph pooling operations to learn different contextual information of a graph and then integrate the results of these three views. The first step of the Figure 4.1 employs three graph pooling methods to deeply understand the graph's local and global topological structures and the node features. In this initial step, we generate two pooled views of the graph G_{local} and G_{feature} and one coarser view G_{coarser}. The second step implements an attention-fusion-view convolution layer that fuses the G_{coarser} with G_{local} and G_{feature}. The last step

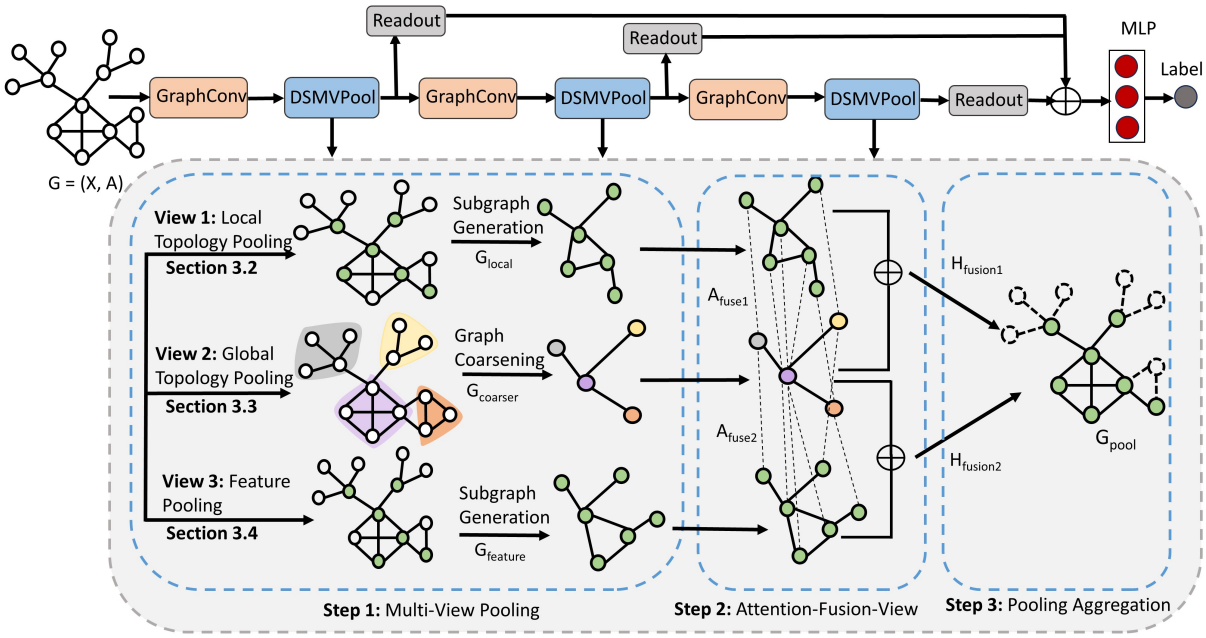


Fig. 4.1: *The architecture of the proposed DSMVPool. Step 1 applies three graph pooling methods that capture the different contextual information of the graph and generate two pooled views, G_{local} and $G_{feature}$, and one $G_{coarser}$ view of the graph. Step 2 implements a fusion-view layer to fuse three subgraphs, such as $G_{coarser}$ with G_{local} and $G_{feature}$ and generate two new fusion feature embeddings $H_{fusion1}$ and $H_{fusion2}$. The last step generates the final pooled graph by aggregating the results of $H_{fusion1}$ and $H_{fusion2}$.*

aggregates the results of views 1 and 3 to generate the final pooled graph G_{pool}. The following sections provide more detailed explanations of each step.

4.4.1 Local Topology Pooling (View 1)

This pooling view learns the graph's structural importance by ranking nodes according to their local neighborhood connectivity. Given a graph $G = (V, E, X)$, we reduce the set of nodes considering the most important ones while preserving the original connectivity, hence generating the pooled graph G_{local}. In this context, the notion of a node's neighborhood can be defined as the explicit connections within a graph or the affinities within node embeddings. To assess the node's importance, we use a GAT layer [125] to capture the local topological structures. Mathematically, the significance of each node is derived through the following formulation:

$$L = \text{attention}(GAT(X, A)); \quad L_{\text{idx}} = \{i|\ L_{\text{i}} > p\} \tag{4.1}$$

where L_i shows the attention score for a node i, A denotes the adjacency matrix of the graph and X represents the initial node features. In L_{idx} we preserve only those nodes having an attention score greater than the pooling ratio p. The pooled graph $\text{G}_{\text{local}} = (V_L, E_L, X_L)$ is then defined as $V_L = \{v_i | i \in L_{\text{idx}}\}$ is the set of nodes, $E_L \subseteq V_L \times V_L \cap E$ hence we use the original graph connectivity, and the set of features $X_L = \{X_i | i \in L_{\text{idx}}\}$.

4.4.2 Global Topology Pooling (View 2)

This pooling view aims to generate a coarser graph view $\text{G}_{\text{coarser}}$ by capturing the graph's global topological information. Current cluster pooling techniques like DiffPool [143] primarily use unweighted graphs to perform pooling operations, neglecting edge weights and typically requiring a predefined cluster ratio to find clusters within a graph. Edge weights play a crucial role in understanding the topological structures within graphs, such as in a molecular graph where edges indicate chemical bonds of different strengths or in a social network graph where edges represent the frequency of interactions among individuals [10]. In these cases, edge weight information is significant for describing the graph's topological structures and can affect how clusters form. Inspired by a dominant set-based clustering method [98] that combines concepts from graph theory and evolutionary game theory to identify clusters in data using edge weights, we adopt this robust concept for graph pooling. Therefore, we design a dominant set cluster pooling method to capture the graph's global topological information and identify all possible dominant clusters. Following that, we transfer each dominant cluster into a single node (supernode) and aggregate the feature within each cluster to generate $\text{G}_{\text{coarser}}$. Next, we provide a detailed explanation of the dominant set notion and describe how we used this concept in graph pooling to generate the graph coarsening.

Dominant Set Clustering

The Dominant Set (DS) clustering extends the concept of identifying maximal cliques to edge-weighted graphs. In this context, the DS method is used to coarsen the input graph grouping nodes having similar features. The DS clustering is different from other graph

coarsening approaches, such as DiffPool and Mincutpool, because it is not dependent on any predefined cluster ratio or a-priori number of clusters, and it works on weighted graphs. Therefore, DS pooling has strong advantages over the competitors, allowing the structures to emerge spontaneously from the graphs, resulting in more natural, expressive, and representative clusters. We then construct an undirected edge-weighted graph $G = (V, E, w)$ without self-loops, where the nodes V correspond to the graph's nodes, represented by feature vectors. The edges $E \subseteq V \times V$ are the pairwise relations between nodes and their weight function $\omega : E \to \mathbb{R}_{\geq 0}$ calculates pairwise similarities. The $N \times N$ symmetric adjacency matrix $A = (a_{ij})$ summarizes G:

$$a_{ij} = \begin{cases} w(i,j) & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

Typically, every clustering method is expected to exhibit two essential properties: high intra-cluster homogeneity and low inter-cluster homogeneity. These properties are crucial for effectively segregating and grouping objects. They directly influence the combinatorial formulation of DS, as detailed in [98]. Pavan et al., established an intriguing connection between clusters, dominant sets, and local solutions of the following quadratic problem [98]:

$$\begin{aligned} \text{maximize} \quad & \mathbf{h}^T A \mathbf{h} \\ \text{subject to} \quad & \mathbf{h} \in \triangle^n \end{aligned} \tag{4.3}$$

where A is the similarity matrix of the graph and $\mathbf{h}$ is the so-called *characteristic vector* which lies in the n-dimensional simplex $\triangle^n$, that is, $(\sum_i \mathbf{h}_i = 1, \forall i \; h_i \geq 0)$. If $\mathbf{x}$ is a strict local solution of (4.3) then its support $\sigma(\mathbf{h}) = \{i \in V | h_i > 0\}$ is a dominant set [97]. In order to extract a DS, a local solution of (4.3) must be found. A well-known method to solve this problem is to use a result from evolutionary game theory [136] known as

replicator dynamics (RD) (see Equation 4.4).

$$h_i(t+1) = h_i(t)\frac{(A\mathbf{h}(t))_i}{\mathbf{h}(t)^T A\mathbf{h}(t)} \tag{4.4}$$

RD is a dynamical system that conducts a selection process on the elements of the vector $\mathbf{h}$. Upon convergence of Equation 4.3 ($||\left(\mathbf{h}(t) - \mathbf{h}(t+1)\right)||_2 \leq \epsilon$), specific elements will emerge ($h_i > 0$) while others will vanish ($h_i = 0$). Convergence of the process is assured when the matrix A is non-negative and symmetric. The dynamical system commences at the barycenter of the simplex, and its elements are updated using Equation 4.4. At convergence, a dominant set is identified using the support of $\mathbf{h}$, the selected nodes are removed from the graph (referred to as the "peeling-off" strategy), and the process iterates again on the remaining nodes until all nodes are assigned to a cluster.

Dominant Set Pooling

Given an unweighted graph $G = (V, E, X)$ with associated features X for each node, we first construct a weighted version of it $G = (V, E, X, W)$ and then extract all the clusters using the DS method. Given two nodes i and j, we compute the edge weights using the cosine similarity of node features $X_i \in X$ and $X_j \in X$ and refine the similarities through a Gaussian kernel.

$$s_{ij} = \frac{X_i \cdot X_j}{||X_i||||X_j||}; \quad W_{ij} = \exp\left(-\frac{(1-s_{ij})^2}{2\sigma^2}\right) \tag{4.5}$$

The σ is a scale parameter (after several experiments, we set it as 1.0).

Given the weight W we used the DS's peel-off strategy to extract all dominant set clusters $C = \{c_1, c_2, ..., c_k\}$ from the graph $G = (V, E, X)$ and generate the $\mathrm{G}_{\mathrm{coarser}} = (\hat{V}, \hat{E}, \hat{X})$ representation with updated features. Here, $\hat{V} = \{\hat{v_1}, \hat{v_2}, ..., \hat{v_k}\}$ is the set of supernodes representing the k clusters, $\hat{E}$ is the new set of edges connecting supernodes, and $\hat{X}$ are the features associated to each supernodes $\hat{V}$. These sets are defined as follows:

$$\hat{X}_i = \alpha(\{X_v | v \in c_i\}) \tag{4.6}$$

$$(c_i, c_j) \in \hat{E} \quad \text{iff} \quad \exists v \in c_i \quad \text{and} \quad u \in c_j | (v, u) \in E \tag{4.7}$$

where α represents a node features aggregation functions (max or average). An edge between supernodes i and j is added to $\hat{E}$ iff an edge already insists between a pair of nodes belonging to the clusters i and j in the un-coarsened graph G.

4.4.3 Node Feature Pooling (View 3)

Beyond the structural information, graphs often come with node features that can broadly describe their properties. For example, in chemical molecule graphs, node features depict the type of atoms essential for predicting the graph properties. Therefore, it is a valuable source for highlighting the significance of a node inside a graph. So, the main goal of this pooling view is to select the most important nodes based solely on their feature values. Given a graph $G = (V, E, X)$, we reduce the set of nodes considering the most important ones while preserving the original connectivity, hence generating the pooled graph $\mathrm{G_{feature}}$. For this purpose, we directly use a Multi-Layer Perceptron (MLP)[1] to calculate the node importance score directly from node features:

$$F = (MLP(X)); \quad F_{\mathrm{idx}} = \{i |\ F_{\mathrm{i}} > p\} \tag{4.8}$$

where F_i shows the feature score for node i and X represents the initial feature matrix. In *view1* pooling, Equation 4.1 already considered the local topology and node features. However, the pooling operations can cause the graph to become sparse and result in isolated nodes (not connected to any other node). This can negatively impact the message passing in the subsequent layers. As a result, the information from these isolated nodes cannot be aggregated by Equation 4.1. However, Equation 4.8 for node feature learning is independent of structural information, meaning that the presence or absence of edges between nodes does not impact the learning process. Consequently, in a sparse graph, this pooling view layer can effectively learn the information of these isolated nodes. The

[1]The MLP is composed of two linear layers of dimensions 64 and 1, respectively, interleaved with a ReLU function.

pooled graph $\mathrm{G}_{\text{feature}} = (V_F, E_F, X_F)$ is then defined as $V_F = \{v_i | i \in F_{\text{idx}}\}$ is the set of nodes, $E_F \subseteq V_F \times V_F \cap E$ hence we use the original graph connectivity, and the set of features $X_F = \{X_i | i \in F_{\text{idx}}\}$.

4.4.4 Fusion-View Attention Convolution

In the first step, illustrated in Figure 4.1, we generate two pooled views and one coarser view of the input graph G_{local}, G_{feature}, and G_{coarser}. Next, we need to fuse these three pooled graphs, each reflecting the graph's distinct contextual aspects. Motivated by MuchPool [43], we developed an attention-fusion-view message passing operation to fuse G_{local} with G_{coarser}, and G_{feature} with G_{coarser}. The calculation of the fusion-view process is as follows:

$$H_{\text{view}} = \sigma\left([H_{\text{view}} + A_{fuse} \cdot H_{\text{coarser}}] \cdot W\right) \tag{4.9}$$

where $H_{\text{view}} \in \mathbb{R}^{p \times d}$ denotes the node embedding matrix of the G_{local} or G_{feature}, $H_{\text{coarser}} \in \mathbb{R}^{k \times d}$ represents the node embedding matrix of G_{coarser} (see Equation 4.6 and $A_{fuse} \in \mathbb{R}^{p \times k}$ denotes the attention scoring relationship matrix between G_{coarser} and G_{local} or G_{feature} pooled graphs, where p and k are the node numbers in the two pooled graphs and G_{coarser}, respectively.

In particular, to obtain A_{fuse}, we perform multiple steps. First, we concatenate node embedding of H_{view} and H_{coarser} and then apply a linear transformation layer with activation function to compute raw attention scores, which capture the affinity between H_{view} and H_{coarser} node pair. After that, we generate a binary relationship matrix $A_{\text{relation}} \in \{0, 1\}^{p \times k}$, where p and k represents the number of nodes in G_{view} and G_{coarser}, respectively. Next, we used this binary relationship matrix to perform element-wise multiplication with A_{fuse} to ensure the attention mechanism only considers node pairs that are local and global topological contextually significant. The message passing architecture within GNNs can be expressed as:

$$H^{(t)} = \sigma\left(\left[H^{(t-1)} + AH^{(t-1)}\right] W^{(t)}\right) \in \mathbb{R}^{n \times d} \tag{4.10}$$

where $H^{(t)}$ denotes the hidden node embeddings computed after t steps of the GNN, $A \in \{0,1\}^{N\times N}$ denotes the adjacency matrix, and $W^{(t)} \in \mathbb{R}^{n\times d}$ denotes a learnable weight matrix and the node embeddings $H^{(t-1)}$ generated from the previous message passing step. In our case, we can use A_{fuse} and H_{coarser} as the adjacency matrix and embedding matrix of the fused graph separately in Equation 4.10 and generate two new embeddings H_{fusion1} and H_{fusion2}.

4.4.5 Pooling Aggregation Operation

Using the fusion-view layer, we generate two fused pooled graph embeddings from views 1 to 3. The indices of the selected nodes in view 1 are denoted as L_{idx}, and the embedding matrix following the fusion-view operation step is denoted as H_{fusion1}. Similarly, the indices of the selected nodes and the embedding matrix in view 3 are F_{idx} and H_{fusion2}, respectively. Next, we aggregate these fused graph embeddings to minimize the loss for graph classification tasks. We perform the following formulations to aggregate the pooled graphs:

$$\text{Idx} = L_{\text{idx}} \cup F_{\text{idx}} \tag{4.11}$$

The above Equation allows for extracting the induced graph, G_{pool}, from the selected nodes. The adjacency matrix of this pooled graph is calculated as:

$$A_{pool} = A_{\text{Idx, Idx}} \tag{4.12}$$

where $A_{\text{Idx, Idx}}$ is the row-wise and col-wise indexed adjacency matrix, and $A_{pool} \in \mathbb{R}^{P\times P}$ is the new adjacency matrix of the pooled graph. The node feature matrix for the aggregated pooled graph, X_{pool}, is defined as:

$$X_{\text{pool}}[i,:] = \begin{cases} H_{fusion1}[i,:] & \text{if } i \in L_{\text{idx}} \\ \frac{1}{2}\left(H_{fusion1}[i,:] + H_{fusion2}[i,:]\right) & \text{if } i \in L_{\text{idx}} \cap F_{\text{idx}} \\ H_{fusion2}[i,:] & \text{if } i \in F_{\text{idx}} \end{cases} \tag{4.13}$$

where $X_{\text{pool}} \in \mathbb{R}^{P \times d}$ is the node feature matrix for the aggregated pooled graph.

4.4.6 Hierarchical DSMVPool Architecture and Readout Layer

We integrate multiple GCN layers with DSMVPool layers to make a hierarchical pooling architecture and perform graph classification tasks. Figure 4.1 illustrates the hierarchical framework consisting of three blocks with GCN and DSMVPool layers, and each block takes a graph as input and generates a pooled graph with updated feature and adjacency matrices. Next, the pooled graph is fed into a readout function to combine all node representations and generate a single graph embedding as follows:

$$Z_j = \left[\max_{1 \le i \le N^l} X_{ij}^l \right] \forall j \in [0, d]; \quad R^l = \left[\frac{1}{N^l} \sum_{i=1}^{N^l} x_i^l \quad || \quad Z \right] \tag{4.14}$$

where $Z \in \mathbb{R}^d$ vector contains the maximum values of each feature dimension across all nodes N^l at layer l-th, X_i^l is the feature vector of i-th node, and $||$ denotes concatenation. The readout $R^l \in \mathbb{R}^{2*d}$ concatenates the two (average and max) feature representations. Finally, this graph embedding is fed into a multi-layer perceptron classifier to make predictions.

4.4.7 Complexity Analysis

The computation complexity of our dominant set method is $\mathcal{O}(k * t * N^2)$, where k is the number of clusters, N is the number of nodes, and t is the number of iterations of the replication dynamics with $t << N$ and k depends on the unknown number of clusters in the graph. For space complexity, our method requires $\mathcal{O}(N^2)$ to memorize the similarity matrix. If the graph is very large, we might use a function to compute the edge weight on the fly; hence, the space needed concerns the vector h (Equation 4.4), which requires $\mathcal{O}(N)$.

4.5 Experiments

This section reports the experimental details and evaluates the performance of the DSMVPool. For experiments, we selected four diverse dataset categories: chemical

Table 4.2: *Characteristics and Statistics of eight datasets.*

Classification	Datasets	Total Graphs	Average Nodes	Average Edges	Classes
Biological	Proteins	1,113	39.06	72.82	2
	D&D	1,178	284.32	715.66	2
Chemical	Mutagen	4,337	30.32	30.77	2
	COX2	467	41.22	43.45	2
	BZR	405	35.75	38.36	2
Social Networks	REDDIT-M-12K	11,929	391.41	456.89	11
	IMDB-Binary	1000	19.77	96.53	2
Computer Vision	MSRC_21	563	77.52	198.32	20

molecules, biological networks, social networks, and computer vision. These datasets are benchmarks against which we compare DSMVPool with state-of-the-art competitors and baselines methods. Table 4.2 shows the details of the datasets. Additionally, we conduct ablation studies to evaluate the contribution of each view in the DSMVPool. We also visualize the results of various pooling layers to see how baselines and the proposed pooling method reduce the nodes.

4.5.1 Competitors and Experimental Settings

(1) Graph Neural Networks: We select four GNN architectures to conduct comparative experiments: GCN [72], GAT [125], GraphSAGE [56] and GraphConv [91] .

(2) Graph Pooling Methods: We compare our DSMVPool with nine advanced hierarchical pooling methods to analyze the results of the experiment: TopkPool [50], SAGPool [77], MAC [140], DiffPool [143], CliquePool [84], Quasi-CliquePool [4], MuchPool [43], MPool [65], and Wit-TopoPool [30].

(3) Experimental Settings: DSMVPool is implemented using the PyTorch framework [96] and PyTorch Geometric library [48]. We used the provided node features to generate the edge weights for our dominant set cluster pooling method (see Section 4.4.2). To ensure a fair comparison, we follow numerous prior research studies [77, 4, 43], employing tenfold cross-validation to assess the performance of the models listed above and provide average accuracy and standard deviation. For all the competitors we used the same GNN backbone [91] as a message passing function to fairly emphasize the contributions of each

Table 4.3: *Comparison of DSMVPool and baselines. The highest score is in* ***bold****, and the second highest score is in underline (OOR referred to as out-of-resources).*

Class	Methods	COX2	BZR	MUTAGEN	RED-M	IMDB-B	D&D	PROTEINS	MSRC-21
GNNs	GCN [72]	78.16$_{\pm 0.85}$	80.01$_{\pm 2.39}$	78.18$_{\pm 2.58}$	23.84$_{\pm 1.92}$	60.10$_{\pm 5.34}$	75.13$_{\pm 4.14}$	73.77$_{\pm 5.59}$	84.04$_{\pm 6.40}$
	GAT [125]	78.37$_{\pm 0.66}$	80.74$_{\pm 2.52}$	78.99$_{\pm 2.44}$	21.73$_{\pm 1.03}$	52.00$_{\pm 2.12}$	76.91$_{\pm 2.68}$	75.30$_{\pm 5.23}$	88.80$_{\pm 4.38}$
	GraphSAGE [56]	79.51$_{\pm 2.96}$	80.99$_{\pm 2.83}$	76.70$_{\pm 2.20}$	28.73$_{\pm 1.34}$	70.50$_{\pm 2.57}$	77.48$_{\pm 3.20}$	76.73$_{\pm 3.00}$	85.63$_{\pm 4.24}$
	GraphConv [91]	80.01$_{\pm 4.43}$	82.12$_{\pm 2.89}$	79.18$_{\pm 3.13}$	37.53$_{\pm 2.34}$	70.12$_{\pm 2.78}$	76.57$_{\pm 3.98}$	74.04$_{\pm 5.07}$	88.66$_{\pm 3.47}$
Pooling	TopkPool [50]	77.93$_{\pm 2.79}$	81.49$_{\pm 4.17}$	78.56$_{\pm 1.94}$	37.56$_{\pm 3.39}$	70.11$_{\pm 2.31}$	73.85$_{\pm 4.63}$	73.32$_{\pm 6.09}$	88.45$_{\pm 3.70}$
	SAGPool [77]	78.37$_{\pm 1.86}$	81.24$_{\pm 4.01}$	77.25$_{\pm 3.03}$	36.79$_{\pm 3.53}$	70.65$_{\pm 3.36}$	72.49$_{\pm 2.87}$	73.50$_{\pm 4.56}$	87.92$_{\pm 3.24}$
	MAC [140]	78.36$_{\pm 0.16}$	82.30$_{\pm 2.58}$	80.33$_{\pm 2.01}$	42.67$_{\pm 2.23}$	57.20$_{\pm 0.92}$	79.13$_{\pm 4.70}$	76.08$_{\pm 3.55}$	89.75$_{\pm 2.12}$
	DiffPool [143]	77.60$_{\pm 2.70}$	78.90$_{\pm 0.40}$	71.80$_{\pm 0.15}$	OOR	68.40$_{\pm 0.51}$	74.31$_{\pm 2.15}$	77.62$_{\pm 4.97}$	83.30$_{\pm 0.51}$
	CliquePool [84]	78.37$_{\pm 1.86}$	82.17$_{\pm 2.25}$	78.47$_{\pm 1.62}$	36.11$_{\pm 2.13}$	70.00$_{\pm 3.71}$	74.81$_{\pm 3.87}$	73.56$_{\pm 2.86}$	88.72$_{\pm 1.14}$
	QuasiPool [4]	80.15$_{\pm 2.13}$	82.21$_{\pm 2.58}$	79.53$_{\pm 1.58}$	38.21$_{\pm 2.43}$	70.30$_{\pm 1.45}$	75.30$_{\pm 3.30}$	75.68$_{\pm 1.38}$	88.15$_{\pm 2.28}$
	MuchPool [43]	79.27$_{\pm 6.09}$	80.28$_{\pm 6.93}$	78.75$_{\pm 2.48}$	OOR	65.10$_{\pm 6.65}$	76.48$_{\pm 7.01}$	78.52$_{\pm 3.89}$	86.85$_{\pm 3.61}$
	MPool [65]	78.10$_{\pm 0.10}$	78.70$_{\pm 0.11}$	79.60$_{\pm 3.70}$	OOR	71.02$_{\pm 3.57}$	**80.20**$_{\pm 2.10}$	**79.30**$_{\pm 3.30}$	87.52$_{\pm 0.54}$
	Wit-TopoPool [30]	78.10$_{\pm 0.10}$	79.75$_{\pm 0.14}$	73.88$_{\pm 1.29}$	23.19$_{\pm 1.12}$	70.20$_{\pm 6.76}$	71.30$_{\pm 0.37}$	75.88$_{\pm 5.60}$	89.16$_{\pm 4.00}$
Our	**DSMVPool**	**81.13**$_{\pm 2.96}$	**83.58**$_{\pm 2.75}$	**81.39**$_{\pm 1.98}$	**44.61**$_{\pm 2.15}$	**72.10**$_{\pm 2.98}$	**80.91**$_{\pm 2.14}$	**79.84**$_{\pm 2.53}$	**90.89**$_{\pm 2.5}$
	Gain	**+0.98**	**+1.28**	**+1.68**	**+1.94**	**+1.08**	**0.71**	**0.54**	**+1.14**

pooling method. We utilize the official source codes of all methods provided by the authors and tune hyperparameters to reproduce the results according to the requirements in their papers. Hyperparameters are fine-tuned within specified ranges, such as embedding dimensions in the range of {64, 128, 256}, learning rate in the range of {0.01, 0.001, 0.0001}, batch size in the range of {32, 64, 128, 256} and pooling ratio in the range {0.5, 0.6, 0.7, 0.8, 0.9}. We utilize the Adam optimizer[70] to initialize our model and apply a negative log-likelihood loss function for training. We implement patience and an early stopping criterion, which stops the training process if the loss value of the validation set does not decrease for 50 consecutive epochs.

4.5.2 Performance Comparison with state-of-the-art

We evaluated our proposed DSMVPool and other baseline methods on the eight datasets for the graph classification task and reported the accuracy and standard deviation in Table 4.3. The proposed DSMVPool achieves superior results among all baseline methods in the domain of social networks, chemical molecules, and computer vision while being on par with biological networks. For example, our pooling method outperforms the top baselines by 1.28% for BZR, 0.98% for COX2, and 1.68% for Mutagenicity in the chemical

molecular domain. Interestingly, the average number of edges in the chemical compounds datasets is significantly less than in the other datasets, as seen in Table 4.2. Because of their sparsity, these five datasets present a considerable challenge for pooling layers to acquire meaningful representations. Our fusion-view layer captures different contextual information from the graph, such as local and global information, allowing DSMVPool to extract hidden information effectively from the sparse graph topology.

In addition, DSMVPool consistently surpasses GCN-based global pooling methods across all datasets. These superior results highlight the effectiveness of our method in generating more significant graph representations, emphasizing the need to integrate hierarchical pooling layers into the learning process. Specifically, DSMVPool and other node cluster pooling methods like DiffPool, CliquePool, Quasi-CliquePool, and MPool demonstrate superior performance compared to GNN models. Moreover, the CliquePool and DiffPool do not consistently perform better than node selection pooling methods like SAGPool and TopkPool. This observation further substantiates the idea that integrating the graph's local and global properties in a pooling method can lead to more effective methods for graph classification tasks. Table 4.3 demonstrates that the proposed approach achieves notable improvements in multiclass datasets, with a 1.14% enhancement on MSRC 21 and a 1.94% enhancement on REDDIT-MULTI. The DSMVPool also considerably improves the IMDB-Binary dataset, resulting in a 1.08% increase. It is worth noting that the average number of nodes and graphs is small in IMDB-MULTI, with only 19 and 1000, respectively. This phenomenon is consistent with the results obtained in chemical molecules, which strongly confirms the correctness of the previous analysis. In conclusion, our DSMVPool consistently outperforms baseline pooling approaches on eight benchmarks.

4.5.3 Ablation Study

This section performs an ablation study on DSMVPool by removing three views pooling to verify further where the performance improvement comes from. For convenience, we name the DSMVPool method without the *local*, *global*, and *feature* views as $\text{DSMVPool}_{\text{NLV}}$,

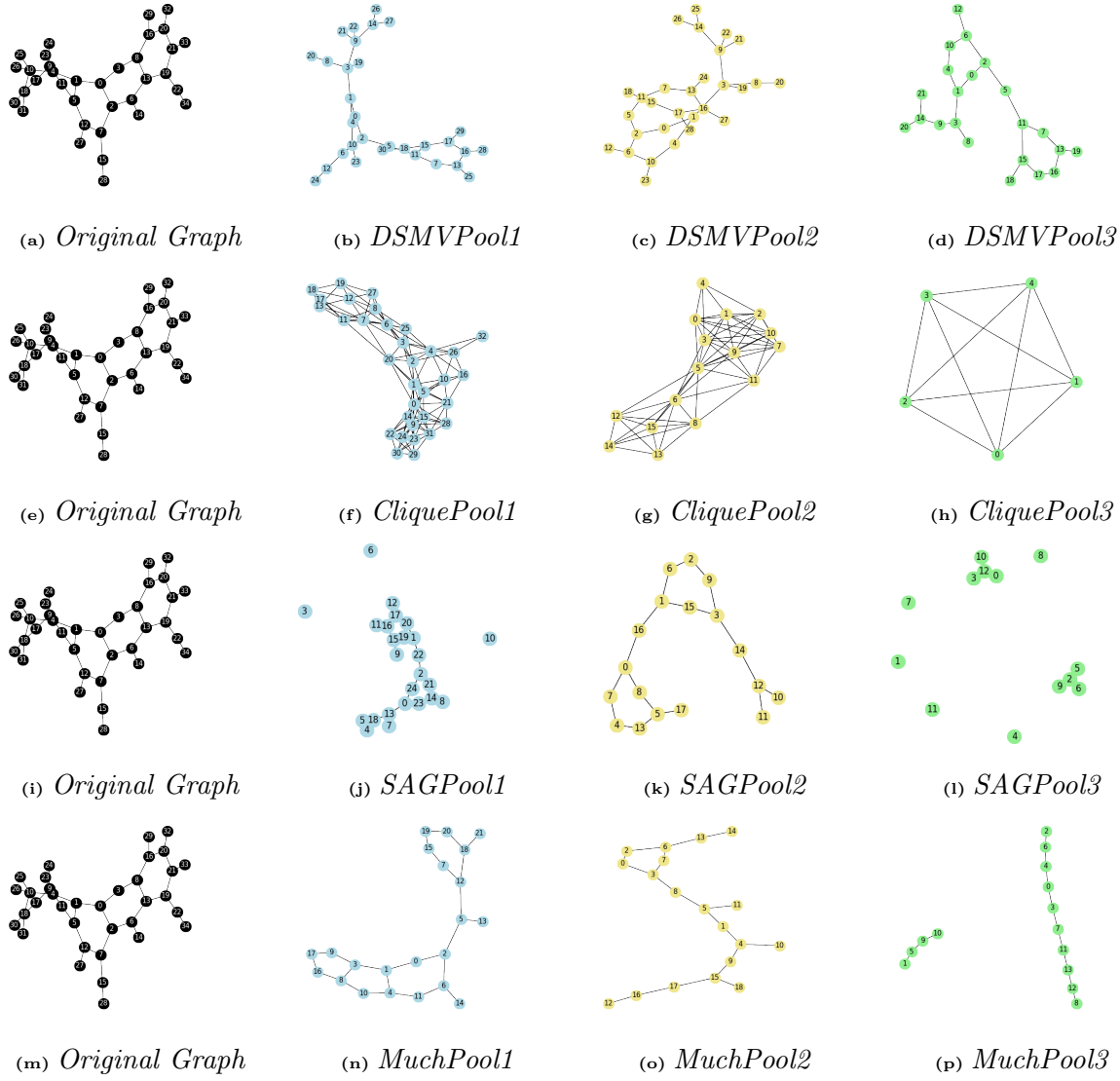
Table 4.4: *Effect of different graph pooling views in DSMVPool.*

Architecture	MUTAGEN	PROTEINS	BZR	IMDB-B	MSRC21
$\text{DSMVPool}_{\text{NLV}}$	79.82 ± 2.63	78.65 ± 2.51	79.02 ± 1.16	70.80 ± 4.41	88.47 ± 4.24
$\text{DSMVPool}_{\text{NGV}}$	78.47 ± 2.97	76.56 ± 2.79	80.00 ± 5.48	71.00 ± 4.08	87.93 ± 3.51
$\text{DSMVPool}_{\text{NFV}}$	80.61 ± 1.70	78.21 ± 2.70	78.77 ± 1.59	71.70 ± 3.03	87.75 ± 5.03
DSMVPool	81.39 ± 2.01	79.84 ± 2.53	83.58 ±3.15	72.10 ± 3.40	90.89 ± 2.5

$\text{DSMVPool}_{\text{NGV}}$, and $\text{DSMVPool}_{\text{NFV}}$, respectively. We chose five different-scale graph datasets covering small and large graphs for experiments. The results presented in Table 4.4 highlight the considerable impact of our dominant set-based clustering pooling, particularly within the domains of chemical molecules and MSRC graphs, since capturing the global structure with local or node features is especially useful for the classifier to distinguish the graphs. Furthermore, it can be observed that $\text{DSMVPool}_{\text{NLV}}$ outperforms $\text{DSMVPool}_{\text{NFV}}$ in datasets characterized by sparse graph structures, such as those involving chemical molecules, because the local topology pooling operation may generate isolated nodes, which could adversely affect message passing in subsequent network layers. Overall, DSMVPool's ability to learn both the graph's local and global topological information with node features and edge weight information enables it to generate robust graph representations that significantly enhance performance in classification tasks.

4.5.4 Graph Visualization

To further demonstrate how DSMVPool outperforms baselines, we use networkx3 to visualize the pooling outcomes of the proposed approach and three other pooling methods: MuchPool, SAGPool, and CliquePool. For a fair comparison, we make a three-layer hierarchical pooling architecture and set a 0.7 pooling ratio. We select a random graph from the Mutagenicity dataset containing 35 nodes. The first pooling layer of all methods shows that the DSMVPool, CliquePool, and MuchPool largely preserved the significant topological structure of the original graph, including ring and branch structures. However, the SAGPool contains several isolated nodes. The findings from the second and third pooling layers demonstrate that SAGPool, CliquePool, and MuchPool struggle to generate pooled graphs with appropriate topological structures.



(a) *Original Graph* (b) *DSMVPool1* (c) *DSMVPool2* (d) *DSMVPool3*

(e) *Original Graph* (f) *CliquePool1* (g) *CliquePool2* (h) *CliquePool3*

(i) *Original Graph* (j) *SAGPool1* (k) *SAGPool2* (l) *SAGPool3*

(m) *Original Graph* (n) *MuchPool1* (o) *MuchPool2* (p) *MuchPool3*

Fig. 4.2: *Graph visualization of different pooling approaches. The graphs show that DSMVPool preserves the graph's original topological structures during the pooling operation.*

Meanwhile, the second and third layers of our DSMVPool generate the pooled graph with reasonable topological structures, including dual ring structures in the original graph. We can see the effectiveness of DSMVPool since ring structures are crucial in characterizing molecules. The visualization findings align with the results in Table 4.3.

4.6 Conclusion and Future Work

This chapter developed a novel Dominant Set Multi-View Pooling approach for hierarchical graph representation learning. We developed a dominant set cluster pooling method

to identify all potential clusters using edge weights, generating a coarser view of the graph. Furthermore, we generated two pooled graph views by selecting the most significant nodes based on the graph's local topological structures and node features importance. Next, we developed an attention-fusion-view layer to fuse the coarser graph with the pooled graphs and perform aggregation to form the final pooled graph. We evaluate the performance of DSMVPool on ten graph-classification datasets, including four distinct domains. Our method outperforms baselines and competitors on all datasets. In the future, we plan to use it for analyzing protein structures, which can help to design drugs that bind to pre-selected regions, targeting specific biological pathways and reducing the probability of off-target effects.

Chapter 5

Glocal Attention: Hierarchical Pooling for Graph Learning

"Topology is precisely the mathematical discipline that allows the passage from local to global."

— Rene Thom

5.1 Preamble

In this chapter, we designed a novel Hierarchical Global Local Attention Pooling Dominant that leverages graph structural information and node features to identify the most relevant global structures (cliques). Furthermore, we developed a multi-attention LocalPool to capture the local node's properties from the ranked global structures to reduce the size of the graph for downstream tasks such as graph classification. The research and findings discussed in this chapter are based on the following paper:

- **Waqar Ali**, Sebastiano Vascon, Thilo Stadelmann, and Marcello Pelillo. "Hierarchical Glocal Attention Pooling for Graph Classification." Pattern Recognition Letters Journal 2024 [5].

The author has the following contributions:

- **Developing** the overall framework of the algorithm.
- **Writing** the complete code.
- **Performing** the overall experiments.
- **Writing** the complete paper.

5.2 Introduction

GNNs have recently gained significant attention due to their ability to process graph-structured data effectively. They have shown effectiveness in various tasks of classifying graphs and learning graph representations [72], including understanding and predicting interactions between molecules and proteins [135, 103] that can lead to significant advancements in drug discovery, analyzing structure and dynamics of interactions in social networks [68], operating on knowledge graphs to enable Retrieval-Augmented Generation (RAG) [88] or enable the detection of visual objects from their context [58], e.g., to support document accessibility [112]. Within the scope of graph classification, the Graph Pooling (GP) methods play a pivotal role in GNN architectures that map the set of nodes or subgraphs into a compact representation to capture a meaningful structure of the entire graph [94]. Early GP methods, referred to as global GPs [12, 116], are the simplest approaches, which reduce the size of the input graph by performing a sum or average of all nodes without considering the hierarchical information of the graph and may lose feature information. Recently, several advanced hierarchical GP methods, such as node cluster and node selection methods [30, 50, 84], have been proposed to tackle the limitations of global pooling and obtain state-of-the-art performance.

The node cluster methods like DiffPool [143] and CliquePool [84] capture the Global Topological Structure (GTS) of a graph, which extracts the overall architecture and connectivity patterns of the entire network by partitioning the graph into clusters. Then, each cluster pooled into a supernode to create a coarsened graph. However, the CliquePool method has limitations in extracting overlapping nodes between cliques and treats all

cliques equally informative without considering node attributes. On the other hand, node selection methods such as Top-k pooling, SAGPool, and Topological Pooling [140, 77, 30] identify the most significant nodes based on their feature values or attention scores while dropping unnecessary nodes. These methods primarily focus on the graph's Local Topological Structure (LTS), which includes the analysis of individual nodes, their attributes, and neighborhoods. However, there is a lack of an effective GP method that combines both global and local properties of the graph hierarchically to improve the graph representation.

To address the limitations of existing GP methods, this chapter proposes Hierarchical Global-Local Attention Pooling (HGLA-Pool), a novel pooling method designed to capture graphs' local and global properties. The proposed approach is structured as a two-fold process, each fold addressing specific limitations of existing GP approaches. The fold-1 leverages the idea of cliques that perform clique pooling [84], which aims to encapsulate the graph's GTS. In this fold, we design a rule-based method to enhance the clique representation by identifying the overlapping nodes between cliques. Identifying overlapping nodes is crucial in various real-world scenarios, such as molecular biology, where a molecule might serve as a shared binding site for multiple proteins, and in social network analysis, where individuals often belong to multiple social circles, including professional networks and personal connections [135, 68]. Additionally, we introduce a novel dynamic fusion method that incorporates graph structure information and node features to calculate scores for each clique and select the Top-k significant cliques using this fusion score. Furthermore, we recognize that every node within a selected clique does not contribute equally to the graph representation during the pooling operation. We argue that the most informative nodes from all ranked cliques should be captured in GP. Therefore, in fold-2 we develop a LocalPool layer on top of fold-1 using multi-attention to pinpoint and emphasize the most informative nodes within the ranked cliques, ensuring a more meaningful representation of the overall graph for pooling operation. The hierarchical architecture of our method, as illustrated in Figure 5.1, reflects this dual-fold approach. We summarized our contributions as follows:

- This chapter introduces HGLA-Pool, a novel pooling layer that sequentially integrates the global structural properties of the graph with the local node's properties.
- We develop a rule-based method to extract the overlapping nodes between cliques. Additionally, we design a novel dynamic fusion method that leverages both graph structural information and node features to identify the most relevant global structures (cliques). Furthermore, this chapter develops a multi-attention LocalPool to capture the local node's properties.
- We experimentally prove that considering both sources of information (global and local) yields a better-learned representation. We consistently outperformed 11 state-of-the-art models on seven diverse and challenging benchmarks.

5.3 Related Work

Related work includes existing GP studies focusing on global and hierarchical techniques. Global pooling methods usually use sum or mean functions to aggregate the node features that generate a single vector representation for the entire graph. For example, Vinyal et al. [127] proposed a Set2Set framework that uses the Long Short-Term Memory model to generate graph representation by identifying informative nodes. Authors [147] developed a novel GNN layer to capture multi-level node features and a sort pooling that sorts these features to keep more graph information. However, global methods ignore the graph's hierarchical information during the pooling operations. Meanwhile, hierarchical methods leverage graph structures and node features for the pooling. Based on designing properties, the hierarchical methods can be classified into two main classes: node selection and node cluster.

Node selection methods reduce the graph size by selecting the most informative nodes based on their node features or attention scores. Hongyang et al. [50] developed a Top-k pooling that uses node scalar values on a trainable projection vector as the node score. SAGPool [77] further adopts the GNN layer to calculate node scores. Jinheon et al. [140] proposed MAC pooling that incorporates dual-node scoring strategies to obtain the

importance of nodes. These methods are considered more computationally efficient as they only require calculating an informative node's score, but they ignore the graph's GTS information.

The node cluster method captures the global properties by finding the clusters within the graph. DiffPool [143] uses the GNN layer output to learn a differentiable soft cluster assignment to extract the clusters. In [84], the authors introduced a CliquePool method to learn the GTS by extracting the maximal cliques from the graph. Quasi-CliquePool [4] further improves the CliquePool by capturing the overlapping nodes between two cliques using the replicator dynamic algorithm. [65] developed a Motif-based pooling method that extracts the high-order graph structure by combining cluster and selection pooling operations. The motif method treated all motif structures equally without considering the node features. Graph Multiset Transformer (GMT) pooling incorporates node structural dependencies with a multi-head attention mechanism to enhance the graph representation by identifying node interaction. Jinlong et al. [43] design a Multi-channel pooling (MuchPool) that combines TopkPool and DiffPool methods to create a more comprehensive hierarchical representation of graphs. However, the MuchPool method suffers from considerable computational complexity due to the simultaneous operation of three distinct channels to capture node feature information, global and local structural information.

A recent study [30] introduced a Wit-TopoPool, which integrates a witness complex-based topological embedding mechanism with a global pooling layer. This approach aims to extract comprehensive and discriminative topological information from graphs.

In conclusion, the current landscape in pooling methods demonstrates a notable gap: the lack of an effective pooling approach that combines node selection and node cluster techniques sequentially to yield a robust graph representation. Addressing this gap, we proposed a novel HGLA-Pool method that captures the local and global properties of the graph with node features to generate a more robust graph representation.

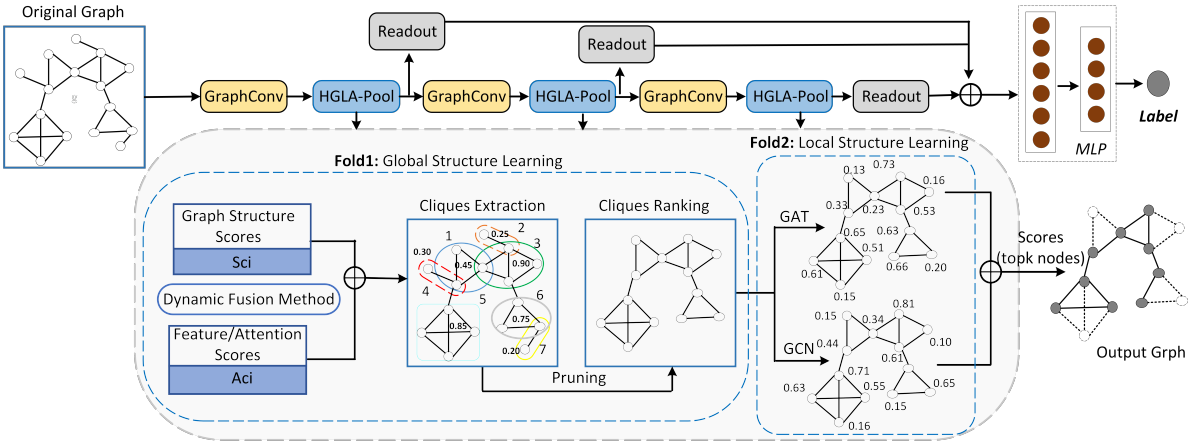


Fig. 5.1: *The hierarchical architecture of the HGLA-Pool integrates with GNN for graph classification. The grey lightbox demonstrates the overall workflow of HGLA-Pool, which contains two folds. The* ***fold-1*** *performs three steps: 1) capture GTS by finding all maximal cliques within the graphs; 2) find overlapping nodes between two cliques to obtain more reasonable clique representations; and 3) incorporate the dynamic fusion method to rank the cliques and form a new pooled graph. Meanwhile, the* ***fold-2*** *takes this pooled graph as input and captures the LTS by selecting several significant nodes from ranked cliques based on scoring criteria (e.g., GCN and GAT). The readout function is applied after each pair of GNN and HGLA-Pool layers to generate the graph-level representation.*

5.4 Methodology

This section explains our proposed HGLA-Pool, shown in Figure 5.1. The HGLA-Pool performs pooling operations in two-fold, capturing the graph's global and local properties, and then aggregating the results to form a new pooled graph. The following subsections present a detailed description of the proposed HGLA-Pool.

5.4.1 Global Topological Structure Learning

In fold 1, we aim to capture the GTS information of the graph. For this purpose, we use the CliquePool [84] with a modified version of the Bron-Kerbosch algorithm [27] to extract all cliques from the input graph G, denoted as a set $C = \{c_1, c_2, ..., c_i\}$. This fold is designed to overcome two significant challenges of CliquePool [84]: extracting overlapping nodes between cliques and ranking cliques based on their node features.

Overlapping Nodes Extraction: Overlapping nodes substantially impact various real-world scenarios, including molecular biology, where a particular molecule may act as a shared binding site for multiple proteins, and in social network analysis, a person may

be part of both a professional network. We design a rule-based algorithm using three conditions to handle nodes that belong to multiple cliques during the clique extraction process: 1) If a node is already assigned to a clique, it is only added to the current clique being considered if the sizes of the existing and current clique are equal or current clique size is bigger than existing clique in terms of nodes, 2) If a node is connected to multiple cliques, in that case, we evaluate whether this specific node has maximum connectivity with the nodes of an associated clique. If it does and is not already part of the associated clique, the node is integrated into that clique, and 3) If nodes of a clique have already been assigned to larger cliques, then we remove that clique. This criterion ensures that the node's association with the clique is not arbitrary but is based on a meaningful and substantial connectivity pattern. In Figure 5.1, the fold-1 demonstrates the extraction of overlapping nodes; see the cliques 1, 2, 3, and 4 with overlapping nodes.

The existing clique pooling method [84] partitions the graph into cliques based on the graph's topological structure without considering the node features. Furthermore, this method treats all cliques equally, which can be problematic because not all cliques are equally significant; some are rich in information, and others might merely consist of a single node, thus carrying significantly less information to subsequent pooling layers. To solve these issues, we developed a novel dynamic fusion method to calculate each clique's importance using graph structure information and node features.

Graph Structure Score: The graph structure can provide useful information on the importance of nodes, such as node degrees and the shortest paths between different nodes, etc. Hence, we borrow the node centrality definition to illustrate each clique's importance within the graph. We introduce a heuristic method to calculate a score for each clique using graph structure information: fusion of degree centrality and unique neighbor count. In the first step, we compute the mean degree centrality $\bar{D}_{c_i}$ of all nodes in c_i. Next, we calculate the number of unique neighbors $\bar{U}_{C_i}$ of a clique c_i by collecting all neighbors of nodes in c_i and removing any nodes that are part of this clique itself. Finally, the structure score S_{c_i} of each clique is derived as a weighted sum of these two $\bar{D}_{c_i}$ and $\bar{U}_{C_i}$

metrics, parameterized by β and calculated as follows:

$$\bar{D}_{c_i} = \frac{1}{|c_i|}\sum_{v \in c_i} d_v; \quad \bar{U}_{c_i} = |\cup_{v \in c_i} N(v) - c_i| \quad S_{c_i} = \beta \times \bar{D}_{c_i} + (1-\beta) \times \bar{U}_{c_i}; \tag{5.1}$$

Node Feature Score: In addition to topological information, the node features information contributes significantly to defining the graph properties. For instance, in chemical molecules, node features represent atom types, which are essential for predicting the graph's characteristics. Therefore, it can be an effective resource to show the clique's importance within a graph. For this purpose, we use a graph convolution layer to calculate attention scores for each node from the node feature. The calculation procedure is outlined as follows:

$$a(v) = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}XW); \quad A_{c_i} = \sum_{v \in c_i} a(v) \tag{5.2}$$

where *a(v)* represents the attention scores of each node and $\sigma(.)$ is a nonlinear activation function (softmax). The $\tilde{A} = A + I$ shows the adjacency matrix with self-loops, $\tilde{D} = \sum_j \tilde{A}_{ij}$ is the degree matrix of $\tilde{A}$ and W is a trainable weight matrix. To calculate the score for each clique, various aggregation functions like max and sum can be employed. In our implementation, we choose the sum function to aggregate the attention scores of the nodes within the corresponding clique c_i. Given a clique c_i, the attention clique score A_{c_i} is computed in Equation 5.2.

Fusion Score: To achieve a more robust score for each clique, it is necessary to use both graph structure and node feature information fully. So, we combine S_{c_i} with the A_{c_i} score using a learnable parameter α to obtain a fuse score F_{c_i} (see leftmost side of Equation 5.3), thus providing a more comprehensive representation of the graph's topology. After obtaining the F_{c_i} scores, we used them to select cliques that the pooling operator should preserve. In detail, we first rank the cliques according to their fusion score F_{c_i}, then a subset of top-ranked cliques are selected for constructing the pooled graph as follows:

$$F_{C_i} = \alpha \times A_{Ci} + (1-\alpha) \times S_{C_i}; \quad C'_j = rank(F_{Ci}, \lceil C_r * |C| \rceil) \tag{5.3}$$

where rank(.) denotes the function that returns the important cliques, C_r is the clique ratio, and $|C|$ is the total cliques. We update the node feature based on selected clique

nodes (see leftmost side of Equation 5.4). We construct a new adjacency matrix A' where only the edges between the nodes in the selected cliques are retained. For this purpose, we define an edge mask matrix E_m, which is structured to correspond with the A using node mask M_i. Each element in E_m is set to 1 if both corresponding nodes (v_i and v_j in M_i) are part of the selected cliques and set to 0 otherwise. This can be represented mathematically as:

$$X' = X \odot M; \quad A' = A \odot E_m \tag{5.4}$$

where X' is the new node feature matrix, X is the original node feature matrix, $\odot$ represents the broadcasted element-wise product, and M is a vector where each entry corresponds to a node in the graph. The dimension of M is the same as the number of nodes in the graph. Each entry in M specifies whether or not the corresponding node is part of a selected clique (1 for nodes in C'_k and 0 for all other nodes). A' is the new adjacency matrix, and A is the original edge index matrix. The first fold's output serves as an input for the subsequent fold. The information provided by this fold acts as a contextual framework that enables the second fold to identify local structures effectively.

5.4.2 Local Topological Structure Learning

This section presents how our proposed LocalPool layer leverages multi-attention mechanisms to learn LTS information by extracting the most informative nodes from ranked cliques. Existing pooling methods like CliquePool and KPlexPool [84, 9] extract all possible cliques from the input graph and often transfer each clique into a single node to form a pooled graph. This transformation aims to reduce the graph size and complexity by aggregating the information within each clique. However, this transformation does result in a loss of information at the individual node level within each clique. The fine-grained details of interactions and relationships between nodes within a clique are not preserved in the pooled representation. This loss of information can potentially affect the downstream tasks, especially if the specific node-level information is crucial for classification tasks.

To address this, we propose a LocalPool layer to refine the graph further by selecting

the most important nodes from ranked cliques. This refinement and selection process allows our pooling operation to help preserve the key information within each clique while reducing the graph size and complexity. Our LocalPool approach significantly diverges from existing methods like SAGPool [77]. Specifically, SAGPool relies on a single strategy for computing node importance, yielding less robust node rankings. Inspired by the MAC method Xu et al. [140], we adopt a dual-strategy-based pooling layer to calculate node importance, as illustrated in Figure 5.1. First, we employ a GCN layer to calculate importance scores for the nodes. This choice stems from the GCN model's ability to learn node representations by aggregating information from neighboring nodes, offering a localized perspective. GCNs aggregate feature information from a node's local neighborhood, providing a stable and robust representation by averaging the features of neighboring nodes. Simultaneously, we employ the Graph GAT layer to compute attention scores. GATs introduce an attention mechanism that assigns different importance to different nodes within a neighborhood, capturing more intricate relationships. GATs can adapt to varying graph structures by learning the attention coefficients, making them versatile and capable of capturing complex patterns and dependencies that might be missed by the uniform aggregation of GCNs. Given that GCN and GAT extract graph information from distinct viewpoints—GCN provides a robust, averaged view of a node's neighborhood, and GAT offers a nuanced, dynamically weighted view. we consider this process a multi-attention strategy for calculating node importance scores. By aggregating the scores from both GCN and GAT (using operations such as sum or max), we achieve a more comprehensive and robust node importance score, which enables our pooling method to select the most important nodes. We consider this process as a multi-attention strategy, and it is calculated as follows:

$$S_{N_i} = Aggregation\left(\sigma\Big(\frac{1}{h}\sum_{h=1}^{h}\sum_{j\in N(i)} {a_{ij}}^{h} W^h X'_j\Big), \sigma\Big(\tilde{D}^{-1/2}\tilde{A}'\tilde{D}^{-1/2}X'W'\Big)\right) \tag{5.5}$$

where S_{N_i} represents an attention score for a node, and Aggregation is an operation such as max, mean, and sum. We use the max function to select the maximum value from each row (e.g., for each node, choose the maximum score from either the GCN or GAT model). The h shows the number of heads, a_{ij} is the attention weights between x'_i and

x'_j, W^h is a trainable weight matrix of head h, X' is a feature vector, and W' represents the trainable weight matrix of the pooled graph of the fold-1. Utilizing the scores from S_{N_i}, we identify and select high-score nodes to construct a pooled graph. The details of this process are as follows:

$$idx = topk(S_{N_i}, \lceil P_r * N' \rceil); \quad X^{l+1} = X'(idx, :) \odot S_{N_i}(idx, :); \quad A^{l+1} = A'(idx, idx) \tag{5.6}$$

where *topk* is the function that returns the indices of the top $N^{l+1} = \lceil P_r * N' \rceil$ values. The $X'(idx, :)$ perform the row-wise (node-wise) indexed node feature matrix, $S(idx, :)$ represents the row-wise indexed node importance score matrix, and $A'(idx, idx)$ is the row-wise and col-wise indexed adjacency matrix. X^{l+1} and A^{l+1} represent the new node feature and adjacency matrices, respectively.

5.4.3 Hierarchical HGLA-Pool Architecture and Readout Layer

We employ a hierarchical pooling architecture, integrating multiple GCN layers with HGLA-Pool layers to perform graph classification tasks. Figure 5.1 shows the architecture consisting of three blocks with GCN and HGLA-Pool layers. Each block receives a graph as input and generates a pooled graph with a new feature and adjacency matrices. Concurrently, we use mean-pooling and max-pooling as a readout layer after each block to aggregate all the node representations to obtain a single graph embedding as follows:

$$\Gamma_j = \left[\max_{1 \le i \le N^l} X^l_{ij} \right] \forall j \in [0, d]; \quad R^l = \left[\frac{1}{N^l} \sum_{i=1}^{N^l} x^l_i \quad || \quad \Gamma \right] \tag{5.7}$$

where N^l is the number of nodes at layer l-th, x_i is the feature vector of i-th node, and $||$ denotes concatenation. The $\Gamma \in \mathbb{R}^d$ vector contains the maximum values of each feature dimension across all nodes. The readout $R^l \in \mathbb{R}^{2*d}$ concatenates the two (average and max) feature representations. Finally, this graph embedding is subsequently fed into a multi-layer perceptron classifier to make predictions.

5.5 Experiments and Analysis

This section presents a comprehensive evaluation and quantitative analysis of our proposed HGLA-Pool's effectiveness. We perform exhaustive experiments across four diverse dataset categories: chemical molecular structures [105, 129] (including Mutagenicity, NCI-1, NCI-109, COX2, and BZR), social networks [9] (Reddit-Multi-12K), biological networks [21, 42] (DD and Proteins) and computer vision [64] (MSRC_21). These datasets are benchmarks against which we compare HGLA-Pool with state-of-the-art baseline methods. Additionally, we conduct ablation studies to evaluate the individual contributions of each fold within the HGLA-Pool layer. To further analyse the robustness of our approach, we also explore the impact of hyperparameter variations on the performance of our pooling method.

5.5.1 Baselines and Experimental Settings

(1) Graph neural networks: We select three GNN architectures to conduct comparative experiments: GCN [72], GAT [125] and GraphConv [91].

(2) Graph pooling methods: In this group, we compare our HGLA-Pool with eleven state-of-the-art hierarchical and global pooling approaches such as DiffPool [143], SortPool [147], TopkPool [50], SAGPool [77], CliquePool [84], GMT [11], MuchPool [43], MPool [65], MAC [140], Quasi-CliquePool [4], and Wit-TopoPool [30].

(3) Experimental settings: We implement HGLA-Pool using the PyTorch framework [96] and the PyTorch Geometric library [48]. For a fair comparison, we follow many previous works [77, 4, 43], employing tenfold cross-validation to evaluate the effectiveness of all models mentioned above and reporting average accuracy and standard deviation. We utilize the source codes of all baselines provided by the authors and tune hyperparameters to reproduce the results according to the requirements in their papers. We use the same GCN [91] layer as a message-passing function.[1] Hyperparameters are optimized within predefined ranges, including embedding dimensions {32, 64, 128, 256}, learning

[1]We reproduced the numbers of all baselines using the source codes given by the authors, and these numbers are reported in Table 5.1. For transparency and reproducibility, the source codes for all baselines referenced in our analysis are available on this link.

rate {0.01, 0.001, 0.0001, 0.05, 0.005, 0.0005}, batch size {64, 128}, pooling and clique ratios {0.5, 0.6, 0.7, 0.8, 0.9}. We employed the Adam optimizer for model initialization and a negative log-likelihood loss function for training. We also adopt patience and early stopping criterion, i.e., if the loss value of the validation set does not reduce for 50 consecutive epochs, then the training process will be stopped.

5.5.2 Performance Comparison

We evaluated the performance of our proposed method (HGLA-Pool with importance scores Z_C and HGLA-Pool with dynamic fusion scores F_C) and baseline methods on the nine benchmark datasets for the graph classification tasks. We outlined the classification accuracy with standard deviation in Table 5.1. In this comparison, HGLA-Pool method achieves superior performance among all other baselines in the chemical molecules, social networks, and computer vision domain datasets. For instance, in the case of the chemical molecular domain, our method outperforms the best baselines by 2.81% improvement for the NCI1, 2.42% for the NCI109, 3.92% for the BZR, 1.57% for the COX2 and 2.07% for the Mutagenicity. It is worth noting that the average number of edges in these datasets is much smaller compared to the other datasets. This implies that these five chemical datasets are relatively sparse, presenting a significant challenge for pooling layers to learn meaningful representations. However, our designed dual-fold attention strategy can capture both the graph's local and global information, which enables HGLA-Pool to extract information undeterred by the sparse graph structure effectively.

Furthermore, HGLA-Pool consistently outperforms GCN-based global pooling methods across datasets from all four domains. This superior performance underscores the efficacy of our approach in generating more meaningful graph representations, thereby emphasizing the value of incorporating hierarchical pooling layers into the learning process. Notably, HGLA-Pool and other node clustering techniques like CliquePool, Quasi-CliquePool, and MPool exhibit better performance than other GNN-based models such as GCN, GAT, and GraphConv. This outcome suggests that capturing GTSs as maximal cliques or clusters is beneficial for enhancing graph representation learning. It is

Table 5.1: *Comparison of HGLA-Pool and baselines. The highest score is in* ***bold****, and the second highest score is in* <u>underline</u> *(OOR referred to as out-of-resources).*

Class	Baselines	PROTEINS	D&D	NCI1	NCI109	COX2	BZR	MUTAGEN	REDM12K	MSRC
GNNs	GCN[2017]	73.77 ± 5.59	75.13 ± 4.14	74.68 ± 3.09	73.47 ± 2.22	78.16 ± 0.85	80.01 ± 2.39	78.18 ± 2.54	23.84 ± 0.92	84.04 ± 6.40
	GAT[2018]	75.30 ± 5.23	76.91 ± 2.68	75.16 ± 3.36	73.78 ± 3.38	78.37 ± 0.66	80.74 ± 2.52	78.99 ± 2.44	21.73 ± 0.03	88.80 ± 4.38
	GraphConv[2018]	74.04 ± 5.07	76.57 ± 3.98	78.15 ± 1.86	77.01 ± 2.52	80.79 ± 4.43	<u>83.51 ± 2.89</u>	79.18 ± 2.24	37.53 ± 1.97	88.66 ± 2.49
Pooling	DiffPool[2018]	77.62 ± 4.97	74.31 ± 2.15	70.70 ± 0.22	70.20 ± 0.24	77.60 ± 2.70	78.90 ± 0.04	71.80 ± 0.15	OOR	83.30 ± 0.51
	SortPool[2018]	74.66 ± 5.08	67.47 ± 6.23	70.58 ± 3.68	68.87 ± 2.38	78.20 ± 0.02	79.70 ± 0.07	75.80 ± 2.43	42.50 ± 0.36	74.70 ± 0.40
	TopkPool[2019]	73.32 ± 6.09	73.85 ± 4.63	74.84 ± 4.62	75.09 ± 2.37	77.93 ± 2.79	81.49 ± 4.17	79.56 ± 1.96	37.56 ± 3.39	88.45 ± 3.70
	SAGPool[2019]	73.50 ± 4.56	72.49 ± 2.87	74.26 ± 1.96	74.94 ± 3.13	78.37 ± 1.86	81.24 ± 4.01	79.25 ± 3.03	36.79 ± 3.53	87.92 ± 3.24
	CliquePool [2019]	73.56 ± 2.86	74.81 ± 3.87	77.26 ± 1.53	76.49 ± 1.14	78.37 ± 1.86	82.17 ± 2.25	78.47 ± 1.62	36.11 ± 2.13	88.72 ± 1.14
	GMT[2021]	75.09 ± 0.59	78.72 ± 0.59	74.21 ± 1.88	71.38 ± 2.03	80.86 ± 0.41	82.25 ± 0.70	80.53 ± 0.11	<u>44.06 ± 0.09</u>	<u>90.53 ± 0.34</u>
	MuchPool[2021]	<u>78.52 ± 3.89</u>	76.48 ± 7.01	74.70 ± 2.25	71.84 ± 2.66	79.27 ± 6.01	80.28 ± 6.93	74.75 ± 2.48	OOR	86.85 ± 3.61
	MPool[2023]	**79.30 ± 3.30**	**81.20 ± 2.10**	77.40 ± 1.90	73.50 ± 2.50	78.10 ± 0.10	78.70 ± 0.11	79.60 ± 3.70	OOR	87.52 ± 0.54
	MAC[2023]	76.08 ± 3.55	<u>79.13 ± 4.70</u>	77.60 ± 1.66	75.84 ± 1.86	78.36 ± 0.16	82.76 ± 5.91	<u>80.33 ± 1.49</u>	42.67 ± 2.23	89.75 ± 0.22
	Quasi-CliquePool[2023]	75.68 ± 1.38	75.30 ± 3.30	<u>78.21 ± 1.83</u>	<u>77.38 ± 2.23</u>	80.15 ± 2.12	82.87 ± 2.58	79.53 ± 1.58	38.21 ± 2.43	88.15 ± 2.22
	Wit-TopoPool[2023]	75.88 ± 5.60	71.30 ± 0.37	70.58 ± 0.29	69.71 ± 0.25	<u>81.73 ± 3.50</u>	79.75 ± 0.14	73.88 ± 1.29	33.87 ± 1.01	89.16 ± 4.00
Proposed	**HGLA-Pool+A_C**	76.58 ± 3.34	77.57 ± 3.01	80.80 ± 1.24	**79.80 ± 2.30**	82.86 ± 3.55	85.93 ± 3.08	**82.40 ± 1.58**	**45.61 ± 2.15**	91.30 ± 2.39
	HGLA-Pool+F_C	73.33 ± 3.18	75.30 ± 3.95	**81.02 ± 1.90**	79.10 ± 1.10	**83.30 ± 3.57**	**87.43 ± 3.87**	82.25 ± 1.56	43.30 ± 2.03	**92.19 ± 2.23**

also noteworthy that CliquePool and DiffPool do not consistently outperform node selection pooling approaches such as SAGPool and TopkPool. This observation further substantiates the idea that integrating the graph's local and global properties can lead to more effective GP methods. We can also see from Table 5.1 that our proposed HGLA-Pool demonstrates considerable improvement in the multi-classes based datasets, with an increase of 1.66% and 1.55% on MSRC_21 and REDDIT-MULTI, respectively. Our analysis identified a significant proportion of isolated nodes and subgraphs in the biological dataset. Furthermore, the graphs in this dataset demonstrate a scarcity of structures resembling maximal cliques. Our approach's effectiveness hinges significantly on identifying and ranking such cliques, leading to challenges in capturing meaningful hierarchical representation structures within these graphs. This structural difference between the dataset and our method's underlying assumption likely contributes to the observed performance decline. To sum up, our proposed HGLA-Pool consistently performs better on seven out of nine benchmarks than baseline pooling techniques.

5.5.3 Ablation Study

This subsection conducts an ablation study on HGLA-Pool by removing independently both folds to verify further where the performance improvement comes from. For con-

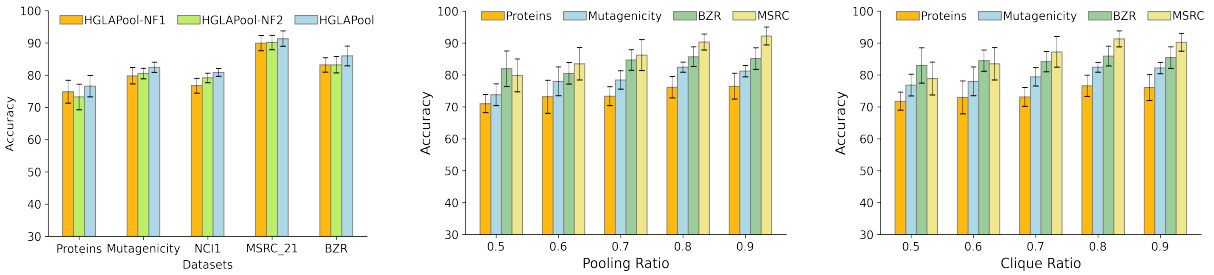


Fig. 5.2: *Effect of dual folds and Hyperparameter analysis on HGLA-Pool performance.*

venience, we name the HGLA-Pool method without the first and the second fold as HGLAPool-NF1 and HGLAPool-NF2, respectively. For these experiments, we chose five different-scale graph datasets covering small and large graphs. From Figure 5.2 (see the leftmost graph), we can see that capturing GTS is crucial in the chemical molecule and MSRC datasets since the GLAPool-NF2 variant obtains a notable performance enhancement as graph attention facilitates the selection of the most informative cliques. Meanwhile, the GLAPool-NF1 variant outperforms GLAPool-NF2 in the PROTEINS as this dataset has limited clique-like structures, which poses challenges for our method in capturing meaningful hierarchical representation structures. This pattern suggests that the LTS with node features are more important for some datasets than the GTS. Overall, HGLA-Pool's ability to learn both the graph's local and global enables it to generate robust graph representations that significantly enhance performance in classification tasks.

5.5.4 Parameter Sensitivity Analysis

In this section, we investigate the sensitivities of the two main parameters, P_r and C_r, influencing the performance of the HGLA-Pool. Figure 5.2 (see second and third graph) summarizes the accuracy of HGLA-Pool under different combinations of these parameters across four different datasets. Our method achieves the highest accuracy for all four datasets when P_r=0.8 and C_r = 0.8, while performance drops with P_r=0.5 and C_r = 0.5. To further investigate these scenarios, we performed a more detailed analysis of the interaction between P_r and C_r, generating a heatmap (see Figure 5.3) for each dataset, illustrating their combined influence. The heatmap reveals specific scenarios where certain combinations of P_r and C_r result in suboptimal performance, particularly in datasets with

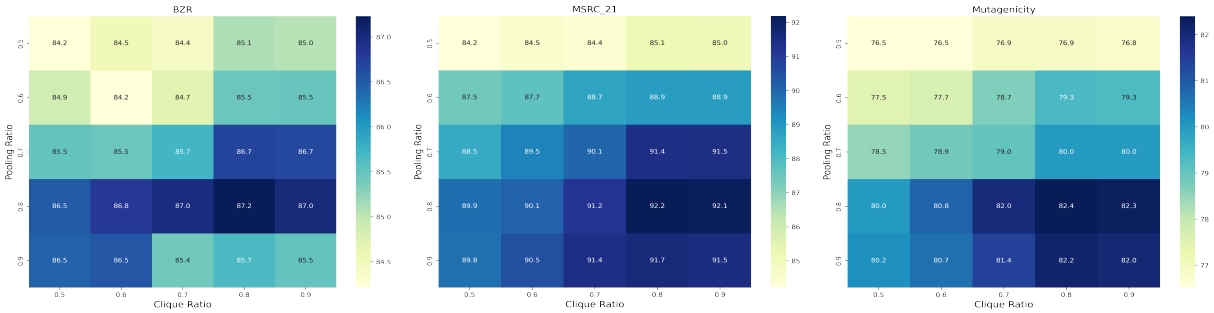
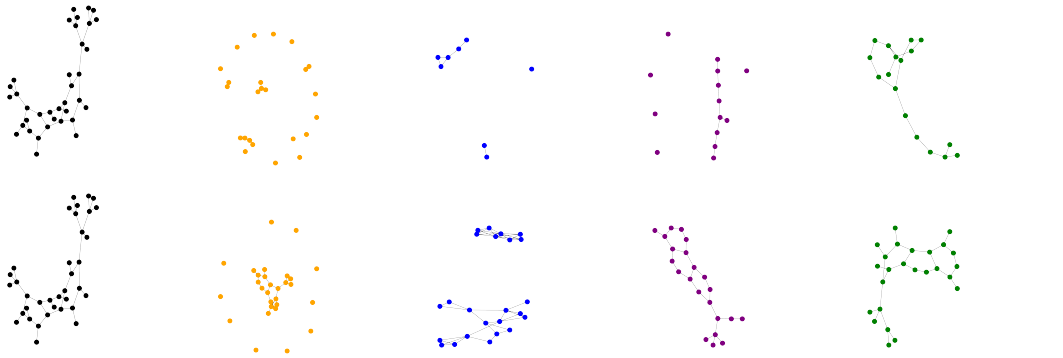


Fig. 5.3: *Effect of different combinations of clique and pooling ratios on HGLA-Pool performance.*

sparsely connected graphs like Mutagenicity. For instance, low values of P_r and C_r (e.g., $Pr = \{0.5, 0.6, 0.7\}$ and $Cr = \{0.5, 0.6, 0.7\}$) tend to exacerbate graph sparsity, leading to a loss of substantial graph structures during pooling, as depicted in Figure 5.3. From our analysis, we also found that a C_r of at least 0.7 is essential for biological, social, and computer vision datasets, while a minimum C_r of 0.8 is necessary for chemical datasets due to their sparse nature. Therefore, the recommended small ranges for C_r and P_r are {0.7, 0.8} for biological, social, and computer vision datasets and {0.8, 0.9} for molecular datasets. Interestingly, the HGLA-Pool demonstrated stability, producing reasonably satisfactory results across most combinations of these parameters.

5.5.5 Graph Visualization for Comparison

To further demonstrate the distinctiveness and superiority of our pooling method compared to existing techniques, we use the Networkx library to visualize the pooling outcomes of HGLA-Pool, CliquePool, TopkPool, and MuchPool. To provide a fair comparison, we build a hierarchical pooling architecture consisting of two layers and set a 0.8 for the C_r and P_r. We randomly selected a graph from the Mutagenicity dataset comprising 36 nodes for the demonstration. The input and pooled graphs of the first and second pooling layers for each method are shown in the first row and the second row of Figure 5.4, respectively. The results demonstrate that the HGLA-Pool, CliquePool and MuchPool mostly preserved the input graph's significant topological structure (ring and branch structures). In contrast, the results of the TopKPool are scattered with numerous isolated nodes, indicating a lack of structural preservation. The results obtained from the



(a) *Input Graph* (b) *TopkPool* (c) *CliquePool* (d) *MuchPool* (e) ***HGLAPool***

Fig. 5.4: *Comparative graph visualization of different polling methods.*

second pooling layer indicate that three baselines encounter challenges with preserving the underlying topology of the initial graph. However, HGLA-Pool can preserve reasonable topological structures, such as dual ring structures present in the initial graph. This underscores the effectiveness of HGLA-Pool since ring structures are crucial in the characterization of molecules.

5.6 Limitations

Our pooling method, rooted in the fundamental notion of cliques, effectively captures the graph's local and global characteristics. However, its efficacy is constrained when applied to datasets with non-clique structures, such as graphs containing many isolated nodes. For example, if the clique-based representation in fold-1 is inaccurate, it might lead to suboptimal outcomes in subsequent steps, such as clique ranking and LocalPool.

5.7 Conclusion and Future Directions

This chapter presents the HGLA-Pool method, an innovative approach for hierarchical graph representation. It employs a two-fold strategy to capture the graph's global and local properties sequentially. Fold-1 captures the global topological structure of a graph by identifying the cliques and overlapping nodes between cliques, coupled with a dynamic fusion scoring method to rank and select significant cliques for pooling. In Fold-2, a

LocalPool layer employing a multi-attention mechanism selects the most informative nodes from these cliques, capturing the LTS. In the end, the outcomes of both folds are integrated sequentially to form a new pooled graph, which retains the original graph's structure and enhances classification performance. The effectiveness of HGLA-Pool has been rigorously assessed through extensive experiments across nine graph classification datasets spanning four distinct domains. Future work will enhance the clique pooling technique to capture a broader range of graph structures, further boosting the graph classification performance.

Chapter 6

Residual Attention and Mixup Augmentation

"If a graph is too large to be drawn on paper in an informative way, then it contains too much information to be useful in anything except in an abstract sense."

— Kary Mullis

6.1 Preamble

This chapter focused on the residual skip connections, attention mechanism, and Mixup augmentation methods for addressing over-smoothing issues in GNNs and enhancing their generalization for node classification tasks. The research and findings discussed in this chapter are based on the following paper:

- Muhammad Affan Abbas, **Waqar Ali**, Florentin Smarandache, Sultan S Alshamrani, Muhammad Ahsan Raza, Abdullah Alshehri, Mubashir Ali. "Residual Attention Augmentation Graph Neural Network for Improved Node Classification." Engineering, Technology & Applied Science Research (ETASR), 14, no. 2 (2024): 13238-13242 [1].

The author has the following contributions:

- **Performing** many experiments.
- **Writing** a part of the paper.
- **Leading** the work to improve the quality of the paper.

6.2 Introduction

Graphs are used in many different disciplines, such as social networks, biological systems, and recommendation engines, to depict intricate relationships and structures. To properly interpret and utilize graph data, it is critical to learn meaningful node representations within these complex network topologies. In light of this, GNNs have become a powerful paradigm that presents a hopeful resolution to this problem [69]. GNNs facilitate efficient node classification, graph classification, and link prediction, among other tasks, by encoding both the local and global graph structure [15]. However, creating effective GNN architectures that meet the unique requirements of node classification is still a challenging issue. Numerous GNN variations have been proposed, each with a unique architectural design and components, creating a vast array of alternatives [154]. The ongoing research for the most effective and efficient GNN architectures that can function effectively on a range of real-world graph data is highlighted by this diversity.

This chapter aims to offer a thorough and workable approach for enhancing GNN performance in the context of node classification in light of these difficulties. The proposed method aims to improve the capabilities of GNN designs while making them simpler, drawing inspiration from recent developments in the field. This approach combines graph convolutional layers with fully connected layers in a simplified architectural layout. It uses Attention Mechanism (AM) [44] and Data Augmentation (DA) strategies, namely MixUp [54], to further enhance the performance of GNNs in node classification. Strategically incorporated into the GNN architecture, these strategies help improve the generalization of the GNN model. Skip Connections (SCs) are used to reduce the accuracy loss caused by over-smoothing. Extensive tests were performed on various node classification tasks

to thoroughly evaluate the performance of the proposed strategies on known benchmarks, such as social networks [8] and citation networks [82]. Concisely, this chapter:

- Developed a Residual Attention Augmentation Graph Neural Network (RAA-GNN) to enhance the evaluation of the node classification task.
- Developed a novel DA method, called MixUpDA, which combines labels and node attributes to produce synthetic data points and improve the model's ability to classify nodes. Additionally, well-designed skip connections and an effective multi-head attention technique were introduced to improve information aggregation and over-smoothing issues, which together improve GNN performance for node classification.
- Evaluated the proposed method on the Twitch social network dataset, and the results showed up to a 1% gain in accuracy, providing further insights for graph-structured data applications.

6.3 Related work

Several studies have investigated SCs, AMs, and DA in the context of graph-based machine learning. Although DA has been beneficial in enhancing model performance in several fields, its implementation in graph-based machine learning has encountered difficulties. Conventional augmentation methods for graph data, including noise addition or perturbing node properties [36], frequently fail because they break the natural graph structure [41]. Furthermore, the addition of synthetic noise can impede the learning and generalization of the model. The proposed MixUpDA strategy [59] provides a logical method to enhance graph data by seamlessly combining node attributes and labels.

Attention mechanisms have revolutionized information aggregation in GNNs by allowing nodes to choose to attend to the relevant neighbors [117]. However, problems with scalability and processing complexity may make them less successful. Current methods are frequently computationally intensive, and therefore, they are unfeasible for large-scale graphs. Currently, the SuperHyperGraph presents the most general form of graph [117].

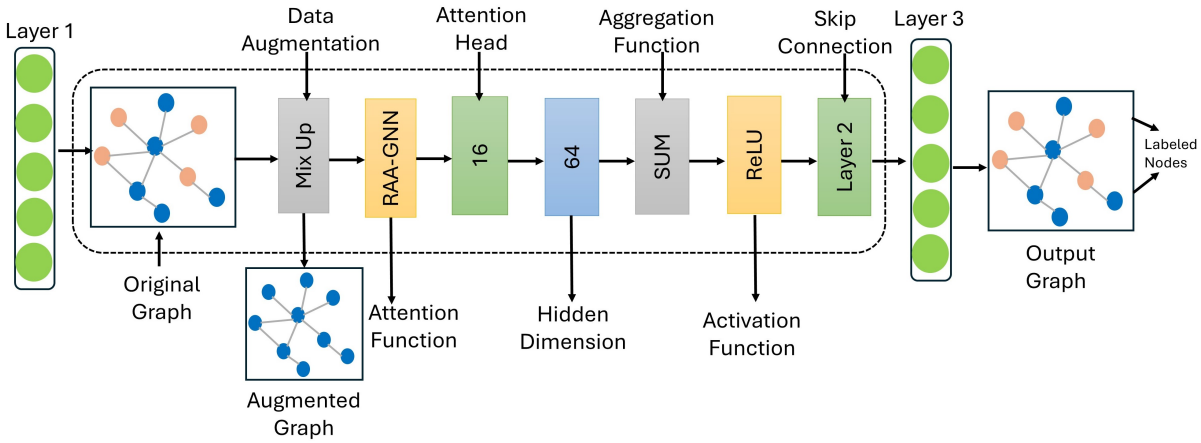


Fig. 6.1: *The RAA-GNN architecture integrates SCs, AMs, and DA for improved node classification.*

These issues are addressed and make it easier to apply attention methods to larger graphs by introducing a multi-head AM that strikes a compromise between expressive capacity and computational efficiency. SCs are important in deep learning architectures because they facilitate the transfer of information between layers [139]. Applying SCs in GNNs has proven difficult, despite their usefulness. Their poor integration can cause over-smoothing, reducing classification accuracy by making nodes indistinguishable through excessive information exchange. This chapter introduces SCs into the GNN design to mitigate the effects of over-smoothing [109], resulting in improved performance without sacrificing accuracy. Consequently, SCs, AM, and DA [122, 110] have all been crucial in the advancement of graph-based machine learning. This chapter addresses these systems' drawbacks by providing a computationally efficient multi-head AM, a more principled approach to DA, and a method for preventing over-smoothing using SCs. Together, these developments enhance node classification in GNNs and enable a greater variety of complicated, real-world graph data to be used in GNNs.

6.4 Methodology

Figure 6.1 shows the architecture of the proposed RAA-GNN model. In the first step, the MixUp augmentation strategy employs a feature-label augmentation method to increase the robustness of the training dataset. Then, the AM is used, which permits the adaptive

weighting of pertinent neighbors, boosting the model's capacity to identify significant local structures and raising classification accuracy all around. Following this, SCs are used to solve the over-smoothing problem of GNNs for node classification.

6.4.1 Node Augmentation Mixup Method

GNNs can be designed with MixUp augmentation as a practical and efficient approach to improve node classification performance. The proposed method is based on meticulous preprocessing of input data represented by X, which guarantees consistency and standardization. MixUp augmentation serves as a dynamic catalyst by carefully combining the training dataset to add robustness and diversity. The amount of augmentation is dynamically influenced by the mixing parameter λ. This coefficient highlights the controlled variability included during training and is randomly generated from a beta distribution with parameters:

$$X_{\text{mix}} = \lambda X + (1 - \lambda) X_{\text{shuffled}} \tag{6.1}$$

MixUp works in unison with the larger GNN design, where SCs and attention processes are essential building blocks to improve the model's comprehension of complex graph structures. The following mathematical formulas capture MixUp's effect on the data and explain how original and shuffled features and labels are well combined, which adds to the model's flexibility:

$$\text{Mixed Data} = \lambda \cdot \text{Batch Data} + (1 - \lambda) \cdot \text{Batch Data[Indices]} \tag{6.2}$$

$$\text{Mixed Labels} = \lambda \cdot \text{Batch Labels} + (1 - \lambda) \cdot \text{Batch Labels[Indices]} \tag{6.3}$$

Integrating MixUp into the GNN model promotes a more robust and flexible learning process for node categorization tasks. The proposed GNN architecture is at the forefront of node classification research because of this deliberate augmentation, which also strengthens the model's ability to generalize across a variety of graph configurations and

enrich the training dataset.

6.4.2 Attention Mechanism for Node Classification

RAA-GNN is used to represent the attention function, which is essential to the model's ability to concentrate on the most pertinent data inside the graph structure. With sixteen attention heads, the model captures many structural details and complex relationships, leading to a thorough comprehension.

The ReLU activation function is used to combine the contributions of each attention head, represented as RAA $GNN_i(X, A)$, to get the final attention scores. Each attention head makes a unique contribution to the overall AM. The adjacency matrix is represented by A. Intricate graph patterns are captured by 64 hidden dimensions, which balance model expressiveness and efficiency. The aggregation function combines data from nearby nodes by applying the summation:

$$H_{\text{agg}} = \sum_{N} H_N \tag{6.4}$$

where N is for the neighbor. By introducing non-linearity, ReLU activation improves the model's capacity to learn intricate relationships:

$$H_{\text{activated}} = \max(0, H_{\text{agg}}) \tag{6.5}$$

6.4.3 Skip Connections

The SCs, denoted by Hskip, enable the smooth transfer of data between the model's layers. These SCs serve to bridge the gap between subsequent layers by integrating the activated features ($H_{activated}$) with the features of the preceding layer ($H_{previous}$), therefore facilitating the transfer and retention of crucial information. By ensuring that important information from previous layers is merged, this additive process improves the model's ability to represent both local and global interdependence.

$$H_{\text{skip}} = H_a + H_p \tag{6.6}$$

where a is the activated and p is the previous. The model's three layers improve node classification by capturing hierarchical representations:

$$H_{\text{output}} = \text{GNN}(H_{skip}^{(3)}) \tag{6.7}$$

The output graph, which displays node classifications based on learned features, is generated by the last layer:

$$Y_{\text{pred}} = \text{Softmax}(H_{\text{output}}) \tag{6.8}$$

This architecture provides a basis for strong node classification in a variety of datasets by utilizing cutting-edge methods to address graph-based learning difficulties.

6.5 Experimental Results and Discussion

Extensive simulations were carried out to evaluate the performance of the proposed GNN with the novel features of DA, AM, and SCs. The findings demonstrate the complex relationships between these elements and the accuracy of node classification on a variety of datasets. The results in Table 6.1 shed important light on the relative importance of the various parts of the proposed GNN model. These features are crucial for accurately capturing complex relationships in graphs, as demonstrated by the model's strong performance across a range of datasets. The results advance the knowledge of GNN architectures and provide useful advice for creating powerful models that are suited to particular uses.

The experimental results demonstrate that the proposed GNN architecture changes increase the model's test set accuracy up to 1%. Adam optimizer, with a learning rate of 0.01, DA, AM, SC, and three GCN layers with the ReLU activation function were utilized in the top-performing design. Table 6.2 shows the reported mean classification accuracy for the fully supervised node classification task for various graph neural network models. The bold numbers represent the best results while the second bests are underlined. The results for GCN, Mix-Hop, and GraphSAGE were obtained from [156]. The results

For GCNII, NodeAug, FSGNN, GPRGNN, and GEOM-GCN were taken from [29, 87]. Traditionally, GNN models such as GCN and GAT have more efficiency on homophily datasets, although they give poor results on datasets with heterophily. Advanced models such as WRGAT, and GPRGNN function are reasonably superior on datasets with both homophily and heterophily. The proposed model performs significantly better on heterophily datasets, particularly with a notable boost on the CiteSeer and Chameleon datasets. Improvements were also noted for the datasets from Actor, Texas, and Cornell. The proposed model achieves consistent and comparable performance to state-of-the-art methods on homophily datasets. It also performed exceptionally well in the evaluation of the new Twitch social network dataset for node classification, demonstrating its flexibility to various graph architectures [107].

Table 6.1: *Node Classification Accuracy (%) for the Proposed Model*

Model	Cora	CiteSeer	PubMed	Chameleon	Wisconsin	Texas	Cornell	Squirrel	Actor
Proposed	88.94	78.32	89.14	79.31	86.14	86.21	86.57	72.22	36.71
Without DA	85.33	73.52	87.37	78.30	84.90	85.12	86.21	71.39	34.99
Without AM	86.22	75.59	87.13	79.01	84.90	84.22	84.43	71.35	33.78
Without SC	85.59	77.23	86.97	78.30	83.09	84.32	85.55	71.87	32.34

Table 6.2: *Node Classification Accuracy (%) for Different Models on Various Datasets*

Model	Cora	CiteSeer	PubMed	Chameleon	Wisconsin	Texas	Cornell	Squirrel	Actor	Mean Acc.
GCN	87.28	76.68	87.38	59.82	59.80	59.46	57.03	36.89	30.26	61.62
GraphSAGE	86.90	76.04	88.45	58.73	81.18	82.43	75.95	41.61	34.23	69.50
MixHop	87.61	76.26	85.31	60.50	75.88	77.84	73.51	43.80	32.22	68.10
GEOM-GCN	85.27	77.99	90.05	60.90	64.12	67.57	60.81	38.14	31.63	64.05
GCNII	88.01	77.13	**90.30**	62.48	81.57	77.84	76.49	N/A	N/A	–
NodeAug	86.20	75.40	82.10	N/A	N/A	N/A	N/A	–	–	–
GPRGNN	88.49	77.08	88.99	66.47	85.88	**86.49**	81.89	49.03	36.04	73.37
FSGNN	87.61	77.17	89.70	78.93	**87.25**	85.90	86.23	**73.32**	34.89	77.88
RAA-GNN	**88.94**	**78.32**	89.14	**79.31**	86.14	86.21	**86.57**	72.22	**36.71**	**78.17**

Table 6.3: *Results on a New Twitch Social Networks Dataset*

Dataset	Train Loss	Train Accuracy	Validation Loss	Test Accuracy
Twitch Social Networks	0.2463	0.8989	0.9186	0.9046

6.6 Conclusion

This chapter presents the RAA-GNN model for node classification that incorporates SC, AM, and DA, showing that these elements can work together to improve its discriminative ability. While SCs handle over-smoothing issues, the AMs specifically allow the model to perform better on graphs for node classification. DA is an essential component that adds variation to the training dataset and promotes robustness against overfitting. The experiments highlighted each component's independent effectiveness, as well as their combined impact on overall performance. The proposed model demonstrated its adaptability by consistently outperforming

state-of-the-art approaches in node classification across multiple datasets. In summary, this chapter extends GNN architectures and sheds light on the complex interactions between SC, DA, and AM. It also sets a new Sota node classification in graph structure learning through the first attempt to integrate SC, AM, and DA into RAA-GNN, thus advancing our understanding of GNNS.

6.7 Acknowledge

The authors would like to acknowledge the Deanship of Scientific Research, Taif University for helping with this work.

Chapter 7

Topology-Aware Augmentation

"Life is a math equation. In order to gain the most, you have to know how to convert negatives into positives."

— Unknown

7.1 Preamble

In this chapter, we identified the limitations of the existing random node dropping augmentation method. We proposed a novel Node-Dropping Augmentation (NDAUG) method using node degree as a criterion to selectively drop less important nodes (low-degree nodes) and preserve essential graph structures, generating diverse and informative augmented graphs. Further, in the case of isolated nodes, we develop a structure learning method to reconnect these isolated nodes by learning attention-based relationships between nodes. The research and findings discussed in this chapter are based on the following paper:

- **Waqar Ali**, Sebastiano Vascon, Thilo Stadelmann, and Marcello Pelillo. "Topology-Aware Node Dropping Augmentation for Graph Classification"; Submitted to European Symposium on Artificial Neural Networks (ESANN, 2025).

The author has the following contributions:

- **Developing** the overall pipeline of the algorithm.

- **Writing** the complete code.
- **Performing** all experiments.
- **Writing** a significant part of the paper.

7.2 Introduction

Recently, GNNs, a specialized form of deep learning designed for graph-structured data, have shown effectiveness in various tasks of classifying graphs and learning graph representations, including predicting chemical molecular properties and analyzing social networks [72]. However, similar to deep learning models in image processing, GNNs are prone to overfitting, particularly when dealing with limited datasets [101]. Data augmentation methods are known for their efficiency and effectiveness in mitigating overfitting issues in deep learning networks [106]. These methods generate new synthetic samples from the existing training data, providing a straightforward and cost-efficient method to enhance the generalization of a deep model. Data augmentation has proven helpful in the computer vision domain [101], but applying these techniques to graph-structured data presents unique challenges due to the graphs' irregular structures. Some recent works focused on developing node feature-based graph mixup augmentation methods for node-level tasks [126], others like Kong et al. [73] recommend enhancing node features through adversarial learning. While graph structural learning augmentation methods [132, 144, 106] usually modify the graph structure by randomly dropping/adding nodes or edges and generating new augmented graphs for graph-level tasks. However, current graph structural learning augmentation methods, such as random node-dropping, often fail to preserve the original graph's essential topological structures during the augmentation process, potentially affecting the performance of the graph classification tasks. This random node-dropping method can also disconnect closely related nodes, generating isolated nodes in the augmented graph [132], which may affect the efficiency of the GNN's message passing mechanism.

To this end, we introduce a novel node-degree based Node-Dropping Augmentation

method for graph classification tasks to address the issues in existing augmentation approaches. The node degree is an important concept in graph theory because nodes with higher degrees often correspond to critical points in various graphs, such as road, social, or protein-protein interaction graphs, which are essential to identifying the structure and functionality of the entire graph [144]. Our method generates augmented graphs by leveraging the node-degree concept to remove less important low-degree nodes while preserving significant topological structures formed by high-degree nodes, thereby maintaining the essential characteristics of the original graph. However, in applications such as molecular datasets with toxic/non-toxic compounds, the functional groups often contain low-degree nodes such as benzene or carbon motif ring structures [144]. To handle such cases, NDAUG identifies and preserves these significant structures, ensuring the retention of vital low-degree nodes. Additionally, we propose an attention mechanism to reconnect isolated nodes that may result from the node-dropping process, thus maintaining the connectivity in the graph and enhancing the performance of GNN layers. To summarize, the main contributions of this chapter are as follows:

- We propose a novel NDAUG augmentation method that uses node degree to remove less important nodes while preserving key topological structures.
- We introduce an attention-based structure learning method to reconnect isolated nodes, maintaining graph connectivity within the augmented graphs and enhancing GNN performance.
- Experiments on eight benchmark datasets show that NDAUG outperforms existing augmentation methods with a $2-5\%$ improvement.

7.3 Related Work

Recently, GNNs [72, 91] have emerged as a powerful tool and attained significant achievements in graph classification tasks. Despite GNNs' success, data availability is a significant limitation in many graph classification problems [144]. For instance, GNNs have

been widely used in predicting molecular properties, where obtaining labeled molecule data often involves complex manual laboratory procedures. This leads to a lack of adequately labeled samples for GNNs to attain a promising prediction performance.

Data augmentation methods are noted for their efficiency and effectiveness in generating new synthetic samples from existing training data, thereby enhancing the generalization capabilities of deep models. This strategy is preferred over more resource-intensive methods like gathering extra real data or making significant changes to the model architecture or training algorithms. Data augmentation methods have proven helpful in the fields of CV and NLP [101]. However, applying such techniques to graphs is more complex due to their non-Euclidean nature, where nodes are irregularly connected by edges [153], presenting unique challenges in augmentation. Recent works have focused on developing node feature space-based augmentation methods for node-level tasks [126, 132], and a limited number of attempts [133] have been undertaken for graph classification tasks. For example, DropEdge [106] employs a random method to remove a uniform portion of edges and generate augmented graphs to enhance the robustness of the GNN model during test-time inference, M-evolve [155] methodology uses motif-similarity mapping methods to add or remove edges connecting nodes that are predicted to have similar labels with a high level of motif-similarity score, and DropNode [144] randomly drop a certain portion of nodes from the original graph and generate a new graph. The random node-dropping method can disconnect closely related nodes, generating isolated nodes in the augmented graph [132]. This may affect the efficiency of the GNN's message passing mechanism.

Furthermore, the authors of [132] develop a GraphCrop method, which generates various cropped-augmented graphs using a node-centric strategy. However, current graph augmentation methods, such as random node-dropping, often fail to preserve the original graph's essential topological structures during the augmentation process, potentially affecting the performance of the graph classification tasks. Hence, this chapter aims to tackle this issue by introducing a node-dropping augmentation method. This method removes less important low-degree nodes while preserving the essential topological struc-

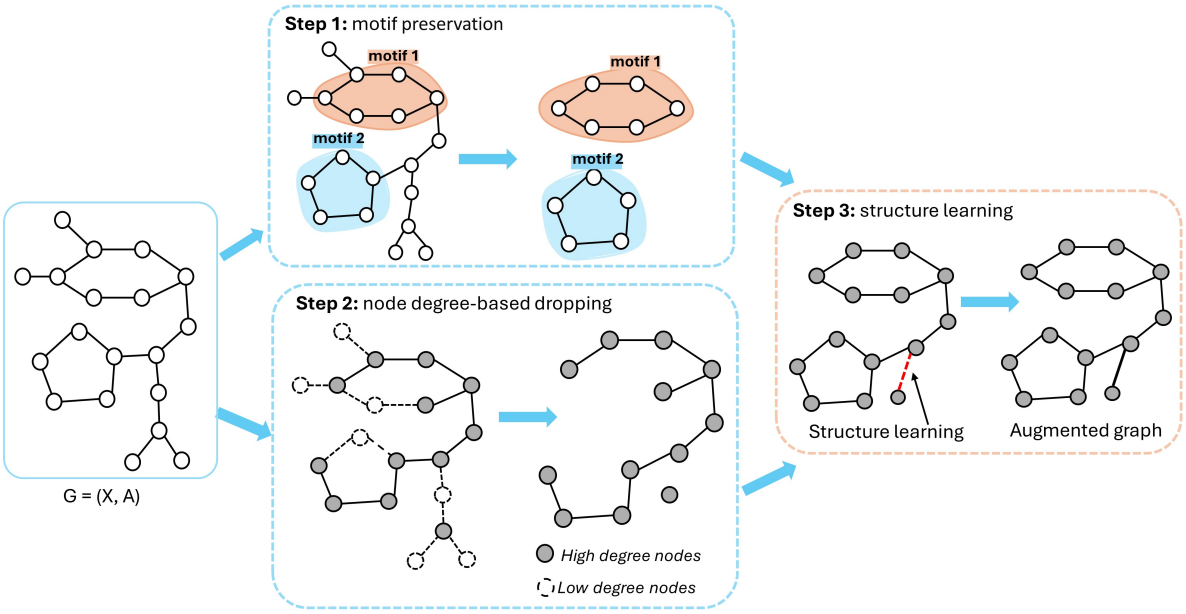


Fig. 7.1: *The pipeline of the proposed NDAUG method. The initial step identifies the important structural motifs of the input graph. Step 2 removes the low-degree nodes while maintaining the key topological structures formed by high-degree nodes. The last step generates the final augmented graph to preserve the identified significant motif structures of step 1 and applies a structure learning method to retain the connectivity of the augmented graph by reconnecting any isolated nodes resulting from the node-dropping process.*

tures.

7.4 Methodology

This section first defines the mathematical notations and problem formulation of graph classification tasks. Then, we present a detailed description of the proposed NDAUG method. Figure 7.1 shows the working pipeline of the NDAUG method for generating an augmentation graph.

7.4.1 Problem Formulation

For the graph classification tasks, each data point in the dataset $D = \{(G_i, y_i)|i = 1, ..., t\}$ consists of a graph G_i and its corresponding label y_i. We split the dataset D into training, validation, and testing sets, depicted as D_{train}, D_{val}, and D_{test}, respectively (for more detail, see the experimental section 7.5.1). Specifically, we aim to generate

new data samples for a classifier such as $G \in D_{train}$ to a new augmented graph G' like $f : (G, y) \mapsto (G', y)$ where y is the label of G. The augmented set D'_{train} add with D_{train} to produce the final training set: $D^{new}_{train} = D_{train} + D'_{train}$.

7.4.2 Motifs Preservation

Motifs are small, recurrent, and connected subgraphs that play a crucial role in measuring the connectivity patterns of nodes within a graph [45]. The importance of motifs in the analysis of graphs has been widely recognized since they serve as key indicators in revealing the fundamental structure and functionality of complex networks [120]. Figure 7.1 illustrates the topological structures of two benzene rings with 5 and 6 nodes, which we refer to as cyclic motif structures, and these hold substantial chemical significance in molecular datasets. These cyclic motif structures often determine the structural and chemical properties of the molecule, impacting its behaviour and interactions [89]. Therefore, preserving such motifs is critical in graph classification like drug discovery or toxicity prediction, where losing these structures could lead to inaccurate interpretations of the molecule's properties. Current augmentation techniques, like DropNode [144], employ a random process to drop a substantial number of nodes from the graph without preserving the underlying graph motif structures. Therefore, our node-dropping method first preserves these essential topological motif structures during augmentation. Let $M(v)$ be an indicator function defined as:

$$M(v) = \begin{cases} 1 & \text{if } v \in M, \\ 0 & \text{otherwise.} \end{cases} \tag{7.1}$$

A node v is preserved in the augmented graph if $M(v) = 1$. Here, M denotes the set of nodes that are part of identified motifs. We implement this $M(v)$ indicator function using the cycle-basis function of the Netwrokx library to identify all cyclic motif structures of nodes 4, 5, and 6.

7.4.3 Node Degree-based Dropping

After preserving significant structural motifs in the graph, we use the concept of node degree to select the most important nodes to generate an augmented graph. In simple terms, the nodes with a high degree indicate their significance within the entire graph [144]. For example, high-degree nodes in road networks often correspond to major intersections or hub areas [144]. Similarly, in social and protein-protein interaction networks, high-degree nodes often represent influential individuals or key connectors within the community and correspond to crucial proteins that interact with many other proteins. So, this step removes low-degree nodes from the graph to generate an augmented graph. The degree of a node $d(v)$, which corresponds to the number of edges connecting to v, can be formally expressed as:

$$d(v) = |\{u \in V : (u, v) \in E \vee (v, u) \in E\}| \tag{7.2}$$

The following conditions guide the node-dropping process: nodes are considered to be removed if their degree is less than or equal to the degree threshold T_d, their associated drop probability prob_v is below a threshold p, and they are not part of any identified motifs M:

$$V_{\text{drop}} = \{v \in V : d(v) \leq T_d \text{ and } \text{prob}_v < p \text{ and } v \notin M\} \tag{7.3}$$

The augmented graph $G' = (V', E')$ is then derived by removing V_{drop} from V, and accordingly adjusting E to exclude edges incident to any node in V_{drop}. See Algorithm 2 for more details. Formally, the augmented graph is constructed as follows:

$$V' = V \setminus V_{\text{drop}}; \quad E' = \{(u, v) \in E : u \notin V_{\text{drop}} \wedge v \notin V_{\text{drop}}\} \tag{7.4}$$

7.4.4 Structure Learning Method

This section explains the working pipeline of our proposed structure learning mechanism, which learns a refined augmented graph structure. As mentioned earlier, the node-

Algorithm 2: Node-Dropping Augmentation NDAUG

Input: A graph G with vertices V and edges E, and Drop Probability P
Output: Augmented graph G'

Initialize an empty set D to store nodes to be dropped;
Identify the essential topological structural motifs M in G;
for *each vertex* v *in* V **do**
    Calculate node degree $\deg(v) = |\{e \in E : v \in e\}|$;
    Generate a random drop probability $prob_v$ for node v;
    if $deg(v) \leq T_d$ ***and*** $prob_v < P$ ***and*** $v \notin M$ **then**
        Add v to D;
Remove the nodes in D from V to get V';
Update the set of edges $E' = \{e \in E : \nexists v \in D$ such that $v \in e\}$;
Re-index nodes in V' starting from 1 up to the size of V';
return $G' = (V', E')$;

dropping augmentation operation can lead to the disconnection of closely related nodes in the augmented graph $G' = (V', E')$. This loss of graph structure information further hinders the message passing procedure of the GNN [132] (also see an example in Figure 7.1). We used a GAT layer, which takes structural information A' and hidden representation X' as input to transform node features by aggregating information from their neighborhoods and resulting attention score vector for each node (as shown in the Equation 7.5). We use cosine similarity to determine the similarity between the transformed features of isolated nodes and those in the main graph. For any isolated node i and a non-isolated node j, their similarity is calculated as follows:

$$F' = \text{GAT}(X', A'); \quad S_{ij} = \frac{F'_i \cdot F'_j}{\|F'_i\| \|F'_j\|} \tag{7.5}$$

The process of reconnecting isolated nodes involves identifying these nodes, represented as the set $I \subseteq V'$ in G'. For each isolated node $i \in I$, we search within $V' \setminus I$ to find a node j that has the highest cosine similarity score with i and then create a new edge between nodes i and j.

$$E' = E' \cup \{(i, j) : i \in I, j = \underset{k \in \{V' - I\}}{\text{argmax}} S_{ik}, S_{ij}\} \tag{7.6}$$

Table 7.1: *Comparison of NDAUG and baselines. The bold text represents the best performances.*

Methods	BZR	COX2	NCI1	MUTAGEN	PROTEINS	DD	IMDB-B	RED12K
No Augmentation	$79.42_{\pm 1.97}$	$79.50_{\pm 1.50}$	$75.50_{\pm 1.50}$	$77.18_{\pm 1.86}$	$72.60_{\pm 3.92}$	$76.50_{\pm 2.35}$	$68.20_{\pm 6.55}$	$38.80_{\pm 3.12}$
NodeDrop[144]	$80.31_{\pm 6.50}$	$78.39_{\pm 3.98}$	$76.47_{\pm 2.03}$	$77.77_{\pm 2.57}$	$73.86_{\pm 2.51}$	$76.66_{\pm 3.89}$	$68.20_{\pm 4.98}$	$41.13_{\pm 1.46}$
EdgeDrop[106]	$81.97_{\pm 3.50}$	$79.88_{\pm 6.44}$	$77.93_{\pm 1.33}$	$79.18_{\pm 1.89}$	$73.41_{\pm 4.45}$	$74.03_{\pm 4.09}$	$69.40_{\pm 4.20}$	$40.53_{\pm 2.61}$
GraphCrop[132]	$79.84_{\pm 3.40}$	$79.76_{\pm 4.64}$	$77.67_{\pm 2.50}$	$79.54_{\pm 2.59}$	$73.10_{\pm 3.50}$	$76.86_{\pm 3.46}$	$70.87_{\pm 3.51}$	$40.81_{\pm 2.71}$
Gmixup[59]	$82.15_{\pm 4.25}$	$78.34_{\pm 5.20}$	$77.18_{\pm 1.56}$	$80.59_{\pm 2.31}$	$72.10_{\pm 5.71}$	$75.29_{\pm 1.69}$	$70.31_{\pm 3.36}$	$41.10_{\pm 2.31}$
M-evolve [155]	$79.30_{\pm 1.53}$	$77.74_{\pm 3.41}$	$77.37_{\pm 2.86}$	$78.84_{\pm 2.25}$	$72.31_{\pm 3.62}$	$76.81_{\pm 2.34}$	$69.40_{\pm 3.81}$	$40.34_{\pm 3.87}$
Mixup [133]	$81.20_{\pm 3.51}$	$79.81_{\pm 4.41}$	$77.08_{\pm 2.10}$	$79.81_{\pm 2.13}$	$74.10_{\pm 3.35}$	$75.40_{\pm 2.80}$	$69.30_{\pm 3.20}$	$40.66_{\pm 2.17}$
NDAUG	$\mathbf{86.16}_{\pm 3.21}$	$\mathbf{81.28}_{\pm 3.37}$	$\mathbf{80.01}_{\pm 2.51}$	$\mathbf{82.01}_{\pm 2.21}$	$\mathbf{75.65}_{\pm 2.54}$	$\mathbf{79.31}_{\pm 3.31}$	$\mathbf{72.40}_{\pm 3.20}$	$\mathbf{45.95}_{\pm 1.61}$

By effectively reconnecting isolated nodes, we maintain the graph's connectivity and ensure the continuity and efficacy of the message passing mechanisms in the GNN, which are essential for accurate graph classification tasks.

7.5 Experiments and Discussion

This section evaluates the efficacy of the proposed NDAUG method on eight classification datasets, including BZR, COX2, NCI1, and MUTAGENICITY for molecular compound classification, DD and PROTEINS for protein categorization, IMDB-M, and REDDIT-MULTI12K for social network classification [92]. These datasets have been widely used as benchmarks for graph classification tasks, as demonstrated in this chapter [138]. Our findings demonstrate that NDAUG consistently outperforms the existing baseline approaches. Furthermore, a graph visualization comparison and a comprehensive series of ablation studies are conducted to evaluate the individual contributions of different components inside the NDAUG method.

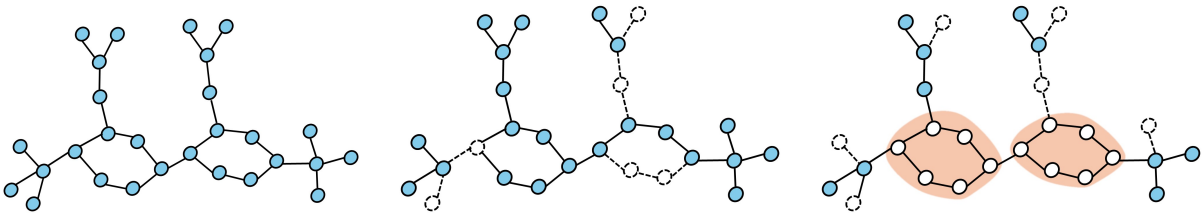
7.5.1 Baseline Methods

We follow numerous prior research studies [144, 106] and employ the 10-fold cross-validation method, dividing the datasets into training, validation, and testing sets with ratios of 80%, 10%, and 10%, respectively. We report the average test accuracy over ten different runs. The training process utilizes the early-stop mechanism, which terminates when the loss value of the validation set does not decrease for 50 consecutive epochs. We

fine-tune hyperparameters for all models on each dataset within the specified range, as follows: 1) initial learning rate $\in \{0.01, 0.0005\}$, 2) embedding dimensions $\in \{64, 128\}$, 3) batch size $\in \{32, 64, 128\}$, 4) DropEdge and DropNode drop ratio $\in \{20\%, 40\%\}$, 5) node degree value $\in \{1, 2, 3\}$ and number of GNN layers $\in \{2, 3, 4\}$. We utilize the Adam optimizer to initialize our model and apply a negative log-likelihood loss function for training. We compare our NDAUG methods, which do not use any data augmentations, and six graph augmentation baseline methods, including DropNode [144], DropEdge [106], GraphCrop [132], Gmixup [59], M-evolve [155] and Mixup [133]. We use the same GNN model [91] and hyperparameter setting for NDAUG and all baseline augmentation approaches to ensure a fair comparison. The source code for the NDAUG method will be available at the given link.

7.5.2 Performance Comparison and Graph Visualization

Table 7.1 compares the performance of our proposed NDAUG and baseline methods across the eight graph classification benchmark datasets. Significantly, NDAUG demonstrates superior performance over all baseline methods in all datasets. Specifically, when comparing with the GCN baseline, NDAUG shows a relative accuracy improvement on the BZR, NCI, MUTAGENICITY, DD, and REDIT12K datasets by margins of 6.74%, 4.51%, 4.83%, 2.45%, and 7.15%, respectively. This advancement underscores the efficacy of graph data augmentation in enhancing GNN performance for graph classification tasks. Moreover, NDAUG consistently surpasses traditional augmentation methods like NodeDrop and EdgeDrop. In the realm of chemical molecule datasets, NDAUG outperforms these baselines by an average of 2-4% in BZR, NCI1, and MUTAGENICITY and by 1.4% in COX2. Across the biological and social network datasets, NDAUG achieves an average improvement of 2.0%, 4.0%, respectively. Existing augmentation methods, such as NodeDrop and EdgeDrop, have limited performance because they randomly drop nodes or edges without preserving the connectivity between the nodes in the augmented graph. This destroys the original graph's essential topological structures, leading to the loss of essential label-related information. We additionally provide graph visualizations



(a) *Training on the NCI1 dataset without any augmentation method obtains an average testing accuracy of 78.44±2.71.*

(b) *The random node-dropping [144] method training achieves an average testing accuracy of 76.47±2.03.*

(c) *Training with our proposed NDAUG method obtains an average testing accuracy of 80.01±2.51.*

Fig. 7.2: *A comparison of NDAUG and the random node-dropping method on the NCI1 dataset shows significant differences. Figure (a) displays a random NCI1 graph with crucial cyclic carbon structures. Figures (b) and (c) illustrate augmented graphs generated by NDAUG and Nodedrop [144], respectively. The random DropNode method degrades classification performance by dropping key nodes from carbon structures. In contrast, NDAUG preserves these structures and enhances classification performance.*

to represent the effect of different augmentation techniques in Figure 7.2. Our analysis, supported by the success of NDAUG on graph datasets, validates the effectiveness of our proposed NDAUG method. This advancement not only sets a new standard in graph augmentation but also opens the potential for future analyses to enhance the performance of GNNs. The overall time complexity of NDAUG, which depends on determining motifs and calculating pairwise similarity, is $O(|V|^c + |V|^2 d)$, where $|V|$ is the number of nodes, d is the dimension of node features, and c represents the complexity of detecting cyclic motifs.

7.5.3 Ablation Studies

This section performs an ablation study on NDAUG by removing three components to verify further where the performance improvement comes from. For convenience, we name the NDAUG method without the node degree measurement, motif structures, and structure learning components as NDAUG w/o NDM, NDAUG w/o MS, and NDAUG w/o ST, respectively. For ablation study experiments, we train GCN-based [91] classification models on four different-scale graph datasets covering small and large graphs and employ the same parameter setting as Section 7.5.1. The results presented in Table 7.2 highlight the considerable impact of node degree measurement and motif structures, particularly

within the domains of chemical molecules, since the preservation of essential motif graph structures such as cyclic benzene with node degree is especially useful to maintain the important graph structures within the augmented graphs. Furthermore, removing the structure learning strategy significantly degrades the performance of NDAUG in NCI and BZR because these datasets are sparse, resulting in augmented graphs containing isolated nodes. It is demonstrated that structure learning, node degree measurement, and preservation of essential graph motif structures are key success factors of NDAUG in generating augmented graphs.

Table 7.2: *Results of ablation studies about different NDAUG components.*

Architecture	Mutagenicity	NCI1	BZR	IMDB-B
NDAUG	**82.021 ± 2.21**	**80.34 ± 2.39**	**86.16 ± 3.21**	**72.70 ± 2.71**
NDAUG w/o ST	80.90 ± 2.27	79.10 ± 2.30	84.31 ± 3.11	71.40 ± 4.01
NDAUG w/o MS	80.30 ± 2.96	79.87 ± 2.10	84.41 ± 3.01	70.80 ± 3.61
NDAUG w/o NDM	78.95 ± 3.01	77.83 ± 3.03	81.61 ± 3.51	70.80 ± 3.61

7.5.4 NDAUG with Different GNN models and Graph Pooling

In addition to the ablation experiments, we extend our analysis of NDAUG to explore its compatibility and enhancement potential with other Graph Neural Networks. To this end, we conducted a series of experiments integrating NDAUG into three commonly used GNN models: GCN [72], GIN [138], and Graphconv [91]. In the experimental settings, we set all parameters uniform for each of the networks. Each experiment employed a three-layer GNN model for classification, leveraging a 10-fold cross-validation scheme, and reported the average accuracy with standard deviation over five runs. The findings, encapsulated in Table 7.3, are quite illuminating. They consistently demonstrate that NDAUG significantly boosts the performance of each GNN architecture and MincutPool across graph classification tasks. This not only provides evidence for NDAUG's adaptability but also highlights its effectiveness as a powerful augmentation method that can enhance the performance of many GNN models and graph pooling methods such as MincutPool.

Table 7.3: *Performance of NDAUG with graph pooling and different GNN models. The mean test accuracy (%) with standard deviations from 10-fold cross-validation over 5 runs is reported. The bold text represents the highest accuracy.*

Architecture	Mutagenicity	NCI1	NCI109	MSRC21	IMDB-B
Vanilla GCN	77.18 ± 1.86	75.50 ± 1.50	75.26 ± 2.92	86.80 ± 4.22	59.90 ± 2.96
NDAUG + GCN	**79.89 ± 2.97**	**77.47 ± 2.38**	**76.92 ± 1.61**	**90.23 ± 3.58**	**61.10 ± 3.45**
Vanilla GIN	80.58 ± 1.76	76.52 ± 3.17	75.21 ± 4.06	87.39 ± 4.79	65.80 ± 3.97
NDAUG + GIN	**81.19 ± 2.49**	**78.30 ± 1.98**	**77.37 ± 2.48**	**90.92 ± 4.27**	**69.30 ± 4.24**
Vanilla Graphconv	79.78 ± 2.19	78.44 ± 2.71	78.71 ± 2.12	89.40 ±3.40	68.20 ± 6.55
NDAUG + Graphconv	**82.01 ± 2.21**	**80.01 ± 2.51**	**79.81 ± 2.22**	**92.03 ±3.23**	**72.70 ± 2.71**
Vanilla MincutPool	76.86 ± 2.86	74.56 ± 2.71	73.52 ± 3.02	87.91 ± 4.65	68.96 ±5.36
NDAUG + MincutPool	**78.12 ± 1.94**	**76.33 ± 2.68**	**74.58 ± 2.21**	**90.12 ± 3.12**	**70.01 ± 4.36**

7.6 Conclusion and Future Work

This chapter introduced a novel data augmentation method named NDAUG for graph classification tasks. At its core, NDAUG used the concept of node-degree measurement to strategically drop less important low-degree nodes from the original graph. This approach is carefully balanced to maintain essential topological motif structures within the augmented graph, even those typically associated with low-degree nodes. Furthermore, we proposed a structure learning technique that employs an attention mechanism to reconnect disconnected nodes to maintain graph connectivity within the augmented graphs and enhance GNN performance. Comprehensive experiments on eight graph classification datasets demonstrated a notable enhancement in the accuracy of up to 5% compared to the existing baselines. In future work, we plan to enhance our NDAUG approach by incorporating edge perturbation techniques to identify key topological substructures and extend its application to node-level tasks.

Chapter 8

Community-Hop Mechanism for Graph Neural Networks

"A representation learning algorithm can discover a good set of features for a simple task in minutes, or for a complex task in hours to months."

— Ian Goodfellow

8.1 Preamble

This chapter proposed a Community-HOP-based GNN model for dealing with homophilic and heterophilic graph structures. Specifically, we incorporated valuable insights from the graph community structure to guide the feature aggregation process of the GNN layer, enabling it to learn diverse graph properties and improve performance on node-level tasks. The research and findings discussed in this chapter are based on the following paper:

- Ahmed Begga, **Waqar Ali**, Gabriel Niculescu, Francisco Escolano, Thilo Stadelmann, and Marcello Pelillo. "Community-Hop: Enhancing Node Classification through Community Preference." Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition and Structural and Syntactic Pattern Recognition (S+SSPR 2024) [16].

The author has the following contributions:

- **Writing** a considerable code.
- **Performing** a considerable experiments.
- **Writing** a significant part of the paper.

8.2 Introduction

GNNs have proven to be powerful methods for analyzing graph-based data, finding use in diverse areas such as social network analysis and predicting molecular properties [72, 125, 56]. However, conventional GNN architectures often face challenges in capturing complex structural information and relationships between nodes at various distances, particularly in graphs with heterophilic properties (where connected nodes have dissimilar labels) [156]. Recent advancements in GNN design have addressed these limitations by incorporating higher-order neighborhood information. Models such as MixHop [3] and FSGNN [86] have demonstrated the effectiveness of aggregating features from nodes at different hop distances. These approaches allow for more flexible feature mixing and improved performance on various graph datasets.

While these methods have shown promise, they often treat all neighbors equally within each hop distance, potentially overlooking important structural information encoded in the graph's community structure. Community detection in graphs has long been a subject of study in network science [114, 128], with spectral methods providing powerful tools for identifying clusters of densely connected nodes [17].

This chapter develops a novel GNN layer that leverages graph community structure to guide the feature aggregation process. By combining spectral community detection techniques [128] with a modified transition matrix for inter-community hops, our approach aims to prioritize information flow within and between communities in a more meaningful way. This community-aware feature aggregation strategy allows the model to capture both local (by just combining community nodes) and global (by using the same GNN for all the clusters) graph structures more effectively.

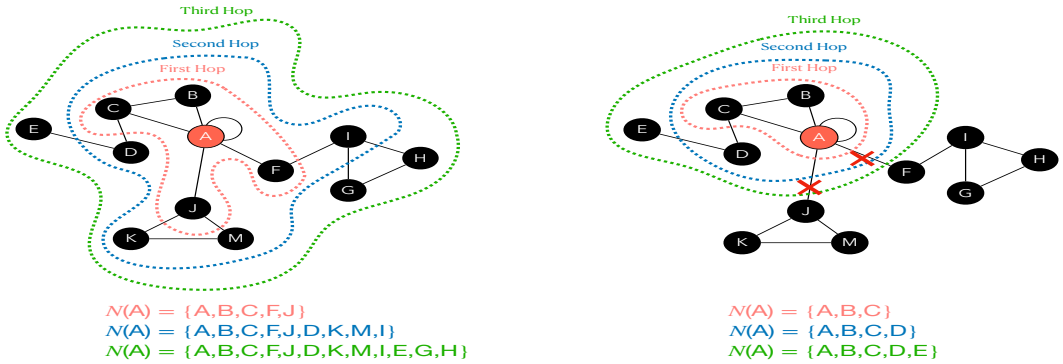


Fig. 8.1: *Traditional Hop vs Community Hop. Evolution of the neighborhood of A, $N(A)$, in traditional hops (left) and in our approach, Community Hop (right).*

8.3 Related Work

Recent years have seen significant advancements in GNN architectures, particularly in addressing the challenges of heterophily and over-smoothing. These innovations have largely focused on modifying the feature aggregation process and leveraging higher-order neighborhood information. Several approaches have been proposed to tackle the heterophily problem, where connected nodes may have dissimilar features or labels. H2GCN [156] introduced ego- and neighbor-embedding separation, along with the exploration of higher-order neighborhood structures. GPR-GNN [31] minimizes over-smoothing by integrating the PageRank method with GNNs. Furthermore, GGCN [142] addresses heterophily and over-smoothing issues by utilizing degree corrections and signed messages.

Interestingly, studies have revealed that basic models like Multi-Layer Perceptrons (MLPs) and LINK [83] can occasionally surpass conventional GNN architectures when dealing with heterophilic datasets. This observation has led to the development of hybrid methods that merge node features with graph-based representations. A prominent example is LINKX [83], which integrates MLPs for node features with LINK regression, showing promising performance on heterophilic graphs.

Another line of research has focused on aggregating features from neighbors at different distances. MixHop [3] and FSGNN [86] utilize the transition matrix's powers to capture multi-hop neighborhood information. FSGNN uses a regularizer method, such as softmax and L2-Normalization in GNN's layers.

Recent work has also explored novel ways to address the over-smoothing problem in deeper GNN architectures. Ordered GNN [118] proposes an approach that aligns the hierarchy of a rooted-tree with ordered neurons in node embeddings, effectively preserving information from different neighborhood depths.

While these advancements have significantly improved GNN performance on various graph types, there remains room for innovation in leveraging graph structure more effectively, particularly in the context of community detection and inter-community information flow. Our proposed method builds on these insights by incorporating spectral clustering and community hops, offering a novel approach to enhance GNN performance across diverse node-level predictions.

8.4 Preliminaries

In this section, we define the mathematical notations used in this chapter. We use the adjacency matrix $A \in \{0,1\}^{N\times N}$ to capture the graph's topological structure, where $A_{ij} = 1$ if $(i,j) \in E$ and $A_{ij} = 0$ otherwise.

To account for self-loops, we modify the adjacency matrix to $\tilde{A} = A + I$, where I denotes the identity matrix. The features of each node are now represented by a matrix $F \in \mathbb{R}^{n\times k}$, where k indicates the dimension of the feature space.

Additionally, we utilize the diagonal degree matrix D for the graph G, where $D_{ii} = d_i$ denotes the degree of node i, calculated by $d_i = \sum_j A_{ij}$. The normalized transition matrix P is then defined as $P = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$.

8.4.1 Spectral Clustering

Spectral clustering is a robust method that utilizes the spectral properties of graph Laplacians to achieve clustering [128]. This section covers the essential matrices and concepts that are fundamental to spectral clustering techniques.

A key component in spectral clustering is the graph Laplacian, which comes in two primary forms. The unnormalized graph Laplacian is given by $L = D - W$, where W

represents the weighted adjacency matrix and D is the diagonal degree matrix, with $D_{ii} = \sum_j w_{ij}$.

Furthermore, the normalized graph Laplacian can be represented by the following formula: $\mathcal{L} = D^{-1/2}LD^{-1/2} = I - D^{-1/2}WD^{-1/2}$. Here, $\mathcal{L}$ provides a normalized version of the Laplacian that adjusts for the degree of nodes, facilitating more effective clustering.

These Laplacians have several important properties [32]:

1. They are symmetric and positive semi-definite.
2. The smallest eigenvalue is 0, with corresponding eigenvector $\mathbf{1}$ for L and $D^{1/2}\mathbf{1}$ for $\mathcal{L}$.
3. They have n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$.

For any vector $f \in \mathbb{R}^n$, we have:

$$f^T Lf = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2; \quad f^T \mathcal{L} f = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}\left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}}\right)^2 \tag{8.1}$$

The multiplicity k of the eigenvalue 0 equals the number of connected components in the graph. The eigenspace of 0 is spanned by the indicator vectors of these components for L, and by $D^{1/2}$-scaled indicator vectors for $\mathcal{L}$ [128].

Spectral clustering is closely related to the Normalized Cut (NCut) problem [114]. Given a partition of V into k disjoint subsets $A_1, \ldots, A_k$, the NCut is defined as:

$$\text{NCut}(A_1, \ldots, A_k) = \sum_{i=1}^{k} \frac{\text{cut}(A_i, V \setminus A_i)}{\text{vol}(A_i)} \tag{8.2}$$

where $\text{cut}(A, B) = \sum_{i \in A, j \in B} w_{ij}$ and $\text{vol}(A) = \sum_{i \in A} d_i$.

Minimizing NCut is NP-hard, but it can be relaxed to a tractable eigenvalue problem. This relaxation leads to the spectral clustering algorithm, which computes the first k eigenvectors $u_1, \ldots, u_k$ corresponding to the k smallest eigenvalues of $\mathcal{L}$ (or generalized eigenvectors of $Lu = \lambda Du$) [128, 114].

These eigenvectors form a matrix $U \in \mathbb{R}^{n \times k}$, where each row represents a node's k-dimensional embedding. This embedding enhances cluster properties in the data [128],

allowing for easier separation in the new representation.

The final step involves clustering these embeddings, typically using the k-means algorithm, to obtain the approximate solution to the NCut problem. This approach effectively captures important graph properties such as communities and structural characteristics through the spectrum of the Laplacian, providing a powerful tool for graph partitioning [32, 128, 114].

8.4.2 Graph Neural Networks

GNNs have emerged as a significant technique for handling data that is structured as graphs. These models adapt the concepts of convolutional neural networks to the non-Euclidean nature of graph data. The core idea behind GNNs is to iteratively update node representations by collecting and processing information from neighboring nodes. A typical GNN layer can be formulated as:

$$H^{(i+1)} = \sigma(AH^{(i)}W^{(i)}), \tag{8.3}$$

where $H^{(i+1)}$ denotes the updated node features matrix at layer $i+1$ after applying the layer transformation, $H^{(i)}$ represents the node features matrix before the transformation, $W^{(i)}$ is a matrix of learnable parameters and σ is a non-linear activation function. In the literature [72], it is common to use the normalized adjacency matrix, which is denoted as $\bar{A} = D^{-\frac{1}{2}}\hat{A}D^{-\frac{1}{2}}$.

8.5 Methodology

Our methodology addresses the limitations of existing GNN approaches by combining spectral graph theory with flexible multi-hop neighborhood aggregation. Figure 8.1 illustrates the difference between traditional hops and our community-based approach, which mitigates oversmoothing by emphasizing communal connections [24].

The foundation of our approach leverages spectral graph clustering to uncover global community structure. We begin with the eigendecomposition of the normalized Laplacian

$\mathcal{L}$ [32]:

$$\mathcal{L} = U\Lambda U^T, \tag{8.4}$$

where U is the matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues. The spectral properties of $\mathcal{L}$ are intimately connected to the graph's structure, with eigenvalues in the interval $[0, 2]$ [32].

We focus on the spectral gap, defined as $\gamma = \lambda_2 - \lambda_1$, where $\lambda_1 = 0$ and λ_2 is the smallest non-zero eigenvalue. This gap is related to the graph's connectivity and mixing time [121]. Specifically, the Cheeger constant $h(G)$, which measures the "bottleneckedness" of the graph, is bounded by the spectral gap through the Cheeger inequality:

$$\frac{\lambda_2}{2} \leq h(G) \leq \sqrt{2\lambda_2}, \tag{8.5}$$

This relationship, known as the Lovász bound [128, 7], provides crucial insights into the graph's community structure. A small Cheeger constant indicates the presence of well-defined communities, while a large constant suggests a more uniformly connected graph [35].

We select the k leading eigenvectors corresponding to the smallest non-zero eigenvalues, where k is a hyperparameter. The choice of k can be guided by examining subsequent spectral gaps $(\lambda_{i+1} - \lambda_i)$, with a large gap suggesting a natural number of clusters [128]. To identify communities, we apply k-means clustering to the rows of the truncated eigenvector matrix U_k. This spectral embedding tends to separate nodes into more linearly distinguishable clusters than in the original graph space.

We then introduce an edge-pruning mechanism to emphasize intra-community connections:

$$\mathcal{A}_{ij} = A_{ij} \cdot [C(i) = C(j)], \tag{8.6}$$

where $C(i)$ denotes the cluster assignment of node i. This pruning creates a block-diagonal structure in $\mathcal{A}$, aligning with the theoretical expectation of an ideal community structure in the spectral clustering framework. Following MixHop [3] and FSGNN [86], we

will use the transition matrix to perform hops but this time with $\mathcal{A}$. Now the transition matrix can be defined as $\mathcal{P} = D^{-\frac{1}{2}}\mathcal{A}D^{-\frac{1}{2}}$

Building on this community-aware structure, we incorporate a multi-hop aggregation scheme with attention-like learnable parameters, inspired by recent GNN advancements [3, 86]. The feature update rule for the $(i+1)$-th layer is:

$$H^{(i+1)} = \left[\alpha_1 A H^{(i)} W_1^{(i)} \| \alpha_2 \mathcal{P}^1 H^{(i)} W_2^{(i)} \| \alpha_3 \mathcal{P}^2 H^{(i)} W_3^{(i)} \| \cdots \| \alpha_{j+1} \mathcal{P}^j H^{(i)} W_{j+1}^{(i)}\right], \quad (8.7)$$

where j is the number of hops, $H^{(i)} \in \mathbb{R}^{n \times d_i}$ is the node feature matrix at the i-th layer, with n nodes and d_i features. $\mathcal{P} \in \mathbb{R}^{n \times n}$ is our community-aware transition matrix, and $W_{j+1}^{(i)} \in \mathbb{R}^{d_i \times d_{out}}$ are learnable weight matrices for each hop distance j at layer i, and $\|$ denotes column-wise concatenation.

We introduce learnable attention-like parameters α_{j+1} for each hop embedding and the original node features and adjacency, allowing the model to weigh the importance of different neighborhood scales adaptively. Importantly, these attention parameters are constrained to sum to 1: $\sum_{k=1}^{j+1} \alpha_k = 1, \quad \alpha_k \geq 0 \quad \forall k \in \{0, 1, \ldots, m\}$:

This constraint ensures that the attention mechanism is a proper weighting system across different hop distances.

This multi-hop aggregation allows the model to simultaneously capture and weigh information from various neighborhood scales, as we illustrate in Figure 8.2. For instance, if $j = 2$, the model considers the initial adjacency ($\alpha_1 A H^{(i)} W_1^{(i)}$), its immediate neighbors ($\alpha_2 \mathcal{P}^1 H^{(i)} W_1^{(i)}$), and its 2-hop neighbors ($\alpha_3 \mathcal{P}^2 H^{(i)} W_2^{(i)}$) in each layer The use of different weight matrices $W_{j+1}^{(i)}$ and attention parameters α_{j+1} for each hop distance and the initial adjacency, enables the model to learn the relative importance of information from different scales while maintaining a balanced aggregation.

This approach generalizes the power iteration method often used in spectral clustering, allowing the capture of higher-order relationships in the graph while maintaining the ability to differentiate between local and global structural information. By learning to assign different importance to various neighborhood scales and preserving the original feature information, our model effectively captures complex patterns of node similarity

and dissimilarity, adapting to both homophilic and heterophilic graph structures.

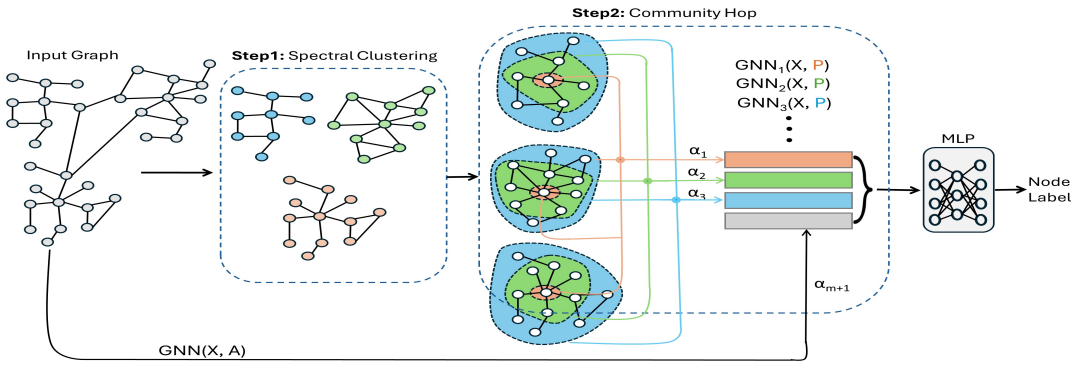


Fig. 8.2: *Illustration of the spectral clustering and GNN propagation process. The input graph undergoes spectral clustering to identify communities (Step 1). Then, a community-aware multi-hop aggregation is performed (Step 2), where information is propagated within communities. The obtained node representations are concatenated and then passed through a MLP for the final prediction of node labels.*

8.5.1 Computational Complexity

Our approach's computational complexity is divided into two primary processes: preprocessing and processing.

The preprocessing phase involves calculating the k leading eigenvectors of the normalized Laplacian matrix. For a graph with n nodes and m edges, this computation has a worst-case time complexity of $O(n^3)$ and a space complexity of $O(n^2)$. However, performance can be enhanced by employing optimized algorithms tailored for sparse graphs.

During the processing phase, k-means clustering and GNN propagation are performed. This step has a time complexity is $O(nk^2 + LHmF)$, where k denotes the number of clusters and the dimension of spectral embedding, L is the number of GNN layers, H represents the number of hops, and F is the number of features. The space complexity for the processing phase is $O(n(k + F) + m)$.

Despite the preprocessing step being computationally intensive, especially for larger graphs, it provides a comprehensive basis for identifying global community structures. This trade-off between computational cost and structural insight allows our approach to effectively capture both global and local patterns in the graph, enabling robust performance on both homophilic and heterophilic graph structures.

Table 8.1: *Node-classification accuracies. Top three models are highlighted:* ***First***, ***Second***, ***Third***.

	Texas	**Wisconsin**	**Cornell**	**Citeseer**	**Pubmed**	**Cora**
HOM LEVEL	0.11	0.21	0.30	0.74	0.80	0.81
# NODES	183	251	183	3,327	19,717	2,708
# EDGES	295	466	280	4,676	44,324	5,278
# CLASSES	5	5	5	7	3	6
MLP	80.81 ± 4.75	85.29 ± 6.40	81.89 ± 6.40	74.02 ± 1.90	75.69 ± 2.00	87.16 ± 0.37
GCN [72]	55.14 ± 5.16	51.76 ± 3.06	60.54 ± 5.30	76.50 ± 1.36	88.42 ± 0.50	86.98 ± 1.27
GAT [125]	52.16 ± 6.63	49.41 ± 4.09	61.89 ± 5.05	76.55 ± 1.23	87.30 ± 1.10	86.33 ± 0.48
GRAPHSAGE [56]	82.43 ± 6.14	81.18 ± 5.56	75.95 ± 5.01	76.04 ± 1.30	88.45 ± 0.50	86.90 ± 1.04
H2GCN [156]	84.86 ± 7.23	87.65 ± 4.89	82.70 ± 5.28	77.11 ± 1.57	89.49 ± 0.38	87.87 ± 1.20
GEOM-GCN [99]	66.76 ± 2.72	64.51 ± 3.66	60.54 ± 3.67	78.02 ± 1.15	89.95 ± 0.47	85.35 ± 1.57
LINKX [83]	74.60 ± 8.37	75.49 ± 5.72	77.84 ± 5.81	73.19 ± 0.99	87.86 ± 0.77	84.64 ± 1.13
GGCN [142]	84.86 ± 4.55	86.86 ± 3.29	85.68 ± 6.63	77.14 ± 1.45	89.15 ± 0.37	87.95 ± 1.05
CGNN [141]	71.35 ± 4.05	74.31 ± 7.26	66.22 ± 7.69	76.91 ± 1.81	87.70 ± 0.49	87.10 ± 1.35
MIXHOP [3]	77.84 ± 7.73	75.88 ± 4.90	73.51 ± 6.34	76.26 ± 1.33	85.31 ± 0.61	87.61 ± 0.85
FSGNN [86]	87.30 ± 5.29	87.84 ± 3.37	85.13 ± 6.07	77.40 ± 1.90	77.40 ± 1.93	87.93 ± 1.00
GPRGNN [31]	78.38 ± 4.36	82.94 ± 4.21	80.27 ± 8.11	77.13 ± 1.67	87.54 ± 0.38	87.95 ± 1.18
Community-HOP	89.46 ± 5.72	89.01 ± 3.84	82.70 ± 3.00	78.30 ± 2.13	89.50 ± 0.47	88.22 ± 1.29

8.6 Experiments and discusions

This section evaluates the proposed method's performance on six node classification benchmarks. The experimental results reveal that the Community-HOP method achieved the highest performance on four out of six datasets compared to baselines. This section describes the datasets and experimental settings, followed by a comprehensive comparison of the results and a detailed analysis.

To evaluate the efficacy of the Community-HOP, we selected six small to medium real-world node classification benchmark datasets: Cora, Cornell, PubMed, Texas, and Wisconsin [99]. A statistical summary of these datasets, including the edge homophily ratio (HOM LEVEL) [156], offers insight into the dataset's heterophily. A higher HOM LEVEL indicates greater heterophily, posing a challenge for vanilla GNN models, which typically perform worse under these conditions.

For the node classification experiments, we utilized the dataset splits provided by [99]. Each split includes 48% of the data for training, 32% for validation, and 20% for testing.

The performance metrics are reported as the average accuracy with standard deviation across 10 different splits. All models were trained for a total of 3000 epochs using the Adam optimizer and cross-entropy loss function. To optimize the Community-HOP method, hyperparameter tuning was carried out, focusing on parameters such as learning rate, dropout rate, number of clusters, number of hops, and hidden dimensions. A grid search strategy was used to examine different combinations of these hyperparameters. Detailed information on the hyperparameter settings for each dataset is available at the following link.

Our experimental findings show that Community-HOP markedly exceeds the performance of existing methods in accuracy across four diverse datasets, showcasing its effectiveness and adaptability in node classification tasks with varying degrees of heterophily. The analysis of edge homophily ratios (HOM LEVEL) emphasizes the difficulties encountered with higher heterophily levels and highlights the improvements offered by Community-HOP over conventional GNNs and other state-of-the-art techniques.

However, our method does not achieve superior performance on the Cornell and PubMed datasets. In the case of Cornell, this limitation can be attributed to difficulties in accurately computing spectral clusters, exacerbated by significant gaps in the dataset's homophily structure. For PubMed, the high volume of nodes and edges impedes our ability to effectively capture the underlying community structure. Consequently, inadequate clustering results in suboptimal performance for community-specific hops.

Our method focuses on community nodes by executing multiple hops within the community and shows promising results, particularly in homophilic environments where neighbors share the same label. This characteristic is advantageous as it aligns with the assumption that nodes within such environments exhibit high intra-community homophily. Notably, our Community-HOP also demonstrates effective performance in heterophilic contexts, suggesting that it can adeptly manage heterophily within communities. This adaptability contributes to its overall improved classification performance across various datasets, showing its potential for broader applicability in diverse graph-based tasks.

8.7 Conclusion and Future Work

In this chapter, we introduced a Community-HOP-based GNN model designed to address the challenge of heterophily in graphs. Central to our approach is the Community-Hop method, which leverages community structural information to refine the feature aggregation process within the GNN layers. This technique enhances the relevance of information flow both within and across communities by integrating spectral community detection with an adapted transition matrix for inter-community hops. However, a significant limitation of our approach is its computational cost, particularly for large-scale graphs and the determination of an optimal parameter k. Future work will focus on addressing these challenges by advancing spectral clustering techniques and developing methods for automatic optimization of k.

Chapter 9

Spectral Rewiring: Local-to-Global Adaptations for GNNs

"Understanding the topology of a graph allows us to reveal hidden structures and dynamics that govern complex networks."

— Philip M. Anderson

9.1 Preamble

In this chapter, we introduced a novel graph rewiring method to improve communication within graphs and generate a new optimized graph. Specifically, our method learns eigenfunctions that are reactive to graph labels and adds a linear number of edges locally to encourage community structures and globally to facilitate long-range connections. Additionally, we utilize the new optimized graph samples as augmented graphs to increase the training size of the dataset, thereby improving the generalization and robustness of the GNN. The research and findings discussed in this chapter are based on the following paper:

- **Waqar Ali**, Ahmed Begga, Francisco Escolano, Sebastiano Vascon, Thilo Stadelmann, and Marcello Pelillo. "Inductive Spectral Theory: Learnable Local-to-Global Spectral Rewiring in GNNs"; Submitted to the 39th Annual AAAI Conference on

Artificial Intelligence 2024 (under review).

The author has the following contributions:

- **Developing** the pipeline of the algorithm.
- **Writing** a considerable code.
- **Performing** most of the experiments.
- **Writing** a significant part of the paper.

9.2 Introduction

GNNs [53, 111, 23] have emerged as powerful tools for analyzing graph-structured data, driving significant advancements in social network analysis, molecular biology, and recommendation systems [154, 137]. Most GNN architectures such as GCN [72], GAT [125] and others [56, 138] operate through message passing, where node features are iteratively updated by aggregating information from neighboring nodes and generate a new representation (node embeddings) for nodes [51]. Further, this node embedding output can perform various tasks like graph and node classification.

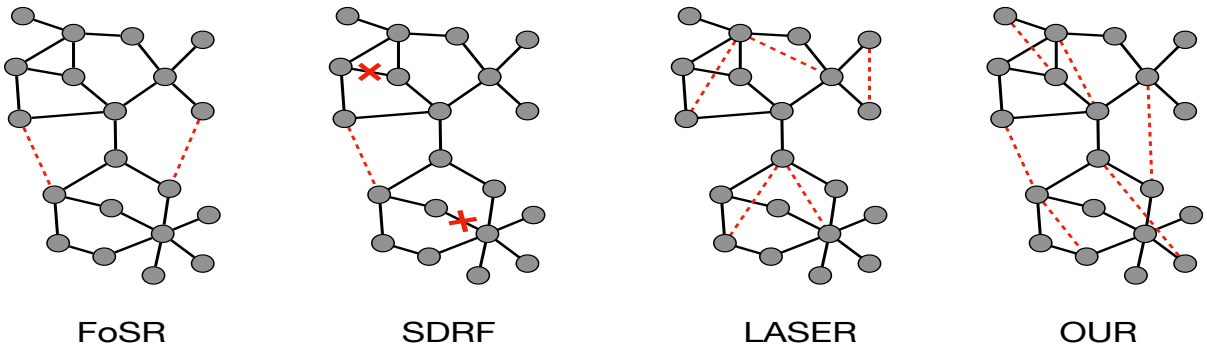


Fig. 9.1: *An analysis of various graph rewiring techniques, including FoSR, SDRF, LASER, and IST(our), for mitigating bottlenecks in the input graph.*

However, the GNN's message passing mechanism faces significant challenges, particularly in practical applications that require capturing long-range interactions. One prominent issue is over-smoothing, where node features become indistinguishable as the number of

layers increases [20]. This convergence of features limits the depth of GNNs, thereby restricting their ability to capture complex relationships within the data. Another critical issue is over-squashing [6], and it occurs when information from an exponentially growing receptive field must be compressed into fixed-size node representations, potentially losing important long-range interactions. Over-squashing is closely related to topological properties of the input graph, such as curvature and effective resistance [7, 66, 19, 14].

One prevalent strategy to address these issues is graph rewiring, which aims to modify the connectivity of the input graph to improve information flow and alleviate over-squashing. These methods can be broadly categorized into spatial and spectral approaches. Spatial rewiring often focuses on connecting nodes within a certain hop distance, including LASER and hopGNN [49, 46, 14] while spectral rewiring optimizes graph-theoretic properties related to connectivity, including Diffwire and First-order Spectral Rewiring (FoSR) [7, 66, 19] (see the Figure 9.1). Each approach presents trade-offs between preserving local structure, maintaining sparsity, and enhancing overall graph connectivity.

In addition to over-squashing, GNN models often struggle with limited data to perform graph classification tasks [155]. For example, GNNs have been extensively used in predicting molecular properties, a domain where obtaining labeled molecular data is often labor-intensive and involves complex laboratory procedures. This scarcity of labeled samples hinders the GNN's ability to achieve promising prediction performance. Data augmentation methods are commonly used to mitigate this issue. These methods can involve adding new features, creating virtual nodes or edges, or generating multiple views of the same graph [106, 155, 153]. This augmentation concept aligns closely with graph rewiring methods, which improve the communication pathways within a graph by strategically adding edges to reduce bottlenecks, resulting in a new optimized graph.

This chapter proposes a novel graph rewiring method called Inductive Spectral Theory (IST) that improves graph communication by optimizing its topology. Specifically, our method learns eigenfunctions that are reactive to graph labels and adds a linear number of edges locally to encourage community structures and globally to facilitate long-range connections. Additionally, we utilize the new optimized graph samples as augmented

graphs to increase the training size of the dataset, thereby improving the generalization and robustness of the GNN. We summarize our contributions as follows:

- **Graph Rewiring for Data Augmentation:** We introduce a novel graph rewiring process to generate augmented views of the original graph, effectively increasing both the size and diversity of the training dataset.
- **Label-Reactive Eigenfunction Learning:** Our technique learns eigenfunctions that are reactive to labels, preserving both label information and structural properties of the graph.
- **Multi-scale Edge Addition:** We add edges both locally to encourage community structures and globally to facilitate long-range connections while maintaining graph sparsity to avoid over-smoothing.
- **Over-squashing Mitigation:** Our approach addresses the over-squashing problem common in graph neural networks by introducing strategic long-range connections.
- **Enhanced Model Performance:** These techniques collectively improve model robustness and generalization capabilities through more diverse and representative training samples.

9.3 Related Work

9.3.1 Graph Rewiring

Recent research has focused on understanding and mitigating over-squashing in GNN through various approaches. These methods can be broadly categorized into spectral, curvature-based, effective resistance, and locality-aware techniques.

Spectral methods, such as FoSR by [66] aim to improve graph connectivity by maximizing the increase in spectral gap. FoSR adds edges strategically while preserving the original graph structure using a relational GNN architecture. Similar spectral approaches include

the work of [13], who proposed flipping edges based on effective resistance to increase the spectral gap, and [7], who developed a method to reweight edges leveraging the Lovász bound.

Curvature-based approaches leverage the geometric properties of graphs. [93] introduced Batch Ollivier-Ricci Flow (BORF), which uses Ollivier-Ricci curvature to address over-smoothing and over-squashing simultaneously. Their rewiring algorithm modifies local graph geometry to improve information flow. This builds upon earlier work by [121] who used Forman curvature to analyze over-squashing and proposed a rewiring technique based on increasing edge curvature.

Effective resistance methods, exemplified by [19] utilize total effective resistance as a measure of over-squashing. Their approach adds edges to minimize total effective resistance, thereby improving connectivity between all node pairs. This concept is related to the work of [124], who proposed incorporating effective resistance-based features into GNNs to capture graph topology information.

Locality-aware methods, such as Locality-Aware SEquential Rewiring (LASER) by [14], attempt to balance local and global graph properties. LASER uses a sequence of rewiring operations considering connectivity measures and locality constraints, aiming to preserve graph sparsity and local structure while reducing over-squashing. This approach shares similarities with multi-hop aggregation methods proposed by [3] and [130], which also attempt to capture local and global graph information. These diverse approaches to

Table 9.1: *Properties of different types of rewirings.*

Method	Differentiable	Preserve locality
FoSR	✗	✗
GTR	✗	✗
Diffwire	✔	✗
Ours (IST)	✔	✔

graph rewiring offer various strategies for mitigating over-squashing: FoSR adds edges based on spectral properties, BORF modifies edge weights to increase curvature, effective resistance methods add edges to minimize total resistance, and LASER uses a sequential

process balancing local and global connectivity improvements. Each method provides unique insights into addressing the over-squashing problem while attempting to preserve important graph properties (see Table 9.1).

9.3.2 Graph Augmentation

Data augmentation methods aim to improve the generalization and robustness of deep neural networks, particularly in fields such as CV [115] and NLP [148]. In CV, methods like image flipping, noise injection, and Cutout have been widely adopted to generate more varied training datasets. Similarly, generative models like Variational Auto-Encoders (VAEs) [71] and Generative Adversarial Networks (GANs) [52] can produce new samples by learning the underlying data distribution. However, applying data augmentation techniques to graph-structured data presents unique challenges due to the non-Euclidean nature of graphs, where nodes are irregularly connected by edges [153, 41].

Recent works have focused on developing graph augmentation by revising the graph's structures and manipulating node features for node-level and graph-level prediction tasks [126, 73]. Feature-based augmentation methods manipulate the node features to create new training samples. Researchers have recently developed Mixup augmentation methods [126, 59] for graph augmentation, which generates augmented graphs by interpolating the features of node pairs or through adversarial learning. Nevertheless, the most commonly used graph augmentation methods are based on randomly modifying graph structures, where edges and nodes are randomly added or removed [106, 144, 155]. Such random transformations may destroy the original topological structural characteristics of the graph and alter label-related information, potentially reducing the effectiveness of these augmentations for improving graph classification model performance [106].

Our approach of graph rewiring strategically improves the communication pathways within a graph by adding edges to reduce bottlenecks: it is both local and global. This results in a new, optimized graph structure that addresses the over-squashing issue and serves as an augmented view of the graph. By using this rewired graph as an augmented sample, we can increase the size and diversity of the training data, thereby enhancing

the model's robustness and generalization capabilities.

9.4 Spectral Graph Theory

In a graph $G = (V, E)$ with $N = |V|$ nodes and edges $|E|$, with $E \subseteq V \times V$ the adjacency matrix $\mathbf{A} \in \{0,1\}^{N \times N}$ is a square matrix where $\mathbf{A}_{ij} = 1$ if edge $(i, j) \in E$, and 0 otherwise. The degree matrix $\mathbf{D}$ is a diagonal matrix with $d_i = \mathbf{D}_{ii}$ representing the degree of node i, which is the count of edges connected to i. Then, from $\mathbf{A}$ and $\mathbf{D}$ we obtain the *graph Laplacian* $\mathbf{L} := \mathbf{D} - \mathbf{A}$. The *normalized Laplacian* $\mathcal{L}$ is given by $\mathcal{L} := \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, where $\mathbf{I}$ is the identity matrix. The eigenvalues of the Laplacian and the normalized Laplacian offer insights into various structural aspects of the graph, including connectivity, community structure, and information diffusion. Specifically, the spectrum of the normalized Laplacian $\mathcal{L}$ consists of non-negative real numbers ordered as $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \leq 2$. Given the spectrum and the corresponding eigenvectors $\mathbf{u}_i \in \mathbb{R}^N$ satisfying $\mathcal{L}\mathbf{u}_i = \lambda_i \mathbf{u}_i$, the spectral decomposition of $\mathcal{L}$ is given by $\mathcal{L} = \mathbf{U}\text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)\mathbf{U}^T = \sum_i \lambda_i \mathbf{u}_i \mathbf{u}_i^T$.

Spectral Graph Theory (SGT) [32] addresses the study of the normalized Laplacian's spectra and their eigenvectors. The most important of these vectors is $\mathbf{v}_2$ (the *Fiedler vector*) the one associated with the *spectral gap* λ_2 (which is positive if the graph is connected). The gap is a fundamental quantity in SGT (e.g. it bounds the graph connectivity and its inverse determines the mixing time of random walks). It is obtained as follows:

$$\lambda_2 = \min_{f \perp \mathbf{D}^{1/2}\mathbf{1}} \frac{\mathcal{E}(f)}{\sum_{i \in V} f_i^2 d_i} = \min_f \frac{\text{vol}G \cdot \mathcal{E}(f)}{\sum_{i,j}(f_i - f_j)^2 d_i d_j} \,. \tag{9.1}$$

where $\perp$ stands for perpendicular, $\text{vol}G = \sum_{i \in V} d_i$ is the volume of the graph and $\mathcal{E}(f) := \sum_{i \sim j}(f_i - f_j)^2$ is known as the *Dirichlet energy* of $f : V \to \mathbb{R}^N$. Actually $\mathbf{u}_2 = \mathbf{D}^{1/2} f$. Herein it is key to note that $f \perp \mathbf{D}^{1/2}\mathbf{1}$ where $\mathbf{u}_1 = \mathbf{D}^{1/2}\mathbf{1}$.

One key concern in this chapter is *spectral clustering*. It is well known [114][128] that

$$\frac{\mathcal{E}(f)}{\sum_{i \in V} f_i^2 d_i} \leq \text{Ncut}(A, B) := \frac{\text{cut}(A, B)}{\text{vol}A} + \frac{\text{cut}(A, B)}{\text{vol}A} \,, \tag{9.2}$$

where $V = A \cup B$, $A \cap B = \emptyset$ is the optimal partition in terms of minimizing the normalized cut Ncut(A,B), which is an NP-Hard problem. In general, if we pack the K smallest eigenvectors of $\mathcal{L}$ in a $N \times K$ matrix $\mathbf{U}$, feeding a K-means clustering with the rows of this matrix leads to partitioning the graph into K communities $C_1, C_2, \ldots, C_K$. Interestingly, the squared distances between two rows are bounded as follows [56]:

$$\left\| \frac{\mathbf{U}_{i,1:K}}{\sqrt{d_i}} - \frac{\mathbf{U}_{j,1:K}}{\sqrt{d_j}} \right\|^2 \leq \max_M NK \cdot \mathrm{NCut}(C_M, C_{L \neq M}) \ . \tag{9.3}$$

Therefore, small distances between the rows in $\mathbf{U}$ are usually associated with nodes in the same cluster and larger distances correspond to inter-cluster nodes.

9.5 Methodology

Despite the usefulness of SGT for providing a wide catalog of topologically meaningful distances (both local and global) to rewire a graph, the computational cost of computing the eigenvectors is $O(N^3)$. This is not feasible for large graphs. In addition, in some tasks such as graph classification (see below), where several training graphs per class are provided, SPG is limited. It cannot capture the typical eigenvectors of each class or find a consensus eigenspace for all classes.

9.5.1 Inductive Spectral Theory

IST studies the expressiveness of the spectral elements of $\mathcal{L}$ (eigenfunctions, gaps and distances) derived from

$$\min_{f \perp \mathcal{P}} \frac{\mathcal{E}(f)}{\sum_{i \in V} f_i(\mathbf{A})^2 d_i} + \mathcal{L}_{task} \ . \tag{9.4}$$

Firstly, $\mathcal{E}(f) := \sum_{i \sim j} [f_i(\mathbf{A}) - f_j(\mathbf{A})]^2$ is a Dirichet energy where f_i and f_j are scalars, the components of a learnable mapping $f : \mathbf{A} \to \mathbb{R}^N$. The purpose of f is to leverage high-order (HO) similarities (common neighbors, see below) between $\mathbf{a}_{:i}$ and $\mathbf{a}_{j:}$, the columns in $\mathbf{A}$ of the nodes in V linked by the edges $(i, j) \in E$. Then, the eigenvectors f which are the natural minimizers of the Dirichlet energies incorporate these similarities in a catalog

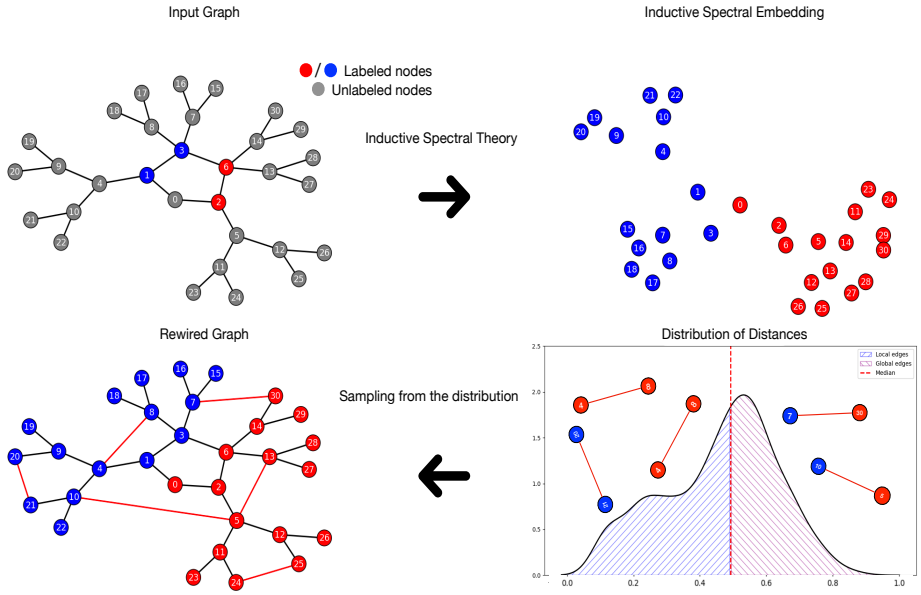


Fig. 9.2: *Visualization of the IST process for graph rewiring. The figure illustrates the transformation from an input graph to a rewired graph through IST. It shows how labeled and unlabeled nodes are mapped to an inductive spectral embedding, resulting in a distribution of distances. The rewired graph is then created by sampling from this distribution, adding both local and global edges based on the learned spectral properties. When applied to node classification, this input graph induces over-squashing but this is avoided by clustering the node embeddings. Over-smoothing is reduced by increasing $\mathcal{E}$ guided by $\mathcal{L}_{task}$.*

of orthogonal functions with respect to an eigenspace $\mathcal{P}$.

On the other hand, $\mathcal{L}_{task}$ is the task-dependent classification loss. Node classification, graph classification, and link prediction are downstream tasks where IST may leverage partially-observed labels to find data-centered eigenvalues and eigenvectors.

IST is rooted in structural semi-supervised learning [119], but herein we incorporate the recent trend in large graph mining where $f(\mathbf{A})$ is an MLP [83]. Making this MLP reactive to the task loss $\mathcal{L}_{task}$, i.e. minimizing $\mathcal{E}(f(\mathbf{A})) + \mathcal{L}_{task}$, we transfer the training labels to the learning of eigenvectors.

Common Neighbors. IST exploits the following observation. Given $f(\mathbf{A}) \in \mathbb{R}^{N\times K}$, with $f(\mathbf{A}) = \sigma(\mathbf{WA})$, and the learnable weight matrix $\mathbf{W} \in \mathbb{R}^{K\times N}$, the expansion $(\mathbf{WA})_{ip} = \sum_{p\in N(i)} \mathbf{W}_{ip}$ means that if a node i has many neighbors p of a given community, then they i and p belong to the same community and $\mathbf{W}_{ip}$ will be large on average. This is consistent with the *friendship paradox* (my friends have more friends than me). Therefore, for the general model $f(\mathbf{A}) = \text{MLP}_\theta(\mathbf{A})$, the extension of Eq. 9.4 for computing all the

empirical eigenfunctions (EE) is

$$\min_{\theta} \text{Trace}[f(\mathbf{A})^T \mathcal{L} f(\mathbf{A})] + \mathcal{L}_{task} \text{ s.t. } f(\mathbf{A}) f(\mathbf{A})^T = \mathbf{I} \,. \tag{9.5}$$

We solve such a problem via SGD. Denoting a generic column of $f(\mathbf{A})$, such as the Fiedler vector f, we have characterized its structure in terms of the weights of the MLP. For a single-layer MLP we prove that such a structure *is dominated by the number of common neighbors* (**Theorem 1** and its corollaries in the Appendix A). To give here an intuition about this fact, note that $f_i = \sigma(\mathbf{W}_{i,:}\mathbf{A})$ with $\sigma = \tanh$ for providing bipolar outputs. Then the Dirichlet energy $\sum_{i\sim j}(g_i - g_j)^2$ of the respective logits $g_i := \mathbf{W}_{i:}\mathbf{A}$, $g_j := \mathbf{W}_{j:}\mathbf{A}$ is expanded as follows:

$$\sum_{i\sim j}(g_i - g_j)^2 = \sum_{i\sim j}[\sum_{p\in N(i)} \mathbf{W}_{ip} - \sum_{q\in N(j)} \mathbf{W}_{jq}]^2 \,, \tag{9.6}$$

where the weights corresponding to the common neighbors $r \in N(p) \cap N(q)$ are included (if they do exist). Note that now we are comparing neighborhoods and their weights instead of scalars as in $\sum_{i\sim j}(f_i - f_j)^2$ which is combinatorially richer. The role of common neighbors allows us to study the particularities of trees vs graphs (with cycles).

Transductive/Inductive Power. Given that we learn a non-linear mapping $MLP_\theta(\mathbf{A})$, we can perform both transductive and inductive learning. For instance, when the task is node classification we can either predict the labels of test nodes or analyze the robustness of the model under structural attacks. Link prediction is more inductive and common-neighbors heuristics usually drives it. Finally, graph classification has been usually addressed via transductive methods, but in this chapter, we show how to provide out-of-the-sample graphs via structural data augmentation.

Overall, the number of labeled samples needed to achieve a good generalization performance *depends on the degree distribution*. We cover this issue in **Theorem 2** and its corollaries in the Appendix A. Again, to give an intuition, note that the denominator of

Eq. 9.4 as per the logits g_i can be expanded as follows:

$$\sum_{i\in V} g_i^2 d_i = \sum_{i\in V} [\sum_{p\in N(i)} \mathbf{W}_{ip}]^2 d_i = \sum_{i\in V} \mathbf{W}_{ip}^2 \sum_{p\in N(i)} d_p \ . \tag{9.7}$$

Since the denominator is maximized, the magnitude of the weights increases proportionally to $\sum_{p\in N(i)} d_p$ instead of d_i as in Eq. 9.4. This results in more separable weights thus avoiding close-to-zero entries in the Fiedler vector whenever d_i is large enough. In general, large degrees lead to a small number of labeled samples. In the Appendix, we will also provide extensive experiments with different types of graphs (trees, SBMs, cycles, etc).

9.5.2 Method: Graph Classification

Following IST, graph classification is addressed as follows.

1) Consensus EEs. Given a set of training samples $\mathcal{T} = \{(G_i, l_i)\}$ (graphs and labels), we feed an MLP with the adjacencies $\{\mathbf{A}_i\}$ (padding ensures a common size) and labels $\{l_i\}$: $f_1(\mathbf{A}) = \text{MLP}_1(\{\mathbf{A}_i, l_i\})$ minimizes the loss Trace + $\mathcal{L}_{task}$ in Eq. 9.5 and $f_1(\mathbf{A})$ encodes a *consensus eigenspace* of K EEs. K is a hyperparameter.

2) Mapping. We train a second MLP, with $f_1(\mathbf{A})$ and the labels. Actually, we have

$$\mathbf{Z} = \text{Readout}(\text{MLP}_2(\text{MLP}_1(\{\mathbf{A}_i, l_i\}))) \ , \tag{9.8}$$

where the second MLP maps $f_1(\mathbf{A})$ with K eigenvectors to $f_2(\mathbf{A})$ with $\mathcal{C}$ (number of classes) eigenvectors. Finally, Readout is a permutation-invariant operation that combines the representations of the nodes (rows of $f_2(\mathbf{A})$).

3) Nodal distances. Now, we freeze the weights of MLP_1 and we feed it with the training adjacencies $\{\mathbf{A}_i\}$. Each of the predicted eigenspaces $\hat{f}(\mathbf{A}_i)$ provides a distribution of pairwise distances $\mathcal{D}_i$ between the rows of the predicted eigenspace associated with the nodes of $G_i = (V_i, E_i)$.

4) Data augmentation. We augment the edges of each training graph $G_i = (V_i, E_i)$ by sampling $\mathcal{D}_i$ for adding $N/2$ *local* edges, and $N/2$ *global* ones, where N is the common

padding size. We add a local edge if the distance between its nodes (i, j) is smaller or equal to the median (see Figure 9.2). Otherwise, we have a global edge.

5) GNNs. We train the GNNs both with the original $\{G_i\}$ and augmented $\{\tilde{G}_i\}$ graphs. Then we perform the test and provide the accuracy. In graph classification, the label transfer is not as obvious as in node classification. Note that the colored labels in the input graph of Figure 9.2 are induced by the weights of the MLP when they react to $\mathcal{L}_{task}$.

9.5.3 Computational Efficiency

The computational complexity of IST is primarily determined by the learning of eigenfunctions and the subsequent rewiring process. For a graph with N nodes and E edges, the space complexity of our method is $O(NK)$, where K is the number of learned eigenfunctions. The time complexity for computing the Dirichlet energy and task-specific loss is $O(EK + NK^2)$, leveraging sparse matrix operations for efficiency. The edge addition step, both local and global, has a complexity of $O(N \log N)$ due to the use of efficient sampling techniques. Overall, IST's computational cost scales favorably with graph size, making it applicable to large-scale graph learning tasks. This efficiency is particularly noteworthy when compared to traditional spectral methods that often require $O(N^3)$ operations for eigendecomposition. Our approach thus offers a scalable alternative for graph rewiring and data augmentation in the context of GNNs.

Table 9.2: *Comparison of the proposed method and baselines. The bold numbers represent the highest accuracy score, and OOR is referred to as out-of-resource.*

Classification	Methods	Mutag	BZR	Mutagen	PTCMM	PROTEINS	ENZYMES	IMDB-B	COLLAB
None	GIN	76.02±0.03	79.45±0.01	79.59±0.03	62.05±0.01	69.23±0.01	30.25±0.01	67.12±0.01	71.77±0.04
Rewiring	SDRF [121]	78.10±0.02	80.20±0.01	79.75±0.03	59.08±0.01	70.31±0.01	31.30±0.02	67.10±0.01	73.20±0.04
	FOSR [66]	74.62±0.02	79.50±0.01	79.10±0.03	60.45±0.01	72.41±0.08	24.10±0.01	66.30±0.09	73.01±0.04
	GTR [19]	79.45±0.02	80.58±0.02	79.89±0.02	61.45±0.02	70.17±0.01	29.01±0.01	67.21±0.02	OOR
	DiffWire [7]	75.21±0.02	78.34±0.01	79.09±0.02	62.17±0.02	69.25±0.01	28.03±0.01	68.30±0.03	73.78±0.04
	BORF [93]	77.30±0.02	79.45±0.01	OOR	63.25±0.01	69.75±0.08	29.76±0.01	67.35±0.09	OOR
	LASER [14]	72.95±0.02	78.58±0.01	61.48±0.01	59.25±0.02	63.77±0.19	20.73±0.08	69.07±0.09	72.50±0.04
Augmentation	DropEdge [106]	77.58±0.61	79.75±0.57	78.08±0.19	62.82±0.61	**74.31±0.27**	31.83±0.61	64.90±0.47	60.90±4.47
	DropNode [144]	78.80±0.85	79.87±0.48	77.50±0.31	56.21±0.61	72.77±0.53	31.54±0.54	68.50±0.59	68.50±0.47
	M-Evolve [155]	75.59±0.94	79.30±0.51	77.84±0.18	58.75±0.71	72.31±0.38	32.35±0.61	67.40±0.67	61.50±0.71
	Gmixup [59]	78.10±0.65	80.89±0.42	78.08±0.64	62.30±0.68	65.81±2.13	30.66±4.39	68.10±1.25	73.10±0.59
Ours	**IST**	**81.20±0.02**	**81.02±0.01**	**80.69±0.03**	**66.01± 0.01**	70.57±0.08	**34.68±0.01**	**69.10±0.01**	**75.39±0.04**

Table 9.3: *Comparison of the proposed method and baselines. The bold numbers represent the highest accuracy score.*

	GCN					GIN				
	None	SDRF	FoSR	BORF	IST	None	SDRF	FoSR	BORF	IST
Cora	86.7 ± 0.3	86.3 ± 0.3	85.9 ± 0.3	87.5 ± 0.2	**88.1 ± 0.3**	76.0 ± 0.6	74.9 ± 0.1	75.1 ± 0.8	78.4 ± 0.4	**78.6 ± 0.3**
Citeseer	72.3 ± 0.3	72.6 ± 0.3	72.3 ± 0.3	73.8 ± 0.2	**74.1 ± 0.2**	59.3 ± 0.9	60.3 ± 0.8	61.7 ± 0.7	63.1 ± 0.8	**63.4 ± 0.4**
Texas	44.2 ± 1.5	43.9 ± 1.6	46.0 ± 1.6	49.4 ± 1.2	**52.4 ± 1.0**	53.5 ± 3.1	50.3 ± 3.7	47.0 ± 3.7	63.1 ± 1.7	**66.9 ± 1.3**
Cornell	41.5 ± 1.8	42.2 ± 1.6	40.2 ± 1.6	**50.8 ± 1.1**	50.1 ± 0.9	36.5 ± 2.2	40.0 ± 2.1	35.6 ± 2.4	**48.6 ± 1.2**	48.4 ± 1.8
Wisconsin	44.6 ± 1.4	46.2 ± 1.2	48.3 ± 1.3	50.3 ± 0.9	**51.1 ± 0.7**	48.5 ± 2.2	48.8 ± 1.9	48.5 ± 2.1	54.9 ± 1.2	**56.0 ± 1.1**
Chameleon	59.2 ± 0.6	59.4 ± 0.5	59.3 ± 0.6	61.5 ± 0.4	**62.0 ± 0.5**	58.1 ± 2.1	58.4 ± 2.1	56.3 ± 2.2	65.3 ± 0.8	**66.8 ± 1.3**

9.6 Experiments

This section provides an empirical evaluation of IST's effectiveness across various tasks, such as node classification and graph classification, in comparison to other rewiring techniques like curvature-based methods, spectral gap approaches, and locality-aware strategies. The code used for these experiments can be found at https://anonymous.4open.science/r/IST-24C6.

Datasets: We conduct experiments on a range of standard node and graph classification tasks, following the same methodology as BORF [93] to ensure a fair comparison. For node classification, we report our findings using datasets such as Cora, Citeseer [113], Texas, Cornell, Wisconsin [99], and Chameleon [108], comparing BORF against both the baseline of no graph rewiring and two other rewiring techniques.
For graph classification, we evaluate well-established benchmarks like PROTEINS, ENZYMES, COLLAB, MUTAG, and IMDB-BINARY [92], which are known for requiring long-range interactions as discussed in [66]. Additionally, we incorporate three more datasets—BZR, PTCMM, and MUTAGENICITY [155]—to further assess the effectiveness of our approach, particularly in scenarios involving varied dataset sizes and complexities. More detailed information about all the datasets used can be found in the Appendix.

Baselines: For graph classification, we benchmark IST against several state-of-the-art rewiring approaches. These include no graph rewiring as a baseline, SDRF [121],

which leverages discrete Ricci curvature for graph rewiring, and BORF [93], another curvature-based rewiring method. We also compare against FoSR [66], which optimizes the spectral gap of the graph, and Locality-aware LASER [14], which focuses on preserving local structure during rewiring. These comparisons aim to verify the efficiency and effectiveness of our IST method across various graph structures. To further assess the performance of IST in an augmentation setting for graph classification, we extend our evaluation to include several widely used graph augmentation techniques. These include DropEdge [106], which randomly removes a certain fraction of edges from the input graph, and DropNode [144], which randomly removes nodes and their associated edges. We also consider Mevolve [155], which generates new graphs through a graph evolution process, and Gmixup [59], which creates new graphs by interpolating between existing ones.

For node classification tasks, we specifically focus on rewiring methods that have proven effective in this context. We compare IST with a baseline without rewiring, SDRF [121], FoSR [66], and BORF [93]. These methods represent some of the few approaches that have addressed node classification through graph rewiring, making them crucial baselines for our evaluation. While many other methods exist in the state-of-the-art for node classification, we specifically concentrate on those employing rewiring techniques to maintain consistency with our approach.

Experiment setup: For our graph classification experiments, we implement each augmentation technique as a preprocessing step on all graphs within the training dataset. This process generates new graph structures that are then integrated into the training set, forming the final dataset for model training. To ensure a fair and consistent comparison across all baseline methods in graph classification, we employ two distinct architectures: GCN and GIN. We maintain consistent hyperparameters across all baselines for the GIN model, utilizing 64 hidden units, a dropout rate of 0.5, and 4 layers, aligning with established research practices.

Our training protocol for graph classification involves a learning rate of 0.001, weight de-

cay set at 0.00001, and a maximum of 1000 epochs, with early stopping implemented after 100 epochs of no improvement. We conduct 100 random trials to ensure the robustness of our results in this task.

In contrast, for node classification experiments, we adhere to the experimental setup proposed by BORF to maintain fairness in comparisons. This includes running each experiment 10 times with a data split of 60% for training, 20% for validation, and 20% for testing. Importantly, we refrain from hyperparameter tuning in node classification tasks, instead using the conditions suggested by BORF across all methods, including our proposed IST rewiring technique.

This comprehensive approach allows us to rigorously evaluate the performance of our IST rewiring method against existing techniques across different graph neural network architectures and tasks, ensuring a balanced and equitable comparison in both graph classification and node classification scenarios.

Results: Our comprehensive evaluation on graph classification tasks, as illustrated in Table 9.2, demonstrates the exceptional efficacy of our proposed method. IST consistently achieves superior accuracy compared to existing rewiring and augmentation techniques across a diverse array of datasets encompassing molecular structures, bioinformatics, and social networks. The performance gains are particularly notable, with IST yielding an average improvement of 2.0% across most datasets, with the sole exception being the proteins dataset. The success of IST can be attributed to its nuanced approach to edge addition. Unlike methods such as SDRF, which rely solely on local curvature and often lead to suboptimal modifications, or FoSR and GTR, which may add edges indiscriminately based on global connectivity, IST employs a balanced strategy. By considering local and global graph properties, IST introduces linear edges that enhance structural cohesion and optimize information flow throughout the graph.

Extending our analysis to node classification tasks, we observe similarly impressive performance from IST. Table 9.3 presents the results across various benchmark datasets, comparing IST against other prominent rewiring methods such as SDRF, FoSR, and

BORF, as well as a baseline without rewiring. IST demonstrates remarkable consistency, achieving the highest accuracy scores across all evaluated datasets for both GCN and GIN architectures. Notably, on the Cora dataset, IST achieves accuracies of 88.1% and 78.6% for GCN and GIN respectively, surpassing the next best method by significant margins. Similar trends are observed for other datasets, with IST showing particular strength on challenging datasets like Texas and Wisconsin. These results underscore IST's versatility and effectiveness in enhancing node representations, facilitating more accurate classifications even in complex network structures. The method's ability to improve performance across different graph neural network architectures further highlights its robustness and broad applicability in node classification tasks.

Ablation Study: To dissect the contributions of various components within IST, we conducted an ablation study across four representative graph classification datasets of varying sizes. We examined the impact of local edge addition (IST w/o Local), global edge addition (IST w/o Global), augmentation (IST w/o augmentation), and label information in eigenfunctions (IST w/o Label). The results, presented in Table 9.4, offer valuable insights into the method's efficacy. Our findings reveal that the optimal edge addition strategy varies depending on the dataset characteristics. For instance, local edge addition within communities proved most beneficial for Enzymes and PTCMM datasets, while global edge addition for enhanced long-range connections was superior for Mutag and IMDBB. This variability underscores the importance of IST's adaptive approach in addressing dataset-specific structural needs. Furthermore, the augmentation component of IST demonstrated significant performance enhancements, particularly on smaller datasets such as Mutag, Enzymes, and PTCMM. This observation highlights the crucial role of IST in mitigating over-squashing effects, thereby improving the overall performance of GNN models across diverse graph structures.

Table 9.4: *Ablation studies about different IST components.*

Architecture	**Mutag**	**ENZYMES**	**PTCMM**	**IMDB-B**
IST	**81.20 ±0.02**	**34.68±0.01**	**66.01 ± 0.01**	**69.10± 0.01**
IST w/o Local	80.45±0.02	33.76±0.01	64.64±0.02	68.52±0.01
IST w/o Global	80.07±0.02	34.21±0.01	65.64±0.01	68.38±0.01
IST w/o Augmen	77.39±0.02	33.03 ±0.01	63.22±0.01	69.06±0.01
IST w/o Label	80.85±0.02	34.11±0.01	64.94± 0.02	69.03±0.02

9.7 Conclusion

In this chapter, we have introduced Inductive Spectral Theory (IST) as a novel approach to address the limitations of traditional graph rewiring techniques in GNNs. By making spectral quantities and functions learnable (e.g., eigenfunctions), IST provides a data-centered framework that adapts to the specific requirements of node, edge, and graph-level tasks. Our approach mitigates common issues such as over-squashing and over-smoothing by balancing long-range connectivity and locality and enhances scalability and applicability in diverse contexts, including graph classification. Furthermore, IST offers a principled methodology for graph data augmentation, pushing the boundaries of current graph rewiring techniques. Our results demonstrate that IST advances state-of-the-art graph rewiring and establishes a robust foundation for future research in graph-based learning tasks.

Chapter 10

Conclusion and Discussion

"There is no real ending. It's just the place where you stop the story."

— Frank Herbert

10.1 Summary of Key Findings

This thesis presented numerous contributions to advance the field of graph representation learning by addressing the key challenges outlined in Chapter 1's problem statement.

In Chapter 3, we introduced a Quasi-Clique pooling method and showed that it overcomes clique pooling's limitations and outperforms several competitive baselines on graph classification tasks.

In Chapter 4, we developed a novel DSMVPool inspired by the dominant set theory. This method used edge weights to extract all possible clusters within a graph without relying on a predefined cluster parameter. We also designed an attention-based fusion-view convolution layer that integrated different sources of information, including local topology, coarser graph structure, and node features, to improve the graph representations. DSMVPool showed state-of-the-art performance in their respective category of the taxonomy.

Chapter 5 introduced a novel HGLA-Pool method to rank and select the most important cliques based on graph structural information and node features. We also developed a rule-based method to handle overlapping nodes between cliques of the same size. Addi-

tionally, a multi-attention LocalPool is developed to capture local node properties.

Chapters 6 and 7 focused on graph augmentation methods that generate weak labelled data samples to improve the generalization and robustness of the GNN model and handle over-smoothing issues. Chapter 6 addressed the over-smoothing issues in GNNs by exploring residual skip connections, Mixup augmentation methods, and attention mechanisms. This led to the design of the RAA methodology, which demonstrated superior performance on node-level prediction tasks. Specifically, in Chapter 7, we introduced the NodeDropping Augmentation method, which selectively removes nodes with lower importance based on their degree while maintaining key topological structures. Additionally, in the case of isolated nodes, we developed a structure learning method to reconnect these isolated nodes by learning attention-based relationships between nodes, resulting in state-of-the-art performance.

In Chapter 8, we improved the message passing mechanism of GNNs by incorporating valuable insights from community detection algorithms. This addressed the heterophily issue in graphs (dissimilar labels among connected nodes) and enhanced the performance on node classification tasks. Finally, in chapter 9, we proposed a novel IST based graph rewiring method to improve communication within graphs and generate a new optimized graph. Specifically, the IST method learns eigenfunctions that are reactive to graph labels and adds a linear number of edges locally to encourage community structures and globally to facilitate long-range connections. Additionally, we utilize the new optimized graph samples as augmented graphs to increase the training size of the dataset, thereby improving the generalization and robustness of the GNN.

10.2 Future Directions

The research work in this thesis covered several different topics of graph representation learning and uncovered some interesting directions for future research work.

The novel graph pooling methods proposed in this thesis have shown notable effectiveness, specifically in their ability to leverage clique structures, extract overlapping nodes between

cliques, utilize dominant sets clustering, and integrate local and global topology structures with node feature information during pooling operation. These contributions underscore these elements' critical role in enhancing the representational power of GNNs. While our experimental results provide valuable guidance on selecting suitable pooling methods, they also show an exciting direction for future research: exploring adaptive pooling strategies capable of extracting and utilizing graph structures beyond the cliques. Additionally, the possible applications of our graph pooling methods extend into areas such as protein structure analysis. By applying these methods, researchers could more accurately identify key regions within proteins, facilitating the design of drugs that bind specifically to targeted areas. Furthermore, Our Global Local Graph Pooling method, which leverages attention mechanisms within GNNs, has demonstrated success in identifying and ranking significant clique structures within graphs. This promising capability suggests a potential application in conjunction with Large Language Models (LLMs), where our pooling method could be employed to rank documents or queries more effectively [88]. By integrating our pooling approach with GNN-enhanced Retrieval-Augmented Generation (RAG-GNN) frameworks, we could significantly improve the accuracy and relevance of responses generated by LLMs, thereby enhancing their overall performance in information retrieval and query-answering tasks.

Our pipeline for augmentation using node degree and attention in GNNs has demonstrated promising results across various graph classification tasks, including those in social networks, chemistry, and biology, particularly in scenarios with limited labelled data. Our results are encouraging that further research is necessary to validate the broader applicability and effectiveness of these augmentation methods. Additionally, while our current augmentation methods have been tailored to specific datasets, future work could focus on developing adaptive or automated augmentation techniques that generalize across different domains, reducing the need for manual adjustments.

Finally, our IST-based graph rewiring method has shown significant potential to improve communication within graphs and address the over-squashing issue in GNNs. While our rewiring method has focused on specific graph types, future studies could explore it across

various graph domains, including heterogeneous and multi-modal graphs. We can extend this approach to a wide range of real-world applications, such as biomedical and genomics networks, where optimizing long-range connections and community structures can assist in identifying functional modules and disease pathways [134, 149, 80]. Additionally, its applicability can extend to transportation and infrastructure networks, optimizing routes and improving network resilience, as well as telecommunications and sensor networks, where enhanced graph communication can lead to better signal propagation and data reliability [134].

10.3 Final Remarks

This thesis set out to enhance the capabilities of graph learning by designing advanced graph representation learning approaches, such as graph pooling, community-based message passing, augmentations, and graph rewiring. Even though our contributions represent just a fraction of the extensive field of graph representation learning, we believe they reflect the evolving landscape of GNNs and underscore the importance of continual innovation in understanding complex graph-structured data.

The diverse scope of this research highlights the depth and expansive landscape of graph representation learning, and we hope it will encourage the scientific community to develop innovative artificial intelligence solutions for real-world problems.

Appendix A

Theoretical and Experimental details for Spectral Rewiring

A.1 Summary of Results

The results below explore the expressive power of the learnable weights $\mathbf{w}$. Each component of the Fiedler vector f is encoded as $f_i =< \mathbf{w}, \mathbf{a}_{:i} >= \sum_{p \in N(i)} \mathbf{w}_p$, i.e. a projection of the corresponding column in the adjacency matrix (permutation-invariant).

Theorem 1 shows that the Fiedler vector f and, consequently, the spectral gap λ_2 can be expressed in terms of common neighbors.

Corollary 1 shows that the denser the graph the closer the learnable weights $\mathbf{w}$ to the Fiedler vector provided by the standard Spectral Graph Theory (SGT). This is enabled by the large amount of common neighbors arising in dense graphs.

Corollary 2 reveals that graph cuts are relaxed in IST with respect to their counterparts in SGT. This is due to the second-order constraints (neighbor of neighbor) imposed on the weights.

Corollary 3. Notable nodes tend to have larger components in terms of their magnitude $\mathbf{w}_i^2$.

Corollary 4. Extremal nodes in paths or leaves in trees (unit degree) tend to have small $\mathbf{w}_i^2$.

Corollary 5. However, when these extremal nodes and leaves are linked to preceding nodes in the structure (path or tree) their weight magnitude becomes similar to that of the nodes in the same loop. In other words, loops smooth magnitudes.

Lemma 1 shows that the weight vector must be orthogonal to the vector of local volumes: $\mathbf{w} \perp \mathbf{d}_N$. This explains the usual condition $f \perp \mathbf{D1}$. However, the new condition has deeper implications since local volumes are typically larger than individual degrees. This also explains cut relaxation with respect SGT.

Lemma 2 adapts the Harnack equality to explain label diffusion. In other words, the Fiedler vector is a harmonic function (the value of a component is the neighboring average). This results in weights and labels being related by the inverse of the degree.

Theorem 2 Leverages Lemma 2 to show that label diffusion leads to uncertainty (components $f_i \approx 0$) as the unlabeled node is far from the labeled one in terms of shortest paths.

Corollary 6 is a "positive" version of Theorem 2: large degrees favor label propagation. Therefore, there is a trade-off between small degrees which limit uncertainty, and large degrees which favor label propagation. In other words, the degree distribution drives label propagation.

Corollary 7 leverages Lemma 2 to show the need for both positive and negative labels. In other words, in the absence of negative labels, the available information on positive labels overrides the min-cut principles of standard SGT in most of cases.

A.2 Results

Theorem A.2.1. *For a linear mapping $f_i =< \mathbf{w}, \mathbf{a}_{:i} >$, where $\mathbf{a}_{:i}$ is the $i-th$ column of $\mathbf{A}$ and $\mathbf{w} \in \mathbb{R}^N$ is a learnable vector, the IST spectral gap λ_2 is dominated by the maximal number of common neighbors.*

Proof. Following Chung[1], the spectral gap λ_2 is given by

$$\lambda_2 = \min_{f \perp \mathbf{D1}} \frac{\sum_{i \sim j} (f_i - f_j)^2}{\sum_{i \in V} f_i^2 d_i} \,, \tag{A.1}$$

where d_i denotes the degree of node i. From the fact that

$$f_i =< \mathbf{w}, \mathbf{a}_{:i} >= \sum_{p \sim i} \mathbf{w}_p \tag{A.2}$$

we obtain

$$\sum_{i \sim j} (f_i - f_j)^2 = \sum_{i \sim j} [\sum_{p \in N(i)} \mathbf{w}_p - \sum_{q \in N(j)} \mathbf{w}_q]^2 \,. \tag{A.3}$$

Then, we proceed to rewrite the denominator as

$$\sum_{i \in V} f_i^2 d_i = \sum_{i \in V} \mathbf{w}_i^2 \sum_{p \in N(i)} d_p \tag{A.4}$$

Then,

$$\lambda_2 = \min_{f \perp \mathbf{D1}} = \frac{\sum_{i \sim j} [\sum_{p \in N(i)} \mathbf{w}_p - \sum_{q \in N(j)} \mathbf{w}_q]^2}{\sum_{i \in V} \mathbf{w}_i^2 \sum_{p \in N(i)} d_p} \tag{A.5}$$

which uncovers the second-order constraints on the components $\mathbf{w}_i$ leading to the Fiedler vector f. Then, expanding the numerator, the structure of each term $i \sim j$ is given by

$$\sum_{i \sim j} (f_i - f_j)^2 = [\sum_{U(p)} \mathbf{w}_p - \sum_{U(q)} \mathbf{w}_q + (\mathbf{w}_j - \mathbf{w}_i)]^2 \tag{A.6}$$

Where $U(p) = \{p \in N(i), p \neq j, p \notin CN_{ij}\}$ and $U(q) = \{q \in N(j), q \neq i, q \notin CN_{ij}\}$, where CN_{ij} is the set of *common neighbors* of nodes i and j. As a result, the existence of common neighbors determines the structure of the IST Fiedler vector. □

[1] Fan R.K. Chung: Spectral Graph Theory, AMS, 1994.

Corollary A.2.1.1. *For the complete graph (clique) K_N, with $N > 2$, the IST Fiedler vector is coincident with that of the standard Spectral Theory and it is mirrored by the optimal weights* $\mathbf{w}$.

Proof. For K_N every node has $N-1$ neighbors and $N-2$ common neighbors for each edge $i \sim j$. As a result, in Eq. A.6 we have that $U(p) = U(q) = \emptyset$ for all edges.

$$\lambda_2 = \min_{f \perp \mathbf{D1}} = \frac{\sum_{i\sim j}(\mathbf{w}_i - \mathbf{w}_j)^2}{\sum_{i\in V} \mathbf{w}_i^2 \sum_{p\in N(i)} d_p} \,. \tag{A.7}$$

Since in K_N any node i is linked with any other $p \neq i$, we have

$$\sum_{p\in N(i)} d_p = \sum_{p\in V} d_p = \text{vol}G \,, \tag{A.8}$$

which is a constant.

Therefore, the IST Fiedler's vector and value for K_N are almost equal to those provided by the standard Spectral Theory:

$$\lambda_2 = \min_{\mathbf{w} \perp \mathbf{D1}} \frac{\sum_{i\sim j}(\mathbf{w}_i - \mathbf{w}_j)^2}{\sum_{i\in V} \mathbf{w}_i^2} \,. \tag{A.9}$$

where vol$G$ is the volume of the graph (sum of degrees). In other words, for K_N, the learnable weights $\mathbf{w}_i$ mirror the Fiedler vector (they can be interpreted in this way). □

Corollary A.2.1.2. *The Barbell graph B_{2N} of $2N$ nodes, is formed by linking two cliques of N nodes each by a unique link which is viewed as a relaxed cut in IST.*

Proof. Given the link $i' \sim \ j'$ the edge that links the two cliques. Consider $i \sim j$ and *internal edge* E_{int} in any of the two cliques if $j \neq i'$ (left clique) or $j \neq j'$ (right clique). Then $i \sim i'$ and $i \sim j'$ are called *external edges* E_{ext}.

Now, leveraging again Eq. A.6 in Theorem A.2.1 we have that for internal edges $i \sim j$ the corresponding term in the Fiedler equation is expanded as follows:

$$(f_i - f_j)^2 = (\mathbf{w}_j - \mathbf{w}_i)^2 \,. \tag{A.10}$$

However, for external edges, j' and i' are reachable from their opposite cliques. Then, defining $\Delta\mathbf{w}_i := \mathbf{w}_{i'} - \mathbf{w}_i$ we have

$$(f_i - f_{i'})^2 = [\Delta\mathbf{w}_i - \mathbf{w}_{j'}]^2 \ . \tag{A.11}$$

and similarly

$$(f_i - f_{j'})^2 = [\mathbf{w}_{j'} - \Delta\mathbf{w}_i]^2 \ . \tag{A.12}$$

Then, we expand $\sum_{i\sim j}(f_i - f_j)^2$ as follows:

$$\sum_{i\sim j\in E_{int}} (\mathbf{w}_i - \mathbf{w}_j)^2 + (\mathbf{w}_{i'} - \mathbf{w}_{j'})^2 + \sum_{i\sim j\in E_{ext}} (\Delta\mathbf{w}_i - \mathbf{w}_{j'})^2 \ . \tag{A.13}$$

Each clique has $N(N-1)/2$ edges, half internal and half external. therefore, we have $N(N-1)/2$ internal edges and $N(N-1)/2$ external. Internal edges and the linking one behave as in the standard theory. However, the term corresponding to external edges includes the reaching of i' and j' from opposite cliques. This enforces the minimization of $(\Delta\mathbf{w}_i - \mathbf{w}_{j'})^2 = (\mathbf{w}_{i'} - \mathbf{w}_i - \mathbf{w}_{j'})^2$ which makes i close to i' (for i in the left clique) and close to j' (for i in the right one).

As per the numerator of the Fiedler equation, we have that the degree of all nodes except i' and j' is $d_i = N-1$, whereas $d_{i'} = d_{j'} = N$. Then, the denominator becomes

$$\sum_{i\neq i', i\neq j'} 2(N-1)^2\mathbf{w}_i^2 + N^2\mathbf{w}_{i'}^2 + N^2\mathbf{w}_{j'}^2 \ , \tag{A.14}$$

where all the magnitudes are $O(N^2)$ and the numerator dominates the minimization. □

Corollary A.2.1.3. *For the star graph S_N with a central node i_0 linked to N outer nodes j not linked between them, then $\mathbf{w}_{i_0}^2 > \mathbf{w}_j^2 \ \forall j \neq i_0$.*

Proof. Instantiating Eq. A.5 for this graph and considering that the peripheral nodes have unit degree, we obtain

$$\lambda_2 = \min_{f\perp\mathbf{D1}} \frac{\sum_{i_0\sim j}(\mathbf{w}_j - N\mathbf{w}_{i_0})^2}{\sum_{j\neq i_0}\mathbf{w}_j^2 + N\mathbf{w}_{i_0}^2} \tag{A.15}$$

As we must maximize the denominator, the weight of the central node is larger than that of the peripheral ones. □

Corollary A.2.1.4. *Path graph P_N of N nodes. Given the sorted nodes $i_1, i_2, \ldots, i_N$, for $1 < k < N$ we define the increments $\Delta\mathbf{w}_k := (\mathbf{w}_{i_k} - \mathbf{w}_{i_{k+1}})$. Then $\mathbf{w}_{i_1} < \mathbf{w}_{i_k}, k > 1$.*

Proof. Instantiating Eq. A.5 and isolating terms, we discover that $\mathbf{w}_{i_1}^2$ must be minimal:

$$\lambda_2 = \min_{f \perp \mathbf{D1}} \frac{\mathbf{w}_{i_1}^2 + \sum_{i_k \sim i_{k+1}} (\Delta\mathbf{w}_k + \Delta\mathbf{w}_{k+1})^2 + \mathbf{w}_{i_{N-2}}^2}{\mathbf{w}_{i_1}^2 + \sum_{1<i_k<N} 2\mathbf{w}_{i_k}^2 + \mathbf{w}_{i_N}^2} \ . \tag{A.16}$$

□

Corollary A.2.1.5. *Cycle graph C_N. For $1 \leq k \leq N$ we define the increments $\Delta\mathbf{w}_k := (\mathbf{w}_{i_k} - \mathbf{w}_{(i_{k+1} \mod N)})$. Then all $\mathbf{w}_i^2$ have a similar magnitude.*

Proof. Now the last node $\mathbf{w}_{i_N}$ is linked with the first $\mathbf{w}_{i_1}$ and all the nodes have degree 2. Then

$$\lambda_2 = \min_{f \perp \mathbf{D1}} \frac{\sum_{i_k \sim i_{k+1}} (\Delta\mathbf{w}_k + \Delta\mathbf{w}_{k+1})^2}{\sum_{i \in V} 2\mathbf{w}_i^2} \ . \tag{A.17}$$

□

Lemma 1. *In IST, the condition $f \perp$ **D1** is rewritten as $\mathbf{w} \perp \mathbf{d}_N$, where $\mathbf{d}_N(i) := \sum_{p \in N(i)} d_p$ is the* local volume *of $i \in V$ excluding d_i.*

Proof. The condition $f \perp$ **D1** (Fiedler vector must be orthogonal to the degree vector) means $\sum_{i \in V} f_i d_i = 0$. Then, by rewriting the denominator in the spectral gap (see Theorem A.2.1) we have

$$\begin{aligned} \sum_{i \in V} f_i d_i &= \sum_{i \in V} [\sum_{p \in N(i)} \mathbf{w}_i] d_i \\ &= \sum_{i \in V} \mathbf{w}_i \sum_{p \in N(i)} d_p \\ &= \sum_{i \in V} \mathbf{w}_i \mathbf{d}_N(i) = 0 \ . \end{aligned} \tag{A.18}$$

Therefore, $\mathbf{w} \perp \mathbf{d}_N$ and λ_2 is rewritten as follows:

$$\lambda_2 = \min_{\mathbf{w} \perp \mathbf{d}_N} \frac{\sum_{i \sim j} [\sum_{p \in N(i)} \mathbf{w}_p - \sum_{q \in N(j)} \mathbf{w}_q]^2}{\sum_{i \in V} \mathbf{w}_i^2 \mathbf{d}_N(i)} \ . \tag{A.19}$$

As a result, the decision boundary provided by $\mathbf{w}$ must be orthogonal to the local volume not to individual degrees. □

Lemma 2. *The IST Harnack equality shows a principle for label propagation relying on weight neighborhoods.*

Proof. The Harnack equality shows that f is Harmonic, i.e. given λ_2, we have that f_i satisfies

$$\frac{1}{d_i} \sum_{j \in N(i)} (f_i - f_j) = \lambda_2 f_i \ . \tag{A.20}$$

Then, each component f_i of the Fiedler vector is defined (up to the scale given by λ_2) as the average discrepancies between its neighbors.

Working on the above equation we obtain

$$\begin{aligned} \frac{f_i}{d_i} - \frac{\sum_{j \in N(i)} f_j}{d_i} &= \lambda_2 f_i \\ \frac{f_i}{d_i} - \lambda_2 f_i &= \frac{\sum_{j \in N(i)} f_j}{d_i} \\ f_i(1 - \lambda_2) d_i &= \sum_{j \in N(i)} f_i \\ f_i &= \frac{\sum_{j \in N(i)} f_j}{(1 - \lambda_2) d_i} \end{aligned} \tag{A.21}$$

A straightforward translation to learnable weights yields

$$\sum_{p \in N(i)} \mathbf{w}_p = \frac{\sum_{p \in N(i)} \sum_{q \in N(p)} \mathbf{w}_q}{(1 - \lambda_2) d_p} \tag{A.22}$$

Now, suppose that $f_i = l_i, i \in V$ where $l_i \in \{-1, 1\}$ is a label. Then, if all the neighbors $q \in N(p)$ but i are labeled (we denote it by $l_i = 0$) we have

$$\sum_{p \in N(i)} \mathbf{w}_p = \frac{\sum_{q \in N(p), p \in N(i)} l_q}{(1 - \lambda_2) d_i} \ . \tag{A.23}$$

Therefore, each label has a fractional contribution to the weights. This is the neural version of Laplacian learning. □

Theorem A.2.2. *Data labels lead to optimal partitions, but their transductive power decays with the shortest-path (SP) distance between labeled and unlabeled nodes.*

Proof. Suppose that $f_i = l_i, i \in V$ where $l_i \in \{-1, 1\}$ is a label. Let $P = \{x_0 = i, x_1, \ldots, x_N = j\}$ be the shortest path of length L between i and j. From Lemma 2, Eq. A.21 results in

$$f_{x_1} = \frac{\sum_{p \in N(x_1)} l_p}{(1 - \lambda_2) d_{x_1}} \tag{A.24}$$

If all the nodes $k \neq i$ are unlabeled ($l_k = 0$, which indicates maximal uncertainty in the Fiedler partitioning), then we have the succession

$$\begin{aligned} f_{x_1} &= \frac{1}{(1 - \lambda_2) d_{x_1}} \cdot l_i \\ f_{x_2} &= \frac{l_i^2}{(1 - \lambda_2)^2 d_{x_2} d_{x_1}} \\ &\vdots \\ f_j &= \frac{l_i^L}{(1 - \lambda_2)^L \prod_{i=1}^{L} d_{x_i}} \end{aligned} \tag{A.25}$$

which results in $f_j \to 0$ for a moderate L even for a small degree (for instance degree 2 in a Path graph).

Translating the succession in Eq. A.25 to the weights notation, we skip x_1 and get the label of x_0 when visiting x_2 (second-order neighbor). Then we have

$$\sum_{q \in N(j)} \mathbf{w}_q = \frac{l_i^{N-1}}{(1 - \lambda_2)^{N-1} \prod_{i=2}^{N} d_{x_i}} , \tag{A.26}$$

with similar results. □

Corollary A.2.2.1. *The injection of a label l at level r may relax the decay if it is compatible with l_i (same sign) or enforce it if it is not compatible.*

Proof. The status of a label at level r can be modified by a single "informed" adjacent node:

$$f_{x_r} = \frac{l_i^r + l}{(1-\lambda_2)^r \prod_{i=1}^r d_{x_i}} \ . \tag{A.27}$$

□

Corollary A.2.2.2. *In general, we need both positive and negative labels to induce consistent partitions in the IST Fiedler vector.*

Proof. At this point, it is interesting to leverage Lemma 1 which states that the weights are orthogonal to local volumes, i.e.

$$\sum_{i \in V} \mathbf{w}_i \mathbf{d}_N(i) = 0 \tag{A.28}$$

Since, we have also $\mathbf{w} \neq \mathbf{0}$, at least one weight is negative.

However, if all our labels are positive, these negative weights come from flipping the sign of small labels of distant nodes or of close nodes with a very high degree, which results in uncertainty $f_i \approx 0$.

An exception to this rule is the star graph S_N where the largest magnitude $\mathbf{w}_i$ is assigned to the central node. In this case, a single positive label is enough. □

A.3 Practical Findings

The above results emerge from a blend of classical SGT and experimentation. Herein, we summarize our experiments when trying to set the *minimum number of labels needed to provide full accuracy* in several prototypical graphs.

Barbell graph B_{2N} links two cliques of size N with a single edge. Minimal labeling puts positive and negative edges at the extremes of the cutting edge. Local high density (large degree) in the clique's block label propagation but small SPs (unit length) make the difference.

When modifying the Barbell graph so that one community is "absorbed" by the other, we need only two more labels. Again, the unit length of SPs makes it work.

Path Graph P_N suffers from label uncertainty for large values of N. We start by labeling the extremes of the central edge in the path as $(-1, +1)$. This is a good heuristic to set a "polarized edge": evaluate how powerful it is in terms of minimizing the NCut of the induced partition.

The central polarized edge at position $O(N/2)$ bisects the graph in two halves and depending on N further labels are needed to bisect each half at positions $O(N/4)$ and $O(3N/4)$. In addition, two more labels are needed at the two extremes of the path.

Cycle Graph C_N behavior is similar to P_N with the "polarized edge" at $O(N/2)$. nodes N and 1. Adding labels at $O(N/4)$ and $O(3N/4)$ we reach an accuracy of 92.5%.

Star Graph. In S_N, where half of the peripheral nodes and the central one belong to the same class, a single label placed at the central node yields full accuracy.

Balanced Tree. A balanced tree $B_{R,T}$ with branching factor $R > 1$ and T levels has $N = R^T - 1$ nodes where R^{T-1} are leaves (with unit degree) and the remaining interior nodes have degree $R + 1$). For $R = 2$ (binary) we have adopted the following labeling strategy: the root of each of the subtrees is labeled with opposite signs., and the root of the full tree (belonging to one of the classes) is not labeled. For $T = 2$ levels, we achieve an accuracy of 89%: a single subtree including leaves is misclassified. If in addition, we label correctly the first level of the subtrees we have full accuracy. This graph is interesting because it exemplifies the over-squashing issue.

SBMs. Stochastic Block Models, with probability $p = 0.75$ of intra-cluster linkage and probability $q = 0.25$ of inter-cluster linkage. This is a hard case where we want to test the IST cut relaxation. Having $O(N/3)$ samples (half in each cluster) we only achieve an accuracy of 50%. Setting now $p = 0.80$ and $q = 0.20$ we peak an accuracy of 90% with $O(2N/3)$ labels.

A.4 Dataset Analysis and Experimental Setup

A.4.1 Dataset Statistics

We present a comprehensive overview of the datasets utilized in our experiments, encompassing both node classification and graph classification tasks. Tables A.1 and A.2 provide detailed statistics for these datasets.

Table A.1: *Statistics of node classification datasets.*

	Cornell	**Texas**	**Wisconsin**	**Cora**	**Citeseer**	**Chameleon**
#NODES	140	135	184	2485	2120	832
#EDGES	219	251	362	5069	3679	12355
#FEATURES	1703	1703	1703	1433	3703	2323
#CLASSES	5	5	5	7	6	5
DIRECTED	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE
HOMOPHILY	0.11	0.30	0.21	0.81	0.74	0.23
AVG DEGREE	1.77	1.62	2.05	3.89	2.73	15.85
DENSITY	0.009	0.008	0.008	0.014	0.008	0.007

Table A.2: *Characteristics and Statistics of eight graph classification datasets.*

Classification	**Datasets**	**#Graphs**	**Avg Nodes**	**Avg Edges**	**Classes**
Biological	PROTEINS	1,113	39.06	72.82	2
	ENZYMES	600	32.63	62.14	6
Chemical	MUTAGENICITY	4,337	30.32	30.77	2
	MUTAG	188	17.93	19.79	2
	BZR	405	35.75	38.36	2
	PTCMM	336	13.97	14.32	2
Social Networks	COLLAB	5000	74.49	2457.78	3
	IMDB-BINARY	1000	19.77	96.53	2

For node classification datasets (Table A.1), we report additional metrics such as homophily, average degree, and density. These metrics provide insights into the structural properties of the networks. Homophily indicates the tendency of nodes to connect with

others of the same class, average degree shows the typical number of connections per node, and density reflects the overall connectedness of the graph.

Graph classification datasets (Table A.2) are categorized into biological, chemical, and social network domains. We present the total number of graphs, the average number of nodes and edges per graph, and the number of classes for each dataset.

A.4.2 Experimental Environment

All experiments were conducted using the hardware specifications outlined in Table A.3. Concerning software, we have used PyTorch Geometric (PyG), NetworkX and scikit learn as main Python libraries.

Table A.3: *Hardware specifications for experimental setup.*

Component	**Specification**
CPU	AMD 7742 64-Core @ 2.25 GHz
GPU	NVIDIA A100 Tensor Core (40GB VRAM)
RAM	1024GB DDR4
Storage	2TB NVMe SSD
Operating System	Ubuntu 20.04.5 LTS

Bibliography

[1] M. A. Abbas, W. Ali, F. Smarandache, S. S. Alshamrani, M. A. Raza, A. Alshehri, and M. Ali. *Residual Attention Augmentation Graph Neural Network for Improved Node Classification*. Engineering, Technology & Applied Science Research, (ETASR), 2024. URL https://etasr.com/index.php/ETASR/article/view/6844/3504.

[2] R. Abboud, R. Dimitrov, and I. I. Ceylan. Shortest path networks for graph property prediction. In *Proceedings of the 1st Learning on Graphs Conference*, pages 1–5. PMLR, 2022. URL https://proceedings.mlr.press/v198/abboud22a.html.

[3] 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*, 2019. URL https://arxiv.org/abs/1905.00067.

[4] W. Ali, S. Vascon, T. Stadelmann, and M. Pelillo. Quasi-cliquepool: Hierarchical graph pooling for graph classification. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 544–552, 2023. URL https://dl.acm.org/doi/abs/10.1145/3555776.3578600.

[5] W. Ali, S. Vascon, T. Stadelmann, and M. Pelillo. Hierarchical glocal attention pooling for graph classification. *Pattern Recognition Letters*, 2024. URL https://www.sciencedirect.com/science/article/pii/S016786552400271X?dgcid=author.

[6] U. Alon and E. Yahav. On the bottleneck of graph neural networks and its practical

implications. *arXiv preprint arXiv:2006.05205*, 2020. URL https://arxiv.org/abs/2006.05205.

[7] A. Arnaiz-Rodríguez, A. Begga, F. Escolano, and N. Oliver. Diffwire: Inductive graph rewiring via the lov\'asz bound. *arXiv preprint arXiv:2206.07369*, 2022. URL https://arxiv.org/abs/2206.07369.

[8] P. H. Avelar, A. R. Tavares, M. Gori, and L. C. Lamb. Discrete and continuous deep residual learning over graphs. *arXiv preprint arXiv:1911.09554*, 2019. URL https://arxiv.org/abs/1911.09554.

[9] D. Bacciu, A. Conte, R. Grossi, F. Landolfi, and A. Marino. K-plex cover pooling for graph neural networks. *Data Mining and Knowledge Discovery*, 35(5):2200–2220, 2021. URL https://link.springer.com/article/10.1007/s10618-021-00779-z.

[10] D. Bacciu, A. Conte, and F. Landolfi. Graph pooling with maximum-weight k-independent sets. In *Thirty-Seventh AAAI Conference on Artificial Intelligence*, 2023. URL https://deepai.org/publication/graph-pooling-with-maximum-weight-k-independent-sets.

[11] J. Baek, M. Kang, and S. J. Hwang. Accurate learning of graph representations with graph multiset pooling. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=JHcqXGaqiGn.

[12] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang. Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the twelfth ACM international conference on web search and data mining*, pages 384–392. WSDM, 2019. URL https://dl.acm.org/doi/10.1145/3289600.3290967.

[13] P. K. Banerjee, K. Karhadkar, Y. Wang, U. Alon, and G. Montúfar. Oversquashing in gnns through the lens of information contraction and graph expansion. In *58th Annual Allerton Conference on Communication, Control, and Computing*, pages 1–8. IEEE, 2022. URL https://doi.org/10.1109/Allerton49937.2022.9929363.

[14] F. Barbero, A. Velingker, A. Saberi, M. Bronstein, and F. Di Giovanni. Locality-aware graph-rewiring in gnns. *arXiv preprint arXiv:2310.01668*, 2023. URL https://arxiv.org/abs/2310.01668.

[15] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. URL https://arxiv.org/abs/1806.01261.

[16] A. Begga, W. Ali, G. Niculescu, F. Escolano, T. Stadelmann, and M. Pelillo. Community-hop: Enhancing node classification through community preference. In *Community-Hop: Enhancing Node Classification through Community Preference*. Joint IAPR International Workshops on Statistical Techniques in Pattern, 2024. URL https://iris.unive.it/handle/10278/5072541.

[17] F. M. Bianchi, D. Grattarola, and C. Alippi. Mincut pooling in graph neural networks. *arXiv*, abs/1907.00481, 2019. URL https://arxiv.org/abs/1907.00481.

[18] F. M. Bianchi, D. Grattarola, and C. Alippi. Spectral clustering with graph neural networks for graph pooling. In *International conference on machine learning, ICML*, pages 874–883. PMLR, 2020. URL https://arxiv.org/abs/1907.00481.

[19] 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, ICML*, pages 2528–2547. PMLR, 2023. URL https://proceedings.mlr.press/v202/black23a.html.

[20] J. Bober, A. Monod, E. Saucan, and K. N. Webster. Rewiring networks for graph neural network training using discrete geometry. In *International Conference on Complex Networks and Their Applications*, pages 225–236. Springer, 2023. URL https://link.springer.com/chapter/10.1007/978-3-031-53468-3_19.

[21] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(1): i47–i56, 2005. URL https://doi.org/10.1093/bioinformatics/bti1007.

[22] 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. URL https://ieeexplore.ieee.org/document/7974879.

[23] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations ICLR*, 2014. URL http://arxiv.org/abs/1312.6203.

[24] C. Cai and Y. Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020. URL https://arxiv.org/abs/2006.13318.

[25] L. Cai, J. Li, J. Wang, and S. Ji. Line graph neural networks for link prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. URL https://ieeexplore.ieee.org/document/9431673?denied=.

[26] C. Cangea, P. Veličković, N. Jovanović, T. Kipf, and P. Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018. URL https://arxiv.org/abs/1811.01287.

[27] F. Cazals and C. Karande. A note on the problem of reporting maximal cliques. *Theoretical computer science*, 407(1-3):564–568, 2008. URL https://www.sciencedirect.com/science/article/pii/S0304397508003903.

[28] 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*, pages 3438–3445, 2020. URL https://api.semanticscholar.org/CorpusID:202539008.

[29] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and deep graph convolutional networks. In H. D. III and A. Singh, editors, *Proceedings of the*

37th International Conference on Machine Learning, volume 119 of *Proceedings of Machine Learning Research*, pages 1725–1735. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/chen20v.html.

[30] Y. Chen and Y. R. Gel. Topological pooling on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 7096–7103, 2023. URL https://dl.acm.org/doi/10.1609/aaai.v37i6.25866.

[31] E. Chien, J. Peng, P. Li, and O. Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=n6jl7fLxrP.

[32] F. R. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997. URL https://bookstore.ams.org/cbms-92.

[33] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. URL https://arxiv.org/abs/1412.3555.

[34] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems, NIPS*, 29, 2016. URL https://dl.acm.org/doi/10.5555/3157382.3157527.

[35] S. Deng, S. Ling, and T. Strohmer. Strong consistency, graph laplacians, and the stochastic block model. *Journal of Machine Learning Research*, 22(117):1–44, 2021. URL https://dl.acm.org/doi/abs/10.5555/3546258.3546375.

[36] A. DeptuCa. Application of the dependency graph method in the analysis of automatic transmission gearboxes. *Engineering, Technology & Applied Science Research*, 11(2):7033–7040, 2021. URL https://etasr.com/index.php/ETASR/article/view/4098.

[37] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. URL https://arxiv.org/abs/1708.04552.

[38] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11):1944–1957, 2007. URL https://ieeexplore.ieee.org/document/4302760.

[39] F. Diehl. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990*, 2019. URL https://arxiv.org/abs/1905.10990.

[40] F. Diehl, T. Brunner, M. T. Le, and A. Knoll. Towards graph pooling by edge contraction. In *ICML 2019 workshop on learning and reasoning with graph-structured data*, 2019. URL url={https://api.semanticscholar.org/CorpusID:208144770}.

[41] K. Ding, Z. Xu, H. Tong, and H. Liu. Data augmentation for deep graph learning: A survey. *ACM SIGKDD Explorations Newsletter*, 24(2):61–77, 2022. URL https://dl.acm.org/doi/10.1145/3575637.3575646.

[42] P. D. Dobson and A. J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003. URL https://www.sciencedirect.com/science/article/abs/pii/S0022283603006284.

[43] J. Du, S. Wang, H. Miao, and J. Zhang. Multi-channel pooling graph neural networks. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, (IJCAI)*, pages 1442–1448, 2021. URL https://doi.org/10.24963/ijcai.2021/199.

[44] A. Elhassouny and F. Smarandache. Trends in deep convolutional neural networks architectures: A review. In *International conference of computer science and renewable energies, ICCSRE*, pages 1–8. IEEE, 2019. URL https://ieeexplore.ieee.org/document/8807741.

[45] R. Elhesha and T. Kahveci. Identification of large disjoint motifs in biological networks. *BMC bioinformatics*, 17:1–18, 2016. URL https://doi.org/10.1186/s12859-016-1271-7.

[46] J. Feng, Y. Chen, F. Li, A. Sarkar, and M. Zhang. How powerful are k-hop message passing graph neural networks. In *Advances in Neural Information Processing Systems, NIPS*, 2022. URL https://dl.acm.org/doi/10.5555/3600270.3600615.

[47] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt. Scalable kernels for graphs with continuous attributes. *Advances in neural information processing systems, NIPS*, 26, 2013. URL https://dl.acm.org/doi/10.5555/2999611.2999636.

[48] M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019. URL https://arxiv.org/abs/1903.02428.

[49] R. B. Gabrielsson, M. Yurochkin, and J. Solomon. Rewiring with positional encodings for graph neural networks. *CoRR*, abs/2201.12674, 2022. URL https://arxiv.org/abs/2201.12674.

[50] H. Gao and S. Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019. URL https://api.semanticscholar.org/CorpusID:153311899.

[51] 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, 2017. URL https://dl.acm.org/doi/10.5555/3305381.3305512.

[52] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Communications of the ACM*, 63, 2020. URL https://dl.acm.org/doi/10.1145/3422622.

[53] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IEEE international joint conference on neural networks*, volume 2, pages 729–734. IEEE, 2005. URL https://ieeexplore.ieee.org/document/1555942.

[54] H. Guo, Y. Mao, and R. Zhang. Mixup as locally linear out-of-manifold regularization. In *Proceedings of the AAAI conference on artificial intelligence*, pages 3714–3722, 2019. URL https://dl.acm.org/doi/abs/10.1609/aaai.v33i01.33013714.

[55] A. Haghighian Roudsari, J. Afshar, W. Lee, and S. Lee. Patentnet: multi-label classification of patent documents using deep learning based language understanding. *Scientometrics*, 127(1):207–231, 2022. URL url={https://api.semanticscholar.org/CorpusID:245343990}.

[56] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. URL https://dl.acm.org/doi/10.5555/3294771.3294869.

[57] W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017. URL https://arxiv.org/abs/1709.05584.

[58] K. Han, Y. Wang, J. Guo, Y. Tang, and E. Wu. Vision GNN: An image is worth graph of nodes. *Advances in NIPSs*, 35:8291–8303, 2022.

[59] X. Han, Z. Jiang, N. Liu, and X. Hu. G-mixup: Graph data augmentation for graph classification. In *International Conference on Machine Learning*, pages 8230–8248. PMLR, 2022. URL https://api.semanticscholar.org/CorpusID:246863555.

[60] 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, 2016. URL https://ieeexplore.ieee.org/document/7780459?denied=.

[61] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[62] F. Hu, Y. Zhu, S. Wu, L. Wang, and T. Tan. Hierarchical graph convolutional networks for semi-supervised node classification. *arXiv preprint arXiv:1902.06667*, 2019. URL https://arxiv.org/abs/1902.06667.

[63] J. Huang, Z. Li, N. Li, S. Liu, and G. Li. Attpool: Towards hierarchical feature representation in graph convolutional networks via attention mechanism. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6480–6489, 2019. URL https://ieeexplore.ieee.org/document/9009471.

[64] J. J. Irwin, T. Sterling, M. M. Mysinger, E. S. Bolstad, and R. G. Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012. URL https://pubs.acs.org/doi/10.1021/ci3001277.

[65] M. I. K. Islam, M. Khanov, and E. Akbas. Mpool: Motif-based graph pooling. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 105–117. Springer, 2023. URL https://link.springer.com/chapter/10.1007/978-3-031-33377-4_9.

[66] G. M. Kedar Karhadkar, Pradeep Kr. Banerjee. Fosr: First-order spectral rewiring for addressing oversquashing in gnns. In *The Eleventh International Conference on Learning Representations ICLR*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=3YjQfCLdrzz.

[67] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann. Benchmark data sets for graph kernels. 2016.

[68] B. Khemani, S. Patil, K. Kotecha, and S. Tanwar. A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions. *Journal of Big Data*, 11(1):18, 2024. URL https://journalofbigdata.springeropen.com/articles/10.1186/s40537-023-00876-4.

[69] S. Khoshraftar and A. An. A survey on graph representation learning methods. *ACM Transactions on Intelligent Systems and Technology*, 15(1):1–55, 2024. URL https://dl.acm.org/doi/abs/10.1145/3633518.

[70] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR*, 2015. URL http://arxiv.org/abs/1412.6980.

[71] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. URL https://arxiv.org/abs/1312.6114.

[72] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. URL https://arxiv.org/abs/1609.02907.

[73] K. Kong, G. Li, M. Ding, Z. Wu, C. Zhu, B. Ghanem, G. Taylor, and T. Goldstein. Flag: Adversarial data augmentation for graph neural networks. *arXiv*, 2020. URL https://arxiv.org/abs/2010.09891.

[74] K. Kowsari, D. E. Brown, M. Heidarysafa, K. J. Meimandi, M. S. Gerber, and L. E. Barnes. Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pages 364–371. IEEE, 2017. URL https://ieeexplore.ieee.org/document/8260658.

[75] 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. URL https://dl.acm.org/doi/10.5555/3540261.3541915.

[76] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60, 2017. URL https://dl.acm.org/doi/10.1145/3065386.

[77] J. Lee, I. Lee, and J. Kang. Self-attention graph pooling. In *International conference on machine learning*, pages 3734–3743. PMLR, 2019. URL https://api.semanticscholar.org/CorpusID:119314157.

[78] X. Lei, H. Pan, and X. Huang. A dilated cnn model for image classification. *IEEE Access*, 7:124087–124095, 2019. URL https://ieeexplore.ieee.org/abstract/document/8756165.

[79] 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. URL https://www.deepgcns.org/arch/deep.

[80] H. Li, Z. Han, Y. Sun, F. Wang, P. Hu, Y. Gao, X. Bai, S. Peng, C. Ren, X. Xu, et al. Cgmega: explainable graph neural network framework with attention mechanisms for cancer gene module dissection. *Nature Communications*, 15(1):5997, 2024. URL https://doi.org/10.1038/s41467-024-50426-6.

[81] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015. URL https://arxiv.org/abs/1511.05493.

[82] R. Liao, Z. Zhao, R. Urtasun, and R. S. Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. *arXiv preprint arXiv:1901.01484*, 2019. URL https://arxiv.org/abs/1901.01484.

[83] D. Lim, F. M. Hohne, X. Li, S. L. Huang, V. Gupta, O. P. Bhalerao, and S.-N. Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. In *Advances in Neural Information Processing Systems*, 2021. URL https://dl.acm.org/doi/10.5555/3540261.3541859.

[84] E. Luzhnica, B. Day, and P. Lio. Clique pooling for graph classification. *arXiv preprint arXiv:1904.00374*, 2019. URL https://arxiv.org/abs/1904.00374.

[85] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD international conference*

on knowledge discovery & data mining, pages 723–731, 2019. URL https://dl.acm.org/doi/10.1145/3292500.3330982.

[86] S. K. Maurya, X. Liu, and T. Murata. Improving graph neural networks with simple architecture design. *arXiv preprint arXiv:2105.07634*, 2021. URL https://arxiv.org/abs/2105.07634.

[87] S. K. Maurya, X. Liu, and T. Murata. Simplifying approach to node classification in graph neural networks. *Journal of Computational Science*, 62:101695, 2022. URL https://api.semanticscholar.org/CorpusID:244102809.

[88] C. Mavromatis and G. Karypis. Gnn-rag: Graph neural retrieval for large language model reasoning. *arXiv preprint arXiv:2405.20139*, 2024. URL https://arxiv.org/abs/2405.20139.

[89] E. A. Meyer, R. K. Castellano, and F. Diederich. Interactions with aromatic rings in chemical and biological recognition. *Angewandte Chemie International Edition*, 42(11):1210–1250, 2003. URL https://doi.org/10.1002/anie.200390319.

[90] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021. URL https://ieeexplore.ieee.org/document/9356353.

[91] 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*, pages 4602–4609, 2019. URL https://dl.acm.org/doi/10.1609/aaai.v33i01.33014602.

[92] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020. URL https://arxiv.org/abs/2007.08663.

[93] 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, ICML*, pages 25956–25979. PMLR, 2023. URL https://dl.acm.org/doi/10.5555/3618408.3619488.

[94] Y. Pang, Y. Zhao, and D. Li. Graph pooling via coarsened graph infomax. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2177–2181, 2021. URL https://dl.acm.org/doi/10.1145/3404835.3463074.

[95] J. Park, H. Shim, and E. Yang. Graph transplant: Node saliency-guided graph mixup with local structure preservation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7966–7974, 2022. URL url={https://api.semanticscholar.org/CorpusID:243938320}.

[96] A. Paszke, S. Gross, S. Chintala, G. Chanan, Z. Yang, Edward, A. Lin, Zeming, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[97] M. Pavan and M. Pelillo. A new graph-theoretic approach to clustering and segmentation. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 1, pages I–I. IEEE, 2003. URL https://ieeexplore.ieee.org/document/1211348.

[98] M. Pavan and M. Pelillo. Dominant sets and pairwise clustering. *IEEE transactions on pattern analysis and machine intelligence*, 29(1):167–172, 2006. URL https://ieeexplore.ieee.org/document/4016559.

[99] 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. URL https://arxiv.org/abs/2002.05287.

[100] M. Pelillo and A. Torsello. Payoff-monotonic game dynamics and the maximum clique problem. *Neural Computation*, 18(5):1215–1258, 2006.

[101] L. Perez and J. Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017. URL https://arxiv.org/abs/1712.04621.

[102] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini. Recipe for a General, Powerful, Scalable Graph Transformer. *arXiv:2205.12454*, 2022. URL https://arxiv.org/pdf/2205.12454.pdf.

[103] M. Réau, N. Renaud, L. C. Xue, and A. M. Bonvin. Deeprank-gnn: a graph neural network framework to learn patterns in protein–protein interfaces. *Bioinformatics*, 39(1):btac759, 2023. URL https://doi.org/10.1093/bioinformatics/btac759.

[104] S. Rhee, S. Seo, and S. Kim. Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. *arXiv preprint arXiv:1711.05859*, 2017. URL https://arxiv.org/abs/1711.05859.

[105] K. Riesen and H. Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 287–297. Springer, 2008. URL https://link.springer.com/chapter/10.1007/978-3-540-89689-0_33.

[106] Y. Rong, W. Huang, T. Xu, and J. Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019. URL https://arxiv.org/abs/1907.10903.

[107] B. Rozemberczki and R. Sarkar. Twitch gamers: a dataset for evaluating proximity preserving and structural role-based node embeddings. *arXiv preprint arXiv:2101.03091*, 2021. URL https://arxiv.org/abs/2101.03091.

[108] B. Rozemberczki, C. Allen, and R. Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021. URL https://arxiv.org/abs/1909.13021.

[109] T. K. Rusch, M. M. Bronstein, and S. Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023. URL https://arxiv.org/abs/2303.10993.

[110] H. Sasaki, S. Yamamoto, A. Agchbayar, and N. Nkhbayasgalan. Extracting problem linkages to improve knowledge exchange between science and technology domains using an attention-based language model. *Engineering, Technology & Applied Science Research*, 10(4):5903–5913, 2020. URL https://doi.org/10.48084/etasr.3598.

[111] 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, 2009. URL https://ieeexplore.ieee.org/document/9869643.

[112] F. M. Schmitt-Koopmann, E. M. Huang, H.-P. Hutter, T. Stadelmann, and A. Darvishy. FormulaNet: A benchmark dataset for mathematical formula detection. *IEEE Access*, 10:91588–91596, 2022. URL https://ieeexplore.ieee.org/document/4700287.

[113] 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. URL https://ojs.aaai.org/index.php/aimagazine/article/view/2157.

[114] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. URL https://dl.acm.org/doi/10.1109/34.868688.

[115] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019. URL https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0.

[116] M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on*

computer vision and pattern recognition, pages 3693–3702, 2017. URL https://ieeexplore.ieee.org/document/8099494.

[117] F. Smarandache. Extension of hypergraph to n-superhypergraph and to plithogenic n-superhypergraph, and extension of hyperalgebra to n-ary (classical-/neutro-/anti-)hyperalgebra. *Neutrosophic Sets and Systems*, 33:290–296, 2020. URL https://api.semanticscholar.org/CorpusID:219486503.

[118] Y. Song, C. Zhou, X. Wang, and Z. Lin. Ordered GNN: Ordering message passing to deal with heterophily and over-smoothing. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2302.01524.

[119] Z. Song, X. Yang, Z. Xu, and I. King. Graph-based semi-supervised learning: A comprehensive review. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):8174–8194, 2022. URL url={https://ieeexplore.ieee.org/document/9737635}.

[120] M. Sun, M. Yang, Y. Li, D. Mu, X. Wang, and Y. Wang. Structural-aware motif-based prompt tuning for graph clustering. *Information Sciences*, 649:119643, 2023. URL https://doi.org/10.1016/j.ins.2023.119643.

[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*, 2021. URL https://arxiv.org/abs/2111.14522.

[122] D. D. Van. Application of advanced deep convolutional neural networks for the recognition of road surface anomalies. *Engineering, Technology & Applied Science Research*, 13(3):10765–10768, 2023. URL https://doi.org/10.48084/etasr.5890.

[123] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. URL https://arxiv.org/abs/1706.03762.

[124] A. Velingker, A. K. Sinop, I. Ktena, P. Veličković, and S. Gollapudi. Affinity-aware graph networks. *arXiv preprint arXiv:2206.11941*, 2022. URL https://arxiv.org/pdf/2206.11941.pdf.

[125] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks, 2017. URL https://arxiv.org/abs/1710.10903.

[126] V. Verma, M. Qu, K. Kawaguchi, A. Lamb, Y. Bengio, J. Kannala, and J. Tang. Graphmix: Improved training of gnns for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, 2021. URL https://arxiv.org/abs/1909.11715.

[127] O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015. URL https://arxiv.org/abs/1511.06391.

[128] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17: 395–416, 2007. URL https://arxiv.org/abs/0711.0189.

[129] N. Wale, I. A. Watson, and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14 (3):347–375, 2008. URL https://api.semanticscholar.org/CorpusID:2596211.

[130] G. Wang, R. Ying, J. Huang, and J. Leskovec. Multi-hop attention graph neural networks. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI*, pages 3089–3096. ijcai.org, 2021. URL https://doi.org/10.24963/ijcai.2021/425.

[131] X. Wang, B. Qian, and I. Davidson. On constrained spectral clustering and its applications. *Data Mining and Knowledge Discovery*, 28:1–30, 2014. URL https://link.springer.com/article/10.1007/s10618-012-0291-9.

[132] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi. Graphcrop: Subgraph cropping for graph classification. *arXiv preprint arXiv:2009.10564*, 2020. URL https://arxiv.org/abs/2009.10564.

[133] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi. Mixup for node and graph classification. In *Proceedings of the Web Conference 2021*, pages 3663–3674, 2021. URL https://doi.org/10.1145/3442381.3449796.

[134] Y. Wang, W. Hou, N. Sheng, Z. Zhao, J. Liu, L. Huang, and J. Wang. Graph pooling in graph neural networks: methods and their applications in omics studies. *Artificial Intelligence Review*, 57(11):294, 2024. URL https://link.springer.com/article/10.1007/s10462-024-10918-9.

[135] Z. Wang, M. Liu, Y. Luo, Z. Xu, Y. Xie, L. Wang, L. Cai, Q. Qi, Z. Yuan, T. Yang, et al. Advanced graph and sequence neural networks for molecular property prediction and drug discovery. *Bioinformatics*, 38, 2022. URL https://doi.org/10.1093/bioinformatics/btac112.

[136] J. W. Weibull. *Evolutionary game theory*. MIT press, 1997.

[137] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019. URL http://arxiv.org/abs/1901.00596.

[138] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018. URL https://arxiv.org/abs/1810.00826.

[139] K. Xu, M. Zhang, S. Jegelka, and K. Kawaguchi. Optimization of graph neural networks: Implicit acceleration by skip connections and more depth. In *International Conference on Machine Learning*, pages 11592–11602. PMLR, 2021. URL https://proceedings.mlr.press/v139/xu21k.html.

[140] Y. Xu, J. Wang, M. Guang, C. Yan, and C. Jiang. Multistructure graph classification method with attention-based pooling. *IEEE Transactions on Computational Social Systems*, 10(2):602–613, 2022. URL https://ieeexplore.ieee.org/document/9767199.

[141] T. Yamamoto. Crystal graph neural networks for data mining in materials science. *Research Institute for Mathematical and Computational Sciences, LLC*, 2019. URL https://api.semanticscholar.org/CorpusID:198989570.

[142] 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 *IEEE International Conference on Data Mining (ICDM)*, pages 1287–1292. IEEE, 2022. URL https://ieeexplore.ieee.org/document/10027737.

[143] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018. URL https://dl.acm.org/doi/10.5555/3327345.3327389.

[144] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33:5812–5823, 2020. URL https://dl.acm.org/doi/10.5555/3495724.3496212.

[145] H. Yuan and S. Ji. Structpool: Structured graph pooling via conditional random fields. In *Proceedings of the 8th International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJxg_hVtwH.

[146] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. *Advances in neural information processing systems*, 17, 2004. URL https://www.researchgate.net/publication/215721835_Self-Tuning_Spectral_Clustering.

[147] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *In Proceedings of the Association for the Advancement of Artificial Intelligence, (AAAI)*, 2018. URL https://dl.acm.org/doi/abs/10.5555/3504035.3504579.

[148] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015. URL https://arxiv.org/abs/1509.01626.

[149] X.-M. Zhang, L. Liang, L. Liu, and M.-J. Tang. Graph neural networks and their current applications in bioinformatics. *Frontiers in genetics*, 12:690049, 2021. URL https://doi.org/10.3389/fgene.2021.690049.

[150] Z. Zhang, H. Yang, J. Bu, S. Zhou, P. Yu, J. Zhang, M. Ester, and C. Wang. Anrl: attributed network representation learning via deep neural networks. In *In International Joint Conferences on Artificial Intelligence, (IJCAI)*, volume 18, pages 3155–3161, 2018. URL https://ijcai.org/proceedings/2018/438.

[151] Z. Zhang, J. Bu, M. Ester, J. Zhang, C. Yao, Z. Yu, and C. Wang. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954*, 2019. URL https://arxiv.org/abs/1911.05954.

[152] Z. Zhang, J. Bu, M. Ester, J. Zhang, Z. Li, C. Yao, H. Dai, Z. Yu, and C. Wang. Hierarchical multi-view graph pooling with structure learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(1):545–559, 2021. URL https://ieeexplore.ieee.org/document/9460814.

[153] T. Zhao, W. Jin, Y. Liu, Y. Wang, G. Liu, S. Günnemann, N. Shah, and M. Jiang. Graph data augmentation for graph machine learning: A survey. *arXiv preprint arXiv:2202.08871*, 2022. URL https://arxiv.org/abs/2202.08871.

[154] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020. URL https://doi.org/10.1016/j.aiopen.2021.01.001.

[155] J. Zhou, J. Shen, S. Yu, G. Chen, and Q. Xuan. M-evolve: structural-mapping-based data augmentation for graph classification. *IEEE Transactions on Network Science and Engineering*, 8(1):190–200, 2020. URL https://arxiv.org/abs/2007.05700.

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

of the 34th International Conference on Neural Information Processing Systems, NIPS, 2020. URL https://dl.acm.org/doi/10.5555/3495724.3496377.

[157] C. Zhuang, N. J. Yuan, R. Song, X. Xie, and Q. Ma. Understanding people lifestyles: Construction of urban movement knowledge graph from gps trajectory. In *International Joint Conferences on Artificial Intelligence, IJCAI*, pages 3616–3623, 2017. URL https://www.ijcai.org/proceedings/2017/506.