Title

Towards Effective And Efficient Graph Neural Networks

Author

WANG, YEWEN

Publication Date

2024

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

Towards Effective And Efficient Graph Neural Networks

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Yewen Wang

2024

ABSTRACT OF THE DISSERTATION

Towards Effective And Efficient Graph Neural Networks

by

Yewen Wang
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2024
Professor Yizhou Sun, Chair

Graph is a pervasive data type in the real world, as it serves as a succinct yet powerful abstraction for entities and their interconnections. Consequently, generating high-quality graph representations that encode the graph information is important, as these representations would be instrumental in graph-related tasks. In this context, Graph Neural Networks (GNNs) have emerged as a significant advancement and gained prominence for their ability to learn powerful graph representations which lead to state-of-the-art performance across a variety of graph-based applications.

However, despite their remarkable capabilities, the design and training processes of GNNs are still fraught with challenges. Given this, my research goal is to identify and address these hurdles from both the effectiveness and efficiency perspectives, aiming to develop GNN models that are potent yet scalable, thereby enhancing GNN's power in producing superior graph representations in practice. This dissertation systematically investigates two fundamental questions: (1) What impedes the effectiveness of GNNs?

(2) How to train GNNs efficiently? Despite providing a comprehensive discussion of each question, it presents practical solutions to mitigate each side and covers both homogeneous graphs and heterogeneous graphs.

The dissertation of Yewen Wang is approved.

Cho-jui Hsieh

Quanquan Gu

Wei Wang

Yizhou Sun, Committee Chair

University of California, Los Angeles

2024

To my family, and myself.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The journey to completing my Ph.D. in UCLA's Computer Science Program has been a challenging yet rewarding experience filled with personal discovery, growth, exhilarating highs, inevitable lows, and moments of joy and tears. This incredible journey, would not have been possible without the companionship, support, guidance, and inspiration of so many brilliant people, my advisors and mentors, collaborators and labmates, instructors, staff, family, and friends.

First and foremost, I want to extend my heartfelt thanks to my advisor, Professor Yizhou Sun, for her invaluable mentorship which is pivotal for my academic journey. Her remarkable intellect, deep knowledge, passion for pioneering scientific problems, and dedicated commitment to research have set a role model that I have continually aspired to reach. Her warm encouragement, hands-on guidance, and steadfast support foster an exceptional environment for me to learn and grow.

I am very grateful to my esteemed committee members: Professor Wei Wang, Professor Quanquan Gu, and Professor Cho-Jui Hsieh. Their academic rigor and discerning feedback significantly broadened my academic perspective and helped me to refine my research. Additionally, I am deeply thankful to Professor Junghoo Cho for his insightful questions and comments during our reading groups. His challenging questions prompted me to delve deeper into various technical and research issues, enhancing my understanding and familiarity with these topics, and also inspired me how to be analytical and thoughtful when learning things.

I am very fortunate to have had the opportunity to complete four internships and collaborate with a group of remarkably talented individuals. My gratitude extends to my mentors at AWS: Soji Adeshina, Vassilis N. Ioannidis, Jiani Zhang, Xiang Song,

Da Zheng, Christos Faloutsos, and George Karypis, and many of my peer interns, with a special thanks to Soji Adeshina. As my manager during all three of my internships at AWS, Soji fostered numerous inspiring discussions about my projects and provided patient, detailed guidance on coding techniques. The internships at AWS broadened my research horizons into the realm of heterogeneous graphs and have also paved the way for me to join Amazon after graduation, marking the beginning of my professional career. In addition, I am grateful to my mentors at MILA, Professor Guy Wolf and Professor Jian Tang, for motivating me to explore the field of parallelization for GNN modeling. Collaborating with all these distinguished individuals has been an invaluable learning experience for me.

I am deeply thankful to be part of such a warm and brilliant lab, and I wish to express my appreciation to my labmates: Ting Chen, Yupeng Gu, Xuelu Chen, Junheng Hao, Yunsheng Bai, Ziniu Hu, Patricia Xiao, Vivian Cheng, Song Jiang, Roshni Iyer, Zijie Huang, Shichang Zhang, Zongyue Qin, Derek Xu, Arjun Subramonian, Fred Xu, Yanqiao Zhu, Fang Sun, Weikai Li, Zongyu Lin, Jiaqi Zhu, Changjun Fan, Jiarong Xu, Xiao Luo, Zijun Xue, Jin Wang, Manoj Reddy, Shengming Zhang, Zeyu Li, Xiusi Chen, Mingyu Derek Ma, Mandy Wang, and Yijia Xiao. Our shared experiences and mutual support have greatly enriched my Ph.D. journey, making this challenging path of academic pursuit enjoyable.

I extend my thanks to the course instructors who provided me with the invaluable opportunity to serve as a TA: Yizhou Sun, Quanquan Gu, and David Smallberg. Their exceptional lectures and meticulously prepared materials have been instrumental in enhancing my knowledge of course design, making my TA experience rewarding and enjoyable. Additionally, I learned a lot from the courses I have taken and am grateful for these amazing instructors as well: Richard Korf, Yizhou Sun, Quanquan

Gu, Lieven Vandenberghe, Junghoo Cho, Ernest K. Ryu, Wei Wang, Cho-Jui Hsieh, Paul Eggert, and Songwu Lv.

My appreciation also goes to the dedicated UCLA CS department staff, for their diligent efforts in maintaining an efficient and supportive departmental environment. Special thanks to Madelene Hem and Osanna Kazarian for their management of our lab, ensuring it remains a conducive space for research and innovation, to Joseph Brown and Helen Tran for assisting with doctoral academic affairs, and to Ning Jiang, Cassandra Franklin, and Diana de los Santos for managing funding and financial matters, to Mildri Lopez-Duarte for issuing keys to access lab, and to Jade Hill and Juliana Alvarez for keeping us well-informed with regular updates.

Last but most importantly, I want to say a huge thank you to my cherished family and best friends. To my father and my mother, your unconditional love and unwavering support transcend the mere words I can find to describe. Your enduring belief in me has been my strength in all the hardest moments, giving me the courage and confidence to get through the dark times. To my husband, your understanding and companionship are the bedrock of this journey, providing solace and joy in my most struggling moments. My heartfelt thanks also go to my dear friends: Jasmine Ouyang, Wendy Ni, Zoey Zhou, and Young Zhang, for always being there to give unending support and comfort to me in my down times. The depth of my gratitude for my family and friends is boundless, they are my refuge, and my pillar of strength.

As this chapter of my journey concludes, I wish to express my heartfelt gratitude to every individual who has impacted my life and work over the past five and a half years. To each of you, I extend my best wishes for the future, hoping that it holds success, happiness, and fulfillment. Thank you for being a part of my journey.

VITA

2014–2018	B.Eng. in Automation, Tsinghua University Beijing, China.
2019–2021	Teaching Assistant/Associate, Computer Science Department, UCLA Los Angeles, CA, United States
2020	Graduate Research Intern, Quebec AI Institute (MILA) Montréal, QC, Canada
2021	Applied Scientist Intern, Amazon Web Services Santa Clara, CA, United States.
2022	Applied Scientist Intern, Amazon Web Services Santa Clara, CA, United States.
2023	Applied Scientist Intern, Amazon Web Services Santa Clara, CA, United States.
2018 - 2023	Graduate Student Researcher, Computer Science Department, UCLA Los Angeles, CA, United States

PUBLICATIONS

Yewen Wang, Shichang Zhang, Junghoo Cho, Yizhou Sun, Laplacian Score Benefit Adaptive Filter Selection for Graph Neural Networks, Proceedings of the 2024 SIAM International Conference on Data Mining (*SDM 2024*, *SoCal-KDD23*)

Zhicheng Ren, Yifu Yuan, Yuxin Wu, Xiaxuan Gao, **Yewen Wang**, Yizhou Sun, Dissimilar Nodes Improve Graph Active Learning, GLFrontiers 2022 Workshop: New Frontiers in Graph Learning (*GLFrontiers-NeurIPS22*)

Yewen Wang, Jian Tang, Yizhou Sun, Wolf Guy, Decoupled Greedy Learning of Graph Neural Networks, OPT2020 Workshop: Optimization for Machine Learning (*OPT-NeurIPS20*)

Yewen Wang, Ziniu Hu, Yusong Ye, Yizhou Sun, Demystifying Graph Neural Networks with Graph Filter Assessment, The Second International Workshop on Deep Learning on Graphs: Methods and Applications (*DLG-KDD20*)

Difan Zou, Ziniu Hu, **Yewen Wang**, Song Jiang, Yizhou Sun, Quanquan Gu, Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks, 2019 Neural Information Processing System (*NeurIPS19*)

CHAPTER 1

Introduction

1.1 Motivation

Graph Neural Network (GNN) is a family of prominent models for learning high-quality representations for ubiquitous graph data, and has gained more and more attention due to its State-Of-The-Art (SOTA) performance across diverse graph-related tasks [14, 109, 139, 10]. However, achieving effective and efficient GNN models presents significant challenges. On the effectiveness side, issues such as over-smoothing [64] where node features become non-distinguishable when the network gets deeper hampers deep GNN architectures from being effective, and heterophily[1] [143] where nodes connected by edges do not necessarily share similarities betrays the underlying homophily assumptions in most of the GNN models, pose considerable hurdles. On the efficiency side, challenges such as update-locking [49] hamper parallel computations, while the inherently costly nature of graph convolutional operations [144] makes large-scale deep GNN applications problematic.

A lot of research efforts are paid to making GNN models more powerful [53, 141, 100] and more efficient [132, 68]. For the effective side, various prominent GNN

[1]Please note that, heterophilic graph and heterogeneous graph are different. Heterophilic graph belongs to the category of homogeneous graphs. Chapter 2 provides a concrete definition and examples for both.

architectures such as GCN [55], GAT [95], deepGCN [64, 61], SIGN [31], GIANT [22] are developed to enhance GNN's representation learning capabilities, while another branch of work such as RGCN [82], NARS [123], MAGNN [129], CLGNN [105] aims to tailor architectures for learning with the heterogeneous graphs. There are also many works that pay attention to identifying and addressing the challenging cases for GNNs such as H2GCN [143], EGP [24] GFNN [73]. For the efficiency side, people develop algorithms to train GNNs more efficiently such as GraphSAGE [38], GraphSAINT [128], GOREN [12], and GCond [52]. Efforts are also extended from inference [131, 91], system [97, 136], hardware [126, 35] perspectives to make GNNs more efficient. Despite these extensive research efforts aimed at enhancing the effectiveness and efficiency of GNNs, existing solutions are insufficient, leaving ample scope for exploration in the field.

To this end, my goal is to develop powerful GNN models that can achieve high-quality graph representations even under demanding situations and enable GNNs to be trained efficiently for large-scale applications under time and memory constraints. My research is mainly structured around the following research questions:

1. Regarding effectiveness, given that the graph convolutional operation is pivotal in empowering GNNs and considering the variety of graph convolutional operators [2] available, we should understand: What specific applicability and limitation scenarios exist for each operator? Additionally, how can we adaptively employ these operators across various graph representation tasks to optimize their utility and performance, especially when facing the heterophilic challenge and

[2]We should distinguish the term "Graph convolution operator" and "Graph convolutional filter", the filter is a matrix assigning weights on each edge for message passing, while the operator indicates applying the filter and conduct message passing on the graph.

the over-smoothing challenge mentioned above?

2. Regarding efficiency, as graph convolutional operations bring the exponential neighborhood growth problem when conducting batch-wise training on large-scale graphs, we should think: How can we mitigate this exponential neighborhood expansion to reduce the computational burden? To take a step further, can we address the update-unlocking which is an inherent challenge that hinders neural networks from being more efficient as well?

3. There are usually trade-offs between the effectiveness side and the efficiency side, a lot of work may trade in one side when focusing on the other (especially when the focus is on the effectiveness side). Therefore, we think about, can we develop GNN models to ensure that improvements in one aspect do not undermine the other?

4. When transitioning the focus to heterogeneous graphs, how do we manage their intrinsic heterogeneity and customize effective and efficient message-passing strategies for them?

1.2 Thesis Contribution

This dissertation delves into the motivations outlined in the previous section, and its major contributions are summarized as follows.

- Regarding effectiveness, my research targets the creation of an adaptive graph convolutional operator design. We start by examining a comprehensive set of existing operators, subsequently selecting a specific operator family as our

focus. We justify the necessity to consider each operator within this family, and introduce an innovative adaptive soft-selection mechanism. This scheme is designed to learn the most suitable operator for any given graph representation task, thereby enhancing the effectiveness of GNNs.

- On the efficiency front, we develop a GNN model in which each layer could be trained in parallel so that the training time can be greatly reduced and the memory cost can be distributed to multiple devices. With prior findings that GNN layers need not be trained jointly, we propose a novel approach that decouples the GNNs into blocks and trains each block independently with auxiliary greedy learning objectives in parallel. We show that this method mitigates the challenges of update-locking and exponential neighbor growth, resulting in markedly increased training efficiency.

- Concerning the inherent trade-offs between effectiveness and efficiency, we care about developing a balanced approach that caters to both sides concurrently. Though it is nearly impossible to develop models that could be the best in terms of both effectiveness and efficiency, we consider scalability as well when targeting a model design to get top effectiveness, and vice versa, ensuring our advancement on one side does not come at the price of the other.

- Concerning the heterogeneity, we tailor message-passing and efficiency-enhancing techniques to suit various node and edge types. To achieve an effective design for heterogeneous GNNs, we introduce an adaptive message-passing scheme that adjusts to a variety of meta-paths, while for efficient training of heterogeneous GNNs, we develop a bi-level sampling strategy that customizes neighbor distributions for distinct node types.

The thesis aims to make a substantive contribution to the advancement of GNNs, paving the way for their broader application and impact across various domains.

1.3 Thesis Outline

The rest of this dissertation is structured as follows:

Chapter 2 establishes the foundational background for this thesis. It provides formal definitions, and examples for both homogeneous and heterogeneous graphs, provides use cases on graph-related applications, and goes through prominent literature for graph representation learning from shallow models to deep models and from homogeneous graphs to heterogeneous graphs.

Chapter 3 focuses on enhancing the effectiveness of GNNs for homogeneous graphs. The graph convolutional filter (GCF) for aggregating neighbor information is shown to be the key factor that leads to GNNs' success. Various GCFs are designed but ***how to select the proper filter*** that can best benefit the data and the task remains an open problem. Thus, to improve the effectiveness, we introduce the **Ada**ptive **F**ilter **S**election (AdaFS) framework that addresses two critical issues: (1) defining a criterion to establish a strong base filter set; and (2) adaptively selecting filters for a specific task, even when labeled data is limited, by employing Laplacian score regularization. We further connect this multiple GCF learning process and the well-developed multiple kernel learning problem to provide a solid rationale for filter selection. The proposed AdaFS achieves top-tier performance in benchmarks and is shown to enjoy the benefit of mitigating the over-smoothing and heterophilic

challenge.

Chapter 4 also concentrates on the effectiveness, but shifts the focus to heterogeneous graphs. Heterogeneous graphs have strong capability of modeling multi-typed nodes and relations, and Heterogeneous GNNs have been greatly explored due to their SOTA performance on heterogeneous graphs. Existing works can be divided into two categories: relation-based ones which construct subgraphs based on the relation subsets and consider message passing and aggregation on the subgraphs; and metapath-based ones which perform message passing via metapaths on the full graph. The relation-based methods may lose the rich semantics as the relation subsets will inevitably fail to cover all the metapaths, while the metapath-based ones suffer scalability bottlenecks since the number of metapath in the full graph is usually exponential to the path length. To address the challenges from both sides and develop effective heterogeneous GNNs, we propose the **S**calable **M**etapath **A**ggregation ba**S**ed **H**eterogeneous GNN model (SMASH), which shows great performance in terms of both accuracy and scalability.

Chapter 5 tackles the efficiency aspect of homogeneous GNNs by developing a **D**ecoupled **G**reedy **L**earning **GNN** (DGL-GNN) approach combined with a lazy update scheme for GNN training. The computational intensity of graph convolution operations and the update-locking issue are notable obstacles in large-scale GNN training. Intending to overcome these hurdles, we introduce the DGL-GNN which decouples the GNN into smaller modules and associates each module with greedy auxiliary objectives. Our approach allows GNN layers to be updated during the training process without waiting for feedback from successor layers, thus making

parallel GNN training possible. Further, we propose a lazy-update scheme during training to further improve its efficiency. Despite its success in reducing the per-GPU memory cost and total running time without significantly compromising model accuracy, the proposed algorithm is compatible with other scalability-enhancing methods such as sampling, and can be integrated with these methods to achieve even greater improvements in efficiency.

Chapter 6 also aims at the efficient training but focuses on heterogeneous GNNs, by introducing a bi-level sampling technique. The uncontrollable neighbor expansion brought by the graph convolutional operation also bothers heterogeneous GNNs. While uniform random sampling has been shown to successfully speed up GNN model training, we observe that uniform random sampling is not enough for heterogeneous GNNs. We examine the performance of heterogeneous GNNs trained with random sampling with and without considering the heterogeneity and find that they fail in the two following cases respectively: 1) a "noisy" graph in which many neighboring node instances do not provide information for the downstream task; 2) a schema-complex graph which includes numerous relation types but has a limited sampling budget. As such, we posit that training heterogeneous GNNs requires a non-uniform relation-importance-aware sampling method for fast and accurate performance. Current sampling methods are mostly either designed for homogeneous graphs and are not capable of handling heterogeneity properly, or are task-agnostic which apply the same sampling heuristics regardless of the downstream tasks and do not explicitly sample to boost the model effectiveness on the current task. Therefore, we propose the Bi-Level Adaptive Sampling (BLADS), to conduct the sampling from both schema and instance levels. The schema-level sampler learns to generate more samples of the

critical relation types and the entity level generates diverse representatives of each relation type uniformly.

Chapter 7 concludes the dissertation, summarizing my research outcomes and discussing potential directions for future research.

CHAPTER 2

Background

2.1 Graphs: Definition, Key Concepts, and Applications

Graphs are ubiquitous in the real world and they serve as an insightful abstraction in numerous domains, capturing the essence of entities and their interactions. For instance, a chemical compound can be represented with a molecular graph, in which nodes are atoms and edges are chemical bonds. Similarly, Facebook's friendship network could be represented with a simple social network, nodes represent individuals, while edges signify friendships. Moreover, in the recommender systems, we could have user-product graphs that depict users and products as nodes, interconnected by edges that may denote relationships such as purchase history or shared preferences.

In general, these graphs fall into two primary categories: homogeneous graphs and heterogeneous graphs. Homogeneous graphs consist of a single type of node and edge, making their structure and learning them relatively straightforward. In contrast, heterogeneous graphs, which consist of multiple types of nodes or edges (could be both as well), are capable of preserving richer semantics but also present more challenges in learning, because diverse types of nodes and links have their type-specific feature spaces and feature distributions and the structure also gets more complex. In the aforementioned examples, the molecular graphs and simple social

networks are homogeneous, while the user-product graphs in recommender systems are heterogeneous. We give formal definitions to each graph type and present the key related concepts in the following.

Definition 1 (Homogeneous Graph) *A homogeneous graph is defined and denoted as* $\mathcal{G}_{hom}(\mathcal{V}_{hom}, \mathcal{E}_{hom})$*, where* $\mathcal{V}_{hom} = \{v_1, v_2, ..\}$ *is the node set, and* $\mathcal{E}_{hom} = \{e_1, e_2, ...\}$ *is the edge set. Each homogeneous graph is associated with an adjacency matrix* $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ *(*$\boldsymbol{A}_{i,j} = 1$ *if there exists an edge between node* v_i *and* v_j*, otherwise 0) and a feature matrix* $\boldsymbol{X} \in \mathbb{R}^{n \times d}$*, where* n *is the total number of nodes in* $\mathcal{G}_{hom}$*, and* d *is the input feature dimension. In this dissertation, we omit the under-script "*$_{hom}$*" for simplicity when we specify the homogeneous context in the chapters.*

Definition 2 (Heterogeneous Graph) $\mathcal{G}_{het}(\mathcal{V}_{het}, \mathcal{E}_{het}, \Phi, \Psi)$*, where* $\mathcal{V}_{het} = \{v_1, v_2, ..\}$ *is a multi-type node set,* $\mathcal{E}_{het} = \{e_1, e_2, ...\}$ *is a multi-type edge set,* $\Phi : \mathcal{V}_{het} \rightarrow \mathcal{A}$ *is a mapping function that associate each node* v_i *to a node type* a *in node type set* $\mathcal{A}$*, and* $\Psi : \mathcal{E} \rightarrow \mathcal{R}$ *is a mapping function that associate each edge* e_i *to an relation type* r *in relation type set* $\mathcal{R}$*. Each heterogeneous graph is associated with a feature matrix set* $\{\boldsymbol{X}_a : a \in \mathcal{A}\}$ *where* $\boldsymbol{X}_a \in \mathbb{R}^{n_a \times d_a}$ *is the feature matrix for node type* a*,* n_a *is the number of nodes with type* a *and* d_a *is the feature dimension. We assume all node types have the same initial feature dimension for simplicity, i.e.* $\forall a, d_a = d$*. Each heterogeneous graph is also associated with an adjacency matrix set* $\{\boldsymbol{A}_r : r \in \mathcal{R}\}$*, where* $\boldsymbol{A}_r \in \mathbb{R}^{n_{a_{src}} \times n_{a_{dst}}}$*,* $n_{a_{src}}$ *and* $n_{a_{dst}}$ *are the node number for node type of source and destination node respectively. In this dissertation, we omit the under-script "*$_{het}$*" for simplicity when we specify the heterogeneous context in the chapters.*

There are many key concepts associated with heterogeneous graphs. We summarize the key concepts we will be using in the dissertation as follows. We provide a concrete

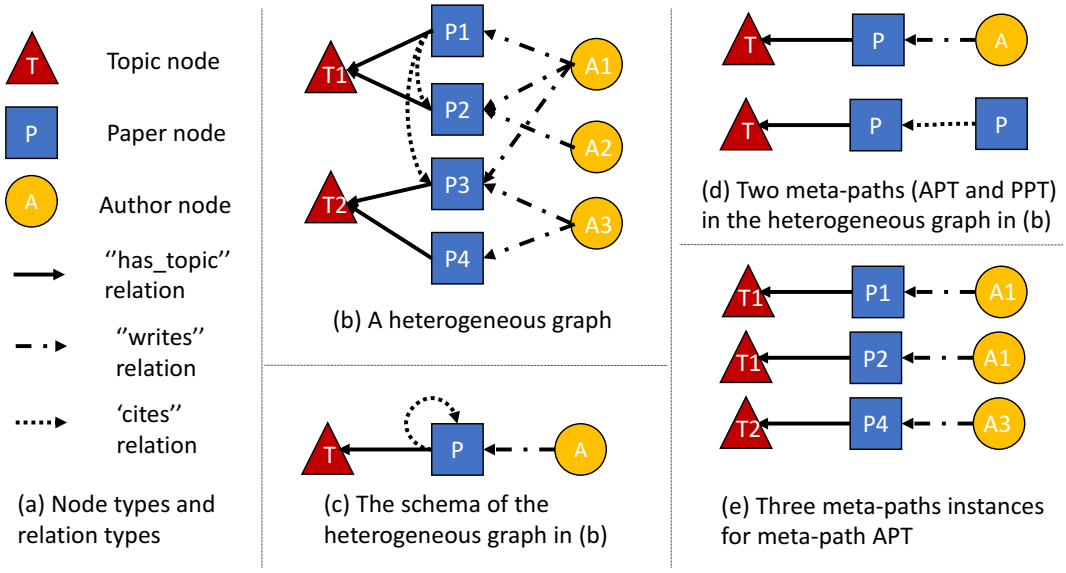


Figure 2.1: Example of an academic heterogeneous graph and related concepts.

example to explain the definition and key concepts in heterogeneous graphs in Figure 2.1 to help understanding.

Definition 3 (Schema) *The network schema [90] or the metagraph is the abstract graph representation of a heterogeneous graph, it preserves all the node types and their relations, but each type only has one instance.*

Definition 4 (Metapath) *A length-k metapath [90] p is a path defined on the network schema:* $a_1 \xrightarrow{r_1} a_2 \xrightarrow{r_2} \ldots \xrightarrow{r_{k-1}} a_k$*, where each* r_i *and* a_i *specifies a relation type and a node type respectively.*

Definition 5 (Metapath Instance) *A metapath instance of a metapath is a path:* $v_1 \xrightarrow{e_1} v_2 \xrightarrow{e_2} \ldots \xrightarrow{e_{k-1}} v_k$*, where each* e_i *and* v_i *is a relation and a node of the corresponding relation and node type in the metapath respectively.*

Various real-world applications can be formulated into graph-related tasks, enabling them to be solved with advanced graph-learning techniques. Here we provide some illustrative examples. At the instance level which we make predictions to node or edge, we have:

- **Node Classification** in which we predict labels for each node. Fraud detection in financial networks can be modeled as a node classification task, where we classify nodes in a transaction network to identify fraudulent activities [69]. Protein function prediction can also be formulated into a node classification task, where we classify proteins' (nodes) functions to help understand their roles in biological processes [119].

- **Link Prediction** in which we predict the edge existence between a pair of nodes. Friend recommendations in social media can be one example of this task, where we identify potential edges indicating two people may know each other and then can make "people you may know" predictions accordingly [65]. Collaboration prediction in academic networks can also be modeled as link prediction, as it forecasts edge existence indicating future collaborations in academic networks between the researcher nodes [130].

- **Ranking** which is to assign a significance score to each node and sort them in order. Influencer analysis can be modeled as a ranking task, as it identifies the most influential user nodes based on their interactions and behaviors [54].

At the graph level where we focus on entire graphs, we have tasks such as:

- **Graph Classification** in which we predict labels for each graph. Chemical

compound property classification can be formulated as a graph classification, where we classify chemical compound graphs to identify their properties [103].

- **Clustering** in which we put nodes in the graph into smaller groups. Community detection can be modeled as a clustering task where we identify node groups with common interests or characteristics in a social network [87]. Market Segmentation is another example application that can be formulated as clustering, where we aim to identify clusters of customers with similar preferences or behaviors in a retail network [28].

- **Graph Generation** in which we aim to generate new graphs with similar properties of a given graph set. Drug discovery can be modeled as graph generation, as it requires generating new molecular graph structures for potential new drugs [50].

2.2 Graph Representation Learning: From Shallow to Deep

Given the wide applications that graphs offer across the diverse real-world problems mentioned earlier, people are motivated to delve deeper into the domain of graph learning. The fundamental step is to perform node representation learning, which aims to map each node into a low-dimensional vector space.

Traditional methods for graph learning are primarily shallow models designed with the fundamental intuition that connected nodes should have similar embeddings, which encourage the learned representations to preserve the structure information of the graph. ***Matrix factorization-based methods*** [79, 2] are the first branch of work for node embedding. These methods construct proximity matrices based on

node connectivity and factorize the proximity matrix with dimensionality reduction techniques. ***Proximity-learning based methods*** [13, 75] are then developed and shown to be effective. These methods also request the construction of proximity matrices, but let the embedding matrix be fully parametrized, and then optimize the inner product of a pair of nodes' embeddings to approximate the corresponding term in the proximity matrix. Despite their achievements at the time they were invented, matrix-factorization base methods and proximity-learning-based methods heavily rely on hand-design deterministic node similarity measures. ***Random walk-based methods*** [77, 37] is also a hot branch for the shallow graph learning models. The idea is to perform random walks among the nodes in the graph, and optimize the fully parametrized embedding matrix to let the inner product of a pair of nodes' embeddings approximate their co-occurrence in the created walks. Though getting rid of the hand-design proximity matrix, the parameter size for the random walk-based methods can be big when the graph has a large scale. Shallow methods are also developed for heterogeneous graphs, based on proximity-preserving intuition as well, such as metapath2vec [29], PTE [92], and HIN2vec [32].

The aforementioned shallow methods exhibit common shortcomings that can significantly restrict their applicability: (1) Poor Generalization. The shallow models are designed for transductive settings, especially those that rely on heuristics, and could be hard to accommodate unseen nodes or generalize across diverse graph types. (2) Featureless. In shallow methods, they rely solely on structural information and neglect the rich node feature data available in many real-world graphs. In most cases, incorporating these node features is crucial as they carry inherent properties of nodes.

To address these problems, inspired by the breakthrough success of Neural Networks (NN) models in other application fields such as computer vision [96] and

natural language processing [27], people started to explore NN-based deep models for graph representation learning. Over recent years, various GNN models have been developed, including those from spatial and spectral perspectives. Spatial GNNs such as [55, 18, 95, 34] for homogeneous graphs and [33, 45, 101] for heterogeneous graphs leverage the message-passing mechanism [36] which integrates feature and structural information by letting nodes aggregate the information from their connected neighboring nodes. Spectral GNNs such as [94, 42, 142, 99] are grounded in spectral graph theory. These models employ eigen-decomposition techniques to discern graph patterns within the frequency Fourier domain.

This dissertation primarily concentrates on spatial GNN models, as they are more straightforward and have broader applications[53]. In the message-passing mechanism that is followed by most existing spatial GNN models, each node will have a representation vector initialized with its raw feature, then at each message-passing step (corresponds to one GNN layer), we perform the two operations: (1) Graph Convolutional Operation, during which each node aggregates the representations of its neighboring nodes, and (2) Projection, where these aggregated representations are transformed into a suitable embedding space. These steps facilitate the effective integration and propagation of information across the graph and naturally encode the feature and topological information of the graph in the obtained representations. This representation learning process can be described with the following mathematical forms, we start with the homogeneous case for simplicity. For matrix-wise formulation, we have:

$$\boldsymbol{H}^{(l)} = \sigma(\boldsymbol{F}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l)}) \tag{2.1}$$

where $\boldsymbol{H}^{(l)} \in \mathbb{R}^{n \times d_l}$ is the representation matrix, d_l is the dimensionality for $l-$th GNN

layer, $\boldsymbol{F} \in \mathbb{R}^{n \times n}$ is the graph convolutional filter that determines the weight on each edge for neighborhood aggregation, $\boldsymbol{W}^{(l)} \in \mathbb{R}^{d_{l-1} \times d_l}$ is the learnable transformation weight for $l-$th GNN layer for projection, σ is the non-linear transformation for projection. We have $\boldsymbol{H}^{(0)} = \boldsymbol{X}$ as initialization. For node-wise formulation, we have:

$$\boldsymbol{H}_v^{(l)} = \sigma(\Sigma_{u \in \mathcal{N}(v)} \boldsymbol{F}_{uv} \boldsymbol{H}_u^{(l-1)} \boldsymbol{W}^{(l)}) \tag{2.2}$$

where $\boldsymbol{H}_v^{(l)}$ is the representation for node v at the $l-$th GNN layer, $\mathcal{N}(v)$ is the neighbor set for node v, $\boldsymbol{F}_{uv}$ is the corresponding item in the graph convolutional filter $\boldsymbol{F}$ for edge $(u.v)$.

Starting from this formulation, various extension techniques are proposed for a more complex GNN design, such as customize $\boldsymbol{F}$ for different GNN layers [95], adding residual links to let GNN layers obtain all its predecessor layers' representations [18], or concatenate representations for all layers in the end to achieve the final node representation [19]. Besides, the formulation can be easily extended to heterogeneous graphs [100], but we may have to construct separate $\boldsymbol{H}^{(l)}$s for different node types, $\boldsymbol{F}$s for different edge types, and $\boldsymbol{W}^{(l)}$s for different node and edge type combinations.

2.3 Challenges in GNNs and Existing Solutions

2.3.1 Challenges in Making GNNs Effective

There are numerous challenges where GNNs may struggle to achieve optimal performance and thus would request specialized approaches and tailored solutions. My thesis addresses two prevalent issues in real-world applications: over-smoothing and heterophily.

Over-smoothing is about the problem that node representations converge towards the same constant value and thus become non-distinguishable when the GNN gets deeper [64]. This becomes the major bottleneck for applying the deep GNN models. Recent studies have proposed various approaches to tackle the challenges of over-smoothing [81]. ***Normalization and Regularization*** techniques mitigate the over-smoothing effect by regularizing the training objective with over-smoothing metrics or by normalizing node embeddings or introducing noise into the optimization process. EGNN [140], DropEdge [78], and PairNorm [134] are the representative works in this branch. Adding ***Residual Links*** can be another way to address over-smoothing, it considers adding residual connections to deep GNNs to obtain the distinguishable node embeddings at earlier layers. GCNII [18], JKNets [112], and DAGNN [115] are some examples within this thread.

Heterophily refers to the scenario where the labels of a node tend to be different from those of its neighbors, and therefore presenting a challenge for GNN models that typically rely on the assumption of label homophily such as [55]. People paid a lot of attention to address the homophily challenge during the past years [137]. ***Blending high-order neighbors*** is one of the solutions for heterophily, instead of only aggregating information from direct neighbors (i.e. 1-hop neighbors), it proposes to incorporate information from higher-order neighbors. MixHop [1], Ordered GNN [86], and EvenNet [58] are some examples of this kind. ***Identifying Potential Neighboring Nodes*** is another effective way to address the heterophily challenge. Unlike the methods in the previous category that adhere strictly to the existing graph structure, this category aims to discover suitable neighboring nodes for each node, thereby learning new graph structures to facilitate message passing in heterophilic contexts. Geom-GCN [76], HOG-GNN [99], and GOAL [138] are some examples using

neighbor discovery to help with heterophily. ***Adaptive Message Aggregation*** also helps with the heterophilic graphs, it learns to assign adaptive weights on each edge or each representation channel, so different neighbors and different representation channels will get customized attention during the message-passing procedure. GOAT [57], GDAMN [16], CGCN [113], and GBK-GNN [30] are some examples of this category. Besides these approaches, other models using hybrid strategies are also developed for heterophilic graphs [143, 51, 40].

2.3.2 Challenges in Making GNNs Efficient

GNNs also face significant scalability challenges when graphs get gigantic with millions or even billions of nodes, and coupled with dense interconnections. Addressing the computational demands of training and applying GNNs on such large-scale graphs, particularly under time or memory constraints, has emerged as a critical area of research. My thesis delves into two pivotal aspects of these scalability challenges: the exponential neighborhood growth resulting from graph convolutional operations and the update-locking issue common to all NN-based models.

Exponential Neighbor Growth refers to the problem that when conducting batch-wise training/inference for large-scale graphs, the neighborhood size for each node grows exponentially with the depth of the GNN (i.e., the number of GNN layers). Addressing this challenge is essential for scalable GNN applications, with strategies spanning training algorithms, inference techniques, system optimizations, and customized accelerators being developed [132]. My thesis narrows its focus to a training perspective.

Graph Modification is one way to improve efficiency, the basic idea is to

generate a modified graph that is much smaller (i.e. with much fewer nodes/edges or both) than the original graph and is thus faster to train. Graph coarsening [48], graph sparsification [62, 135], and graph condensation [52] are three representative tracks within graph modification. ***Sampling*** is another way to enhance efficiency, it dynamically selects a subset of nodes and edges for each training iteration, thus greatly reducing the computation graph size and could benefit the efficiency [67]. Various node-wise sampling [38, 15, 116], layer-wise sampling [17, 144, 46], and subgraph-wise sampling [128, 20, 127] methods are proposed toward this direction. Beyond graph modification and sampling, some research [107, 123, 19] proposes to move the graph convolutional operations to the pre-processing stage, store the aggregated features, and thus simplify training a GNN into training an MLP with the aggregated features as input. Moreover, some parallelization-based methods [97, 9] are also proposed to distribute data, tensor, or model to different devices to achieve faster training speed and less per-device GPU cost.

Update-locking states that each layer heavily relies on subsequent layers' feedback to update itself, and therefore must wait for the information to propagate through the whole network before updating. This characteristic of neural networks prevents concurrent updates across layers, resulting in inefficiencies as earlier layers remain idle awaiting gradients from later ones, thereby slowing down the training process. With the observation that the joint training objective for NNs can be relaxed [6, 120], researchers explored greedy layerwise training schemes as a solution to address the update-locking issue [47, 7]. These methods facilitate more independent and parallel layer updates, significantly enhancing training efficiency by mitigating idle wait times and expediting the learning process.

Part I

Effectivenss

CHAPTER 3

Laplacian Score Benefit Adaptive Filter Selection for Graph Neural Networks

3.1 Introduction

Graph representation learning [14] has received great attention during the past years as it maps the ubiquitous irregular structured data to informative embeddings, which benefits dozens of graph-related downstream applications. With people's efforts in graph representation learning research, Graph Neural Networks (GNNs) are shown to be the most powerful tools that can take advantage of both node features and graph structure to learn high-quality representations, and achieve superior performance in tasks such as node classification [55], graph classification [110], and recommender systems [116].

In general, GNNs include two operations: (1) the graph convolutional operation which enables each node to aggregate information from its neighborhood to get a smoothed feature, and (2) the transformation operation which feeds the smoothed features into MLPs to project them into a good embedding space. Compared to other neural networks, the key factor that empowers GNNs is the graph convolutional operation [111]. The Graph Convolutional Filter (GCF) acts as a feature extractor in

the graph convolutional operation, which takes features from immediate or multi-hop neighbors as the input. Thus, great efforts are paid to the GCF design, but how to select suitable GCFs that can be adaptive to the given task remains an open problem.

To select a proper GCF, the first question is what would be the **proper scope of the neighborhood** to consider. Dozens of GCFs are designed to aggregate immediate neighbors' (i.e. 1-hop) information [55, 95]. For these designs, each node receives information from t-hops when t-layers of GNN layers are stacked, which assume the homophily of the graph structure and always apply non-linear transformation in each layer. Some work [107, 56, 31, 133] extend the neighborhood and enable the central nodes to aggregate information from multi-hop neighbors in each network layer, and some spectral-based GNNs are designed to obtain further neighborhood's information by utilizing spectral properties [25]. These works have shown great improvement, which encourages us to consider different hops of neighbors instead of only focusing on immediate neighbors when aggregating neighborhood information.

The next question is how to **put weight on neighbors** within the scope. The GCFs usually can be viewed as a transformed Graph Laplacian[1], and they control the weight of each neighbor node when aggregating neighbor information. The early work usually uses a fixed symmetrically normalized Graph Laplacian to design GCFs [55, 107], which is not flexible and would hinder the network to identify important neighbors for each node. Later work realizes the importance of a flexible GCF which allows the network to learn to adaptively aggregate neighbor information [95, 102]. However, this would enlarge the filter space and can cause overfitting [98]. Hence, it

[1]Please note that Graph Laplacian and Laplacian Score are different. Graph Laplacian is a matrix form of the negative discrete Laplace operator on a graph, while the Laplacian Score is a scalar that acts as a feature selection metric.

is critical to allow flexibility in neighbor aggregation while alleviating overfitting for GCF design.

The discussions surrounding these two questions provide valuable insights and inspiration for considering a larger filter space and constructing it effectively. However, this naturally leads to another important question: how can we **explore this expanded filter space** and identify filters that are specifically tailored to our current task, rather than being agnostic?

In this work, with the aim of answering this question, we present the **Ada**ptive **F**ilter **S**election Graph Neural Networks (AdaFS) to address two challenges in the GCF selection: (1) defining a criterion to establish a strong base filter set, and (2) adaptively selecting filters for the downstream task. The AdaFS provides a framework for graph representation learning with adaptive GCF soft selection. To give a concrete instantiation, for the base filter family, we incorporate filters from four categories and their combinations in a higher-order receptive field: (1) a random walk-based GCF that controls the amount of information received by each node, (2) a reverse random walk-based GCF that controls the amount of information sent by each node, (3) a symmetric Graph Laplacian-based GCF that tries to balance both sides, and (4) an identity matrix-based GCF which aims to keep the raw input. To address the second challenge, we introduce a Laplacian score-based regularization term as an auxiliary training objective, which is able to evaluate the quality of embeddings extracted by GCF in an unsupervised manner. Besides, AdaFS takes advantage of the decoupled graph convolutional and transformation operations, and thus enjoys better efficiency than traditional GNNs. Additionally, we connect this multiple filter learning process to the well-established multiple kernel learning problem, justifying the filter selection rationale. Our main contributions are summarized as follows:

- **Framework**: Propose the adaptive AdaFS framework that learns a soft selection of GCFs with extended neighbor scope and diverse Graph Laplacians.
- **Rationale**: Connect filter selection and kernel selection to justify the rationale behind filter selection.
- **Regularizer**: Propose an unsupervised objective using Laplacian score, which assists selecting appropriate filter with the expanded filter space.
- **Experiments**: Evaluate AdaFS on 9 datasets, conduct thorough ablation studies to validate the importance of key components, and demonstrate its adaptability in consistently producing competitive results.

3.2 Preliminaries

In this work, the proposed AdaFS is a filter selection-based GNN framework, we provide concrete instantiation, and take the semi-supervised node classification (NC) task for graph data as an example task. Thus in this section, we provide the necessary preliminaries regarding the task and the GNNs with diverse GCFs.

3.2.1 GNNs for Semi-Supervised NC

We denote an undirected attributed graph by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the node set and $\mathcal{E}$ is the edge set. Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ be the adjacency matrix for $\mathcal{G}$, let $\boldsymbol{X} \in \mathbb{R}^{n \times d_f}$ be the feature matrix, where row $\boldsymbol{X}_v$ is the feature for node $v \in \mathcal{V}$. For semi-supervised NC, let $\boldsymbol{Y}$ be the class assignment vector, C be the number of classes, and $\boldsymbol{Y}_v \in \{1, \cdots, C\}$ be the class that node v belongs to. Then, the goal is to learn a mapping function $f(\boldsymbol{A}, \boldsymbol{X}) : \mathcal{V} \to \{1, \cdots, C\}$ to predict the class labels for the unlabeled nodes, i.e., $\hat{\boldsymbol{Y}}_v = f(v)$.

GNNs are shown to be the most promising technique so far to solve the semi-supervised node classification task. It includes 2-step operations and usually can fit into a unified message-passing framework for layer l: $\boldsymbol{H}^{(l)} = \sigma(\mathcal{F}(\mathcal{G})\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l)})$, where $\boldsymbol{H}^{(l)}$ is the node representation for $l-$th layer, $\mathcal{F}(\mathcal{G})$ is the GCF for $\mathcal{G}$, $\boldsymbol{W}^{(l)}$ is the learnable mapping parameter. This formula describes the graph convolutional operation with $\mathcal{F}(\mathcal{G})\boldsymbol{H}^{(l-1)}$, and the transformation operation with $\sigma(\cdot\boldsymbol{W}^{(l)})$. Various GNN architectures have been proposed with their own GCF designs, in the next subsection, we take a close look at existing designs.

3.2.2 Graph Convolutional Filters

In the previous section, we highlighted the importance of selecting an appropriate Graph Convolutional Filter (GCF) by considering both the ***proper scope of the neighborhood*** and the ***weighting scheme for neighbors*** within that scope. Existing designs encompass GCFs that consider immediate neighbors' (1-hop) information [55, 95], as well as those that incorporate extended multi-hop neighbors [107, 56, 31, 133]. Weighting schemes also vary, including fixed ones [55, 107] and learnable ones [95, 102]. In the following, we categorize GCFs based on their neighborhood scope and flexibility. We introduce GCFs in each category and analyze their pros and cons.

Fixed and Immediate Neighborhood. The early GNNs obtain information from immediate neighbors and aggregate the information with a fixed weighting scheme. The work of GCN [55] first adopts the convolutional operation on graphs and uses the symmetrically normalized adjacency matrix as the graph filter. Several studies propose to use a sampling strategy to speed up GNN training [144, 38]), which can

be considered as a sparser version of GCN's filter. Besides, some work noticed the importance to use diverse GCFs based on different propagation rules [26] and tried to integrate them together, but their designed filter is still a fixed data-agnostic combination of GCFs and is only for immediate neighborhood. These models are shown empirically to be less powerful due to their lack of flexibility, in addition, though they can reach further neighborhoods by stacking multiple network layers, a deeper network would not enlarge the filter space and may bring the over-smoothing problem.

Fixed and Higher-order Neighborhood. Some studies aim to expand the neighborhood scope and capture information from a wider receptive field. SGC [107], MixHop [1], and GFNN [73] utilize a higher-order symmetrically normalized adjacency matrix with a predefined exponent as their GCF. SIGN [31] and GFN [19] propose concatenating embeddings obtained with GCFs of different receptive fields. These approaches enable nodes to gather information from distant neighbors without redundant computations. GDC [56] employs the diffusion process and uses heat kernel and personalized PageRank as combination weights to aggregate information from neighbor nodes at different hops. While these models benefit from an expanded receptive field, they require manual adjustment of the neighborhood scope and lack flexibility in the aggregation strategy, which restricts their performance.

Learnable and Immediate Neighborhood. Efforts are also paid to developing flexible models with learnable GCFs. Certain works [111, 21] augment the self-loop skip connection with a learnable parameter. Others [59, 39, 66] design filters as polynomials or rational polynomials of graph Laplacian matrices, incorporating learnable coefficients. GAT [95] introduces an attention mechanism to assign weights to nodes in the neighborhood, serving as a flexible GCF based on parametric attention. HOG-

GCN [99] constructs a learnable homophily degree matrix using label propagation to adjust message passing weights on each edge. GBK-GNN [30] applies two different transformation operations on the neighbor nodes, and proposes a learnable kernel selection gate for combination weights. Despite their good performance obtained due to their adaptiveness, these learnable GCFs exhibit to be either too simple (e.g. GIN [111]) or too complex (e.g. GAT [95]). The simple ones lack adaptiveness, while the complex ones require additional computation and are prone to overfitting on simple graphs. Moreover, they solely focus on the immediate neighborhood, limiting the information received within each network layer.

Learnable and Higher-order Neighborhood. Recently, there have been attempts to propose GCFs with both adaptiveness and large receptive fields. The work of ADC [133] is built on top of the GDC [56], but enables the diffusion process to have a learnable neighborhood ratio. While it offers flexibility without hand-tuned parameters, it requires optimizing the validation set to address overfitting. FSGNN [72] proposes a learnable weighted concatenation of embedding matrices obtained with different GCFs. Although it achieves good performance, it lacks a systematic interpretation.

3.3 Method

Our filter selection-based AdaFS framework aims to be capable of learning a data-specific soft-selection of a set of GCFs $\mathcal{F} = \{F_0, F_1 ... F_m\}$ with diverse neighbor scopes and weighting schemes. The overview of the AdaFS is shown in Figure 3.1. In this Figure, each F_i is a GCF computed based on $\boldsymbol{A}$, each ϕ_i is a normalized output of a filter-specific MLP with the aggregated feature $F_i\boldsymbol{X}$. The α_is are the trainable

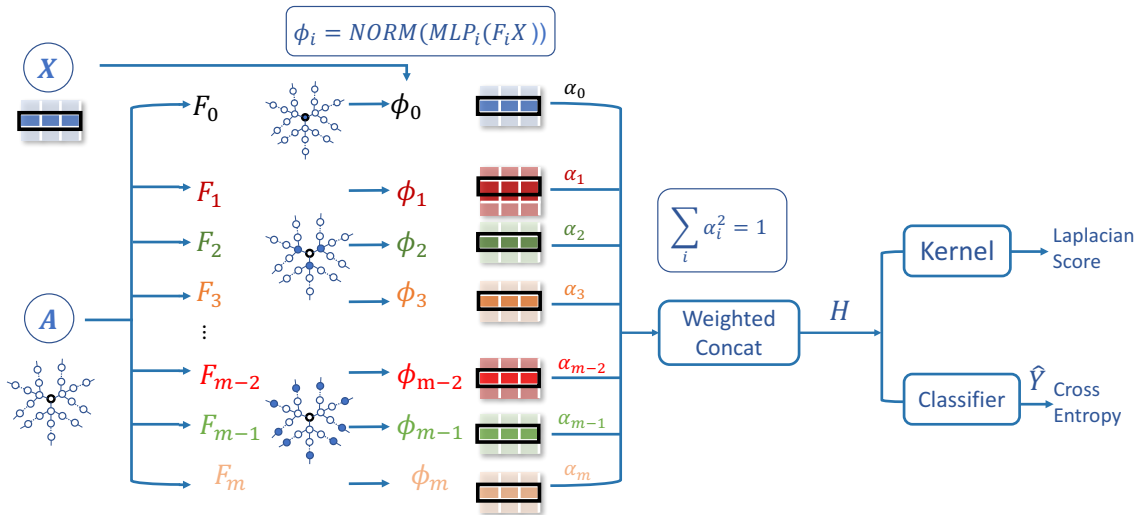


Figure 3.1: Overview of the AdaFS Framework.

weights for the weighted concat module, which would help to identify the importance of the corresponding ϕ_i. With our implementation, we set F_0 to be the identity matrix, $F_{3j+1}, F_{3j+2}, F_{3j+3}$ to be the filters with three different aggregation strategies but all can reach $(j+1)$-hop neighbors. The formal form of the learned embedding can be represented as:

$$\boldsymbol{H} = [\alpha_0\phi_0, \alpha_1\phi_1, \cdots, \alpha_m\phi_m] \tag{3.1}$$

where ϕ_i is the hidden embedding obtained with the i-th GCF F_i in the filter set, and α_i is the concatenation weight. The AdaFS should be trained with the downstream classification task as well as the additional objective for assisting learning with multiple GCFs and addressing the overfitting concerns brought by the expanded filter space. In the following, we formally present the AdaFS framework. We start by discussing how to prepare the GCF set $\mathcal{F}$ by providing a concrete example. Next,

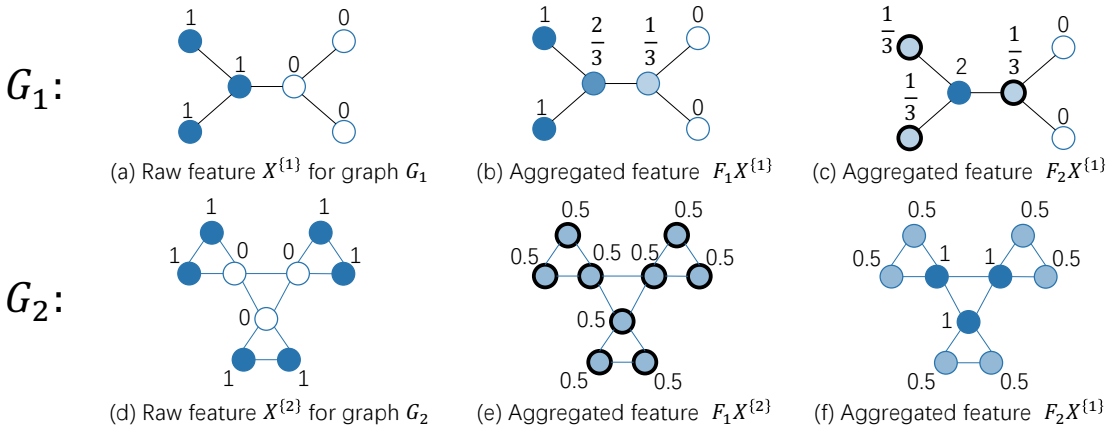


Figure 3.2: Toy Examples to Illustrate Different Graphs Prefers Different GCF.

we establish the rationale behind our design by connecting it to the well-developed multiple kernel learning problem. Then we introduce the Laplacian score as an auxiliary unsupervised objective for training. Finally, we wrap up with discussions on AdaFS's training, advantages, and complexity.

3.3.1 Constructing the GCF Set

Based on the preceding discussion, a GCF set comprising filters with varying neighborhood scopes and diverse weighting schemes is pivotal for a powerful GNN. Building upon this insight, we present an example GCF set for AdaFS, serving as a strong instantiation.

We begin by exploring basic neighborhood weighting schemes utilizing different Graph Laplacian transformations, as they can be easily applied to any given neighbor scope. The following four matrices are considered: $F_0 = \boldsymbol{I}$, $F_1 = \boldsymbol{D}^{-1}\boldsymbol{A}$, $F_2 = \boldsymbol{A}\boldsymbol{D}^{-1}$,

and $F_3 = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, where $\boldsymbol{D} = diag(\sum_j \boldsymbol{A}_{ij})$ is the degree matrix. Among them, the identity matrix F_0 preserves the raw input feature. F_1 employs a random walk-based Laplacian to average the features of a node's neighbors, ensuring every node's new aggregated feature is in the same range. F_2 uses reverse random walk-based Laplacian whose information propagation idea is similar to the PageRank algorithm, distributing the same amount of information to immediate neighbors. It considers node degree during aggregation, which can be beneficial for the case in which node degree plays an important role in classification. F_3 uses symmetric graph Laplacian, combining aspects of both F_1 and F_2. Figure 3.2 gives two toy examples of binary classification in which the graph favors F_1 and F_2 respectively, showing the necessity of considering different Graph Laplacian. Both graphs assume nodes with the same raw feature on them are in the same class. For G_1, after applying F_1 as shown in (b), two classes are still distinguishable, but after applying the F_2 as shown in (c), the three nodes with a bold outline are not separable, thus F_1 is better than F_2 for this case. For G_2, after applying F_1 as shown in (e), all the nodes get non-distinguishable, while applying F_2 is still separable as shown in (f).

Various empirical research outcomes [143, 31, 72, 18, 56] point out the importance of considering a neighborhood of different scopes to ensure obtaining richer neighborhood information at one network layer, and this finding inspires us to explore GCF candidates that can reach $<= t$-hop neighbors with previous base GCF F_1, F_2, and F_3 when given maximum hop number t. For example, when $t = 3$, potentially we can explore filters such as $F_1F_2F_3$, F_2^2, and F_3 . To give a concrete example, in our current implementation, for each hop number j, we select F_1^j, F_2^j, F_3^j from the 3^j possible combinations to construct the GCF set. To put it more clearly, given maximum hop t, we construct the GCF set consisting of the following $3t + 1$ filters

$\{F_0, F_1, ..., F_m\}$, where $m = 3t$ and $F_{3j+i} = F_i^{j+1}$ for $i \in \{1, 2, 3\}$.

As for the theory side, existing works [107, 31, 19] and [26, 102] have provided analysis for the necessity of combining GCFs of diverse neighbor scope and weighting scheme in this selected filter set respectively. We formally organize two claims for the sake of answering the question of why using this GCF set example, and justify them in the following.

- **Claim 1**: From a spectral perspective, GCFs with different Graph Laplacian would not include additional information as they share the same eigenvector space, but provide different coordinate systems to enable GNNs to learn good representations easier.
- **Claim 2**: From a signal processing perspective, different neighborhood scopes would bring us filters for graph signals with different frequencies.

To justify Claim 1, let $\boldsymbol{L}_1 = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A}$, $\boldsymbol{L}_2 = \boldsymbol{I} - \boldsymbol{A}\boldsymbol{D}^{-1}$, $\boldsymbol{L}_3 = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ be the corresponding graph Laplacian for GCF F_1, F_2, F_3 respectively. Let the eigen-decomposition for the symmetric L_3 be: $\boldsymbol{L}_3 = \boldsymbol{U}_3\Lambda_3\boldsymbol{U}_3^\top$, where $\Lambda_3 = diag(\lambda_3^{(1)}, ...\lambda_3^{(n)})$ is a diagonal matrix consists of $\boldsymbol{L}_3$'s eigenvalues, and $\boldsymbol{U}_3 = [\boldsymbol{U}_3^{(1)}, ..., \boldsymbol{U}_3^{(n)}]$ is a matrix whose columns are the corresponding eigenvectors of $\boldsymbol{L}_3$ and satisfies $\boldsymbol{U}_3\boldsymbol{U}_3^\top = \boldsymbol{U}_{(\cdot)}^\top\boldsymbol{U}_{(\cdot)} = \boldsymbol{I}$. Then, $\forall\ \lambda_3^{(i)},\ \boldsymbol{U}_3^{(i)}$:

$$
\begin{aligned}
&\boldsymbol{L}_3\boldsymbol{U}_3^{(i)} = \lambda_3^{(i)}\boldsymbol{U}_3^{(i)} \\
\Rightarrow &\begin{cases} (\boldsymbol{D}^{-1/2})\boldsymbol{L}_3(\boldsymbol{D}^{1/2}\boldsymbol{D}^{-1/2})\boldsymbol{U}_3^{(i)} = \lambda_3^{(i)}\boldsymbol{D}^{-1/2}\boldsymbol{U}_3^{(i)} \\ (\boldsymbol{D}^{1/2})\boldsymbol{L}_3(\boldsymbol{D}^{-1/2}\boldsymbol{D}^{1/2})\boldsymbol{U}_3^{(i)} = \lambda_3^{(i)}\boldsymbol{D}^{1/2}\boldsymbol{U}_3^{(i)} \end{cases} \\
\Rightarrow &\begin{cases} \boldsymbol{L}_1(\boldsymbol{D}^{-1/2}\boldsymbol{U}_3^{(i)}) = \lambda_3^{(i)}(\boldsymbol{D}^{-1/2}\boldsymbol{U}_3^{(i)}) \\ \boldsymbol{L}_2(\boldsymbol{D}^{1/2}\boldsymbol{U}_3^{(i)}) = \lambda_3^{(i)}(\boldsymbol{D}^{1/2}\boldsymbol{U}_3^{(i)}) \end{cases}
\end{aligned}
$$

which indicates $\boldsymbol{L}_1$, $\boldsymbol{L}_2$, and $\boldsymbol{L}_3$ share the same eigenvalues and their eigenvectors can be expressed as $\boldsymbol{U}_1^{(i)} = \boldsymbol{D}^{-1/2}\boldsymbol{U}_3^{(i)}$, and $\boldsymbol{U}_2^{(i)} = \boldsymbol{D}^{1/2}\boldsymbol{U}_3^{(i)}$. Since $\boldsymbol{D}^{-1}$ is a non-singular matrix, we know that $\boldsymbol{L}_1$, $\boldsymbol{L}_2$, and $\boldsymbol{L}_3$'s eigenvectors share the same vector space, and thus $F1, F2, F3$ would not include additional information but would provide different coordinate systems and enable the GNNs to learn more powerful representations.

To justify Claim 2, following the work of [73, 74], taking F_3 as an example, we know that the message passing step $(F_3)^k\boldsymbol{X}$ in GNNs can be reformed as:

$$
\begin{aligned}
(F_3)^k\boldsymbol{X} &= (\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2})^k\boldsymbol{X} = (\boldsymbol{I} - \boldsymbol{L}_3)^k\boldsymbol{X} && (3.2)\\
&= (\boldsymbol{I} - \boldsymbol{U}_3\lambda_3^{(i)}\boldsymbol{U}_3^\top)^k\boldsymbol{X} \\
&= (\boldsymbol{U}_3\boldsymbol{U}_3^\top - \boldsymbol{U}_3\lambda_3^{(i)}\boldsymbol{U}_3^\top)^k\boldsymbol{X} && (3.3)\\
&= \big(\boldsymbol{U}_3(\boldsymbol{I} - \lambda_3^{(i)})\boldsymbol{U}_3^\top\big)^k\boldsymbol{X} \\
&= \big(\boldsymbol{U}_3(\boldsymbol{I} - \lambda_3^{(i)})\boldsymbol{U}_3^\top\big)...\big(\boldsymbol{U}_3(\boldsymbol{I} - \lambda_3^{(i)})\boldsymbol{U}_3^\top\big)\boldsymbol{X} \\
&= \boldsymbol{U}_3(\boldsymbol{I} - \lambda_3^{(i)})^k\boldsymbol{U}_3^\top\boldsymbol{X}
\end{aligned}
$$

From the graph signal perspective, the eigenvalues correspond to the frequencies of the graph signals. Therefore, graph filters with different exponents correspond to the eigenvalues $(1 - \lambda_3^{(i)})^k$ can give us different frequencies and thus can bring us additional information from the spectral perspective.

3.3.2 Filter Selection: A Kernel Perspective

With the GCF set $\{F_0, F_1, ..F_m\}$ in Subsection 3.3.1, we employ filter soft-selection to combine them, effectively determining the importance of each filter. Before delving into how to do the soft selection, let's first establish a connection between our filter

combination and the extensively studied Multiple Kernel Learning (MKL) problem [**?**], which operates on the principle of not relying solely on one kernel, but rather explores a predefined set of kernels $\mathcal{K} = \{K_1, K_2, \ldots\}$ by designing algorithms to perform the soft selection and learn a powerful composite kernel $K = \text{COMBINE}(\mathcal{K})$.

Let's associate each GCF F_i with a feature mapping $\phi_i(\boldsymbol{X}) : \mathbb{R}^{d_f} \to \mathbb{R}^{d_i}$ that generates filter-specific embeddings. Each mapping function corresponds to a kernel $K_i : \mathbb{R}^{d_f} \times \mathbb{R}^{d_f} \to \mathbb{R}$ such that for any pair of nodes u and v, $K_i(\boldsymbol{X}_v, \boldsymbol{X}_u) = \langle \phi_i(\boldsymbol{X}_u), \phi_i(\boldsymbol{X}_v) \rangle$. The Kernel function K_i serves as a similarity measure between pairs of instances from the kernel perspective. Then, for the filter learning, let's consider the mapping function ϕ which combines the filter-specific embeddings ϕ_i with the weighted concatenation combinator: $\phi = [\alpha_0\phi_0, \alpha_1\phi_1, ...\alpha_m\phi_m]$, where each α_i can reflect the importance of its corresponding filter F_i and can act as a soft-selection score. Let K be ϕ's kernel function, then we find:

$$
\begin{aligned}
K(\boldsymbol{X}_u, \boldsymbol{X}_v) &= \langle \phi(\boldsymbol{X}_u), \phi(\boldsymbol{X}_v) \rangle \\
&= \sum_{i=0}^{m} \alpha_i^2 \langle \phi_i(\boldsymbol{X}_u), \phi_i(\boldsymbol{X}_v) \rangle \\
&= \sum_{i=0}^{m} \alpha_i^2 K_i(\boldsymbol{X}_u, \boldsymbol{X}_v)
\end{aligned}
$$

This observation can be summarized as follows: Multiple filter selection using weighted concatenation is analogous to multiple kernel selection using the linear combination. This connection seamlessly bridges the gap between learning multiple filters and the well-established MKL problem, and offers support to justify the rationale behind performing filter soft-selection.

3.3.3 Laplacian Score-based Filter Selection

To mitigate potential overfitting issues resulting from the expanded filter space, it is crucial to incorporate an auxiliary unsupervised objective to enable the algorithm to acquire a generalized model that is not overly biased towards the training set and learn a higher-quality embedding. As we want the auxiliary training objective to be capable of evaluating the representations obtained by the selected filter, it is worth exploring the feature selection metrics, which enable us to assess the quality of representations.

Laplacian score [41] is a popular feature selection metric. It selects features that can best preserve the data manifold structure [63] and can be applied in an unsupervised setting. The basic idea is to use the features for the input instances to construct a k-Nearest Neighbor graph with the weighted adjacency matrix $\boldsymbol{A}_{knn} \in \mathbb{R}^{n \times n}$, then construct the corresponding degree matrix $\boldsymbol{D}_{knn}$ and Laplacian matrix $\boldsymbol{L}_{knn} = \boldsymbol{D}_{knn} - \boldsymbol{A}_{knn}$. Finally, the Laplacian score for feature channel $f_r \in \mathbb{R}^N$ is computed as:

$$L_r = \frac{\tilde{f}_r^T \boldsymbol{L}_{knn} \tilde{f}_r}{\tilde{f}_r^T \boldsymbol{D}_{knn} \tilde{f}_r} \tag{3.4}$$

where $\tilde{f}_r = fr - \frac{f_r^T D \mathbf{1}}{\mathbf{1}^T D \mathbf{1}} \mathbf{1}$ is the centralized f_r and $\mathbf{1}$ indicates all-one vector. In general, a smaller Laplacian score indicates a stronger locality-preserving power, and therefore we would select features with a smaller L_r.

As the Laplacian score has shown its effectiveness in identifying features that capture the underlying manifold structure, in our work, we leverage the Laplacian score within the graph learning context as an example auxiliary objective to guide the training process and mitigate overfitting resulting from the expanded filter

space. Specifically, we utilize the Laplacian score to assess the quality of the learned embeddings produced by our AdaFS. To compute the Laplacian score, we start with the learned graph embedding matrix $\boldsymbol{H}$. By employing cosine similarity, we compute the graph kernel $K(\boldsymbol{H})$. Subsequently, based on $K(\boldsymbol{H})$, we construct $\boldsymbol{A}_{knn}$ by preserving the edges between instance pairs (i, j) if and only if i is among the top-k most similar instances to j. Following the aforementioned steps, we compute the Laplacian score (as defined in Eq.3.4) for each feature channel $f_r = \boldsymbol{H}_{:,r}$. The auxiliary unsupervised training objective ℓ_{lap} is obtained by summing up all L_r values. We aim to minimize ℓ_{lap} during training.

3.3.4 Learning Representations with AdaFS

Other Details. Under the AdaFS framework in Figure 3.1, with the example GCF set in Subsection 3.3.1, taking the adopted Laplacian score for the unsupervised regularization, we already have a clear workflow. In the following, we complete the remaining details for AdaFS instantiation. We set $\phi_i = \text{NORM}(\text{MLP}_i(F_i\boldsymbol{X}))$, where NORM is the row-wise L2-normalization for the sake of scaling each row of the output of the MLP to a unit sphere. We use 1-layer MLPs here instead of a simple linear transformation in order to inject non-linearity. Different MLP_i have different parameters, giving each ϕ_i the flexibility to learn the parameters that is the best for the corresponding filter. Besides, with Subsection 3.3.2, we use the weighted concatenator to combine the ϕ_is with the formula Eq.3.1. We restrict the learnable weights α_is with $\sum_i \alpha_i^2 = 1$ to make it identical to the constraint in the multiple kernel soft-selection.

AdaFS Training. We train the AdaFS model by jointly optimizing the cross-

entropy loss ℓ_{CE} and the Laplacian loss ℓ_{Lap}. As for ℓ_{CE}, we pass the embedding $\boldsymbol{H}$ in Eq.3.1 through a classifier consisting of a linear projection and a softmax to get the predicted labels $\hat{Y} = \text{Softmax}(\boldsymbol{HW})$, then compute the cross entropy value as $\ell_{CE} = -\sum_{v \in \mathcal{V}_{train}} \hat{Y}_v log(Y_v)$. As for the Laplacian loss, we follow Subsection 3.3.3. Finally, the overall optimization objective is: $\ell = \lambda \ell_{CE} + (1 - \lambda)\ell_{Lap}$ with hyper-parameter $\lambda \in [0, 1]$.

3.3.5 Discussion on AdaFS

Advantages of AdaFS. The proposed AdaFS offers several advantages. (1) By utilizing an enriched filter set, AdaFS can explore a larger filter space encompassing various aggregation strategies. This flexibility allows AdaFS to effectively handle graphs with diverse properties (e.g. different levels of homophily). (2) The filter soft-selection scheme in AdaFS enables adaptive identification of the importance of each filter non-agnostic to the graph and the task, while its connection to kernel selection strengthens its justifiability. Besides, the success of the Laplacian Score in assisting the filter selection inspires us that, as a successful filter selection will lead to a more powerful aggregated feature, various feature selection metrics might be applied to boost the filter selection process. (3) Since each filter F_i is fixed, the graph convolutional operation $F_i\boldsymbol{X}$ can be performed as a one-time procedure during pre-processing. This significantly reduces computational costs and improves the scalability of the model.

Comparison to Existing Works. To distinguish AdaFS from the baselines SIGN and FSGNN, which all shared the idea of decoupling the convolutional and transformation operation in GNNs and taking various GCFs into consideration. We highlight

the following points: **(1)** AdaFS is not a single model but a framework designed to leverage diverse GCFs while maintaining flexibility. SIGN and FSGNN also align with this idea and can be seen as specific instantiations within the AdaFS framework. **(2)** AdaFS introduces the Laplacian loss as an auxiliary objective, which evaluates the quality of learned embeddings and guides GCF soft-selection. This enhances model stability, as demonstrated in our ablation study. **(3)** Even only considering the AdaFS instantiation, it has a broader family of GCFs, allowing for a more powerful filter design. In contrast, SIGN only considers different neighborhood scopes, while FSGNN considers cases with and without self-loops in addition, both lack consideration for GCFs with diverse weighting schemes.

Complexity Analysis of AdaFS. For AdaFS with m filters and graph $\mathcal{G}$ with N nodes, it takes $\mathcal{O}(mNd_fd)$ complexity, where d_f is the input feature dim, and d is the output dim. For Laplacian loss, according to the work of [**?**], the KNN graph construction takes $\mathcal{O}(N^{1.14}d)$ complexity, therefore, AdaFS trained with the Laplacian loss takes $\mathcal{O}(mNd_fd + N^{1.14}d)$ complexity in total. Comparing to the classic GCN whose complexity is $\mathcal{O}(L|\mathcal{E}|d_f + LNd_fd)$ where $L = \frac{1}{3}m$ is the layer number and $|\mathcal{E}|$ is the total edge number in $\mathcal{G}$, we know the regular training for our AdaFS instantiation is more efficient than GCN, but the Laplacian loss may slightly sacrifice the running time for performance gain. In addition, other adaptive methods such as GAT usually have unavoidable pair-wise computation on each node pair, and are taking graph convolutional operation every epoch, thus would be less efficient than AdaFS.

Limitations of AdaFS. Although AdaFS has shown great performance, there are still areas where potential improvements can be explored. (1) It would be beneficial to investigate alternative filter designs that can offer additional spectral information beyond providing diverse coordinate systems within the same eigenvector space, as it

could potentially enhance the expressive power and effectiveness of AdaFS. (2) In our current instantiation, we employ a weighted concatenator to combine embeddings obtained with different filters. However, when the number of filters is large, the concatenated embedding matrix $\boldsymbol{H}$ can have high dimensionality, making the filter soft selection more challenging and imposing a higher computational cost. Exploring techniques to address this issue could lead to improvements in both the efficiency and effectiveness of the AdaFS framework.

3.4 Experiments

3.4.1 Datasets, Baselines, and Settings

Datasets. We evaluate the AdaFS on nine benchmark datasets commonly used for node classification. Three of them exhibit homophily: Cora, Citeseer, Pubmed [83]. These datasets are citation networks where nodes represent documents, edges represent citation links, and node features are sparse bag-of-words feature vectors. The class labels correspond to the field of each document. The remaining six datasets exhibit heterophily: Texas, Wisconsin, Cornell [76], Squirrel, Chameleon [80], and Actor [93]. Texas, Wisconsin, and Cornell are web pages of universities where each node is a web page, each edge is a hyperlink, node features are bag-of-words representations, and the categories include courses, faculty, students, projects, or staff. Squirrel and Chameleon consist of English Wikipedia pages related to squirrels and chameleons until December 2018, edges represent hyperlinks, and node features are keywords extracted from the pages. The nodes are classified into five categories based on their average monthly traffic between October 2017 and November 2018. Actor is a graph

Table 3.1: Statistics of Benchmark Datasets. The h-score is the Homophily Score of Graphs Defined in [143].

Dataset	**Cora**	**Citeseer**	**Pubmed**	**Texas**	**Wisconsin**	**Cornell**	**Squirrel**	**Chameleon**	**Actor**
Nodes	2708	3327	19717	183	251	183	5201	2277	7600
Edges	5429	4732	44558	309	499	295	198353	36101	26659
Classes	7	6	3	5	5	5	5	4	5
Feature	1433	3703	500	1703	1703	1703	2089	2325	932
h-score	0.81	0.74	0.80	0.11	0.21	0.30	0.22	0.23	0.22

of actor co-occurrence in Wikipedia pages, node features are the extracted keywords, and the categories are in terms of words of actor's Wikipedia. Table 4.1 provides a summary of the dataset statistics.

Baselines. We compare against the following baseline models. MLP, GCN [55], GAT [95], and GraphSAGE [38] are the classic models that are widely applied in the node classification tasks. Among them, GCN is the first work that adapts the convolutional operation to graphs; GAT adds an attention module to GCN and enables a learnable feature aggregation scheme; GraphSAGE introduces randomness to the GCN model by a node-wise neighborhood sampler module and can work under both inductive and transductive settings. GCNII [18], MixHop [1], H2GC [143], SIGN [31], FSGNN [72], and GBK-GNN [30] are regarded as the state-of-the-art GNN models, most of them considers higher-order neighborhoods. GCNII proposes to add initial residual and identity mapping between layers to make use of the output of each layer. MixHop proposes to concatenate the representation obtained by different hops of the neighborhood. H2GCN also concats the representation of different hops but utilizes the ego- and neighbor-embedding separation to focus on handling heterophily graphs. SIGN aims to improve the model scalability by decoupling the graph convolutional

operation and the transformation operation. FSGNN further improves SIGN by making the aggregation coefficient for different hop numbers learnable and exploring comprehensive experimental settings. GBK-GNN only considers immediate neighbors at a layer but provides different transformations for homophily and heterophily neighbors. Though there can be a long list of related baselines, we select the aforementioned ones for the following reasons: (1) They are representative works and can cover all types of graph convolutional filters discussed in subsection 2.2; (2) They have either released their public implementations or reported their results on most of the benchmark datasets under the same experimental setting as us.

Settings. Follow the experimental setting in [143], we split each dataset evenly for each class into $48\%, 32\%, 20\%$ for training, validation, and testing respectively. We run 10 splits on each dataset, then report the mean and variance of the performance. As for the baseline models, for a fair comparison, we use their reported results [143, 72] (with 3-hop), which are the performance obtained by their public implementation under the same experimental setting as us. For hyper-parameters, we set the maximum hop number as $t = 3$, set the number of layers as 1 for each MLP_i, set the loss weight $\lambda = 0.5$, and set the k-value in k-Nearest Neighbor graph construction as $k = 5$ (as it is one of the most frequently applied hyper-parameter settings when computing Laplacian Score [41]). We initialize the learnable weight α_is as $1/10$ as we have 10 different filters in total. For the homophily datasets, following the experimental convention, we add self-loop to the adjacency matrix. Our code is run on a Tesla V100 GPU device.

Table 3.2: Performance of AdaFS and Baselines on Node Classification Benchmark Dataset.

	Homophily Graphs (h-score>=0.5)			Heterophily Graphs (h-score<0.5)						Average	
	Cora	Citeseer	Pubmed	Texas	Wisconsin	Cornell	Squirrel	Chameleon	Actor	AVG9	AVG6
MLP	74.75±2.2	72.41±2.1	86.65±0.3	81.89±4.7	85.29±3.6	81.08±6.3	29.68±1.8	46.36±2.5	35.76±0.9	65.99	80.35
GCN	87.28±1.2	76.68±1.6	87.38±0.6	59.46±5.2	59.80±6.9	57.03±4.6	36.89±1.3	59.82±2.5	30.26±0.7	61.62	71.27
GAT	82.68±1.8	75.46±1.7	84.68±0.4	58.38±4.4	55.29±8.7	58.92±3.3	30.62±2.1	54.69±1.9	26.28±1.7	58.55	69.24
GraphSAGE	86.90±1.0	76.04±1.3	88.45±0.5	82.43±6.1	81.18±5.5	75.95±5.0	41.61±0.7	58.73±1.6	34.23±0.9	69.50	81.83
GCNII	88.01±1.3	77.13±1.3	**90.30±0.3**	77.84±5.6	81.57±4.9	76.49±4.3	N/A	62.48±2.7	N/A	N/A	81.89
MixHop	87.61±0.8	76.26±1.3	85.31±0.6	77.84±7.7	75.88±4.9	73.51±6.3	43.80±1.4	60.50±2.5	32.22±2.3	68.10	79.40
H2GCN	86.92±1.3	77.07±1.6	89.40±0.3	84.86±6.7	86.67±4.6	82.16±4.8	36.42±1.8	57.11±1.5	35.86±1.0	70.72	84.51
SIGN	85.47±1.7	76.81±1.5	89.78±0.4	78.37±6.7	81.17±3.6	78.37±5.9	48.20±1.4	64.12±2.0	37.13±0.8	71.05	71.05
GBK-GNN	**88.6±0.4**	**79.1±0.9**	89.11±0.2	81.08±4.8	84.21±4.3	74.27±2.1	N/A	N/A	**38.97±0.9**	N/A	82.76
FSGNN	87.61±1.3	77.17±1.4	89.70±0.4	87.57±4.7	88.24±3.4	**87.30±5.9**	73.86±1.8	78.93±1.0	35.38±0.8	78.39	86.26
AdaFS	87.82±1.1	77.86±1.5	89.82±0.3	**87.83±4.8**	**89.80±3.8**	86.42±6.1	**74.32±1.9**	**79.33±1.2**	37.43±1.1	**78.96**	**86.59**

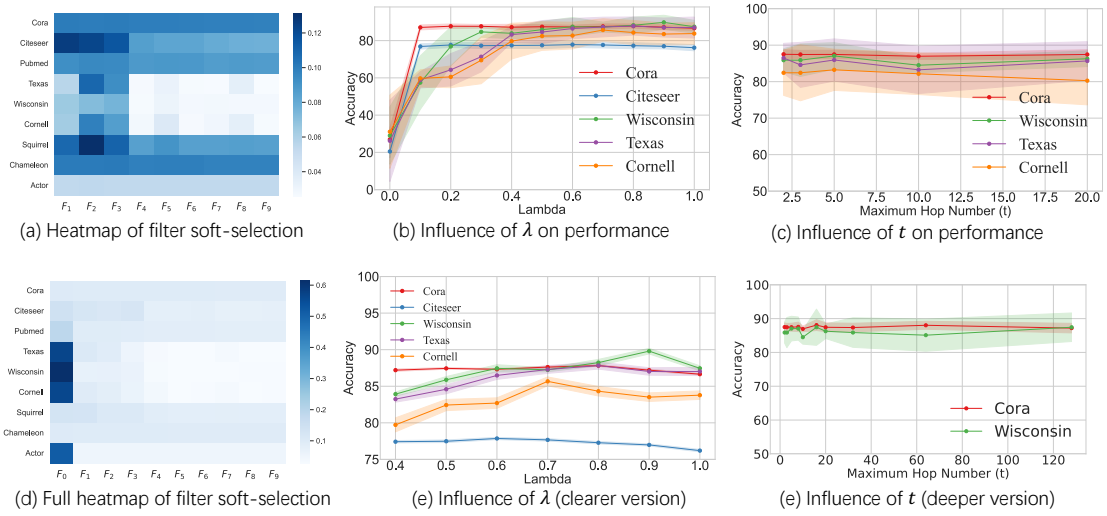
3.4.2 Main Results

In this subsection, we present the results of AdaFS on benchmark datasets. We summarize the classification accuracy of AdaFS and baselines in Table 3.2. Besides, we compute the average accuracy on all the benchmarks (AVG9) and on the first six benchmarks (AVG6, as the other three benchmarks miss results from some baselines). The results show that AdaFS consistently gets competitive test performance.

We observed that, for homophily datasets, all the models can obtain satisfactory performance, and the simple, traditional models such as GCN and GraphSAGE may even outperform the complicated ones, as simple GCFs are powerful enough and their filter space and parameter space are smaller and are less likely to overfit. GCNII achieves the best performance on Cora and Pubmed, and gets the second best on Citeseer, as its skip-connections not only keep a reasonable portion of the initial features, but also reduce unnecessary interaction between feature channels, which would benefit the performance. Our AdaFS is also very competitive and gets the

best on Citeseer, second on Cora, and Pubmed. For heterophily datasets, the classic models GCN, GAT, and GraphSAGE has poor accuracy, as they fail to focus on the raw features that are shown to be the best for the heterophilic graphs. GCNII, H2GCN, FSGNN, and GCNII work better on these graphs as they preserve the raw feature and consider neighbors of different hops. Our model has the best performance on 4 out of 6 heterophilic graphs.

3.4.3 Ablation Study and Discussion



(a) Heatmap of filter soft-selection (b) Influence of λ on performance (c) Influence of t on performance

(d) Full heatmap of filter soft-selection (e) Influence of λ (clearer version) (e) Influence of t (deeper version)

Figure 3.3: Heatmap for the Learned Filter Combination Weight and Curves for Hyper-parameter Tuning.

Soft-selection Results Learned by AdaFS.

We present the learned weights α_is for soft selection in Figure 3.3(a)(d), we omit the F_0 in (a) as it takes a large portion in many datasets and makes the importance score for other filters indistinguishable, but we also want to provide

Table 3.3: Performance of AdaFS and SIGN on Node Classification Benchmark Datasets With and Without the Laplacian Regularizer.

	Homophily Graphs (h-score>=0.5)			Heterophily Graphs (h-score<0.5)					
	Cora	Citeseer	Pubmed	Texas	Wisconsin	Cornell	Squirrel	Chameleon	Actor
SIGN	85.47	76.81	89.78	78.37	81.17	78.37	48.20	64.12	37.13
SIGN-Lap	86.29(↑0.82)	76.72(↓0.09)	89.63(↓0.15)	79.72(↑1.35)	81.90(↑0.73)	80.14(↑1.77)	51.06(↑2.86)	64.49(↑0.37)	36.82(↓0.31)
AdaFS	86.58	76.49	89.55	82.16	87.45	82.97	67.98	78.17	36.88
AdaFS-Lap	87.82(↑1.24)	77.86(↑1.37)	89.82(↑0.27)	87.83(↑5.67)	89.80(↑2.35)	86.42(↑3.45)	74.32(↑6.37)	79.33(↑1.16)	37.43(↑0.55)

the full heatmap including the weights for F_0 in (d). In this figure, *Citeseer* shows a preference for one-hop filters, particularly the random walk-based F_1, while still incorporating information from higher-order filters, this might be because it is a homophilic graph and thus benefits from direct neighbor's information. *Cora, Pubmed, and Chameleon* exhibit balanced importance scores across filters, suggesting that instead of selecting a few important filters, AdaFS learns a more powerful combination of filters that can leverage all available information during the soft-selection process. For *Texas, Wisconsin, Cornell, and Actor*, though exhibiting a slight preference for the 1-hop GCFs, they assign relatively small importance scores to $F_1 \sim F_9$ compared to other graphs, indicating a higher reliance on raw features due to their high level of heterophily. *Squirrel*, despite being heterophilic, benefits from graph structural information and favors the reverse-random walk-based 1-hop filter F_2, as its graph structure may play a non-negligible role for the current node classification task. The heatmap highlights the importance of including different filters to accommodate the preferences of different graphs as well as demonstrates that AdaFS is capable of learning a high-quality GCF non-agnostic to the graph and the task.

Furthermore, to compare the power of different filters and demonstrate the benefit of the soft-filter selection intuitively, we provide visualizations of the aggregated

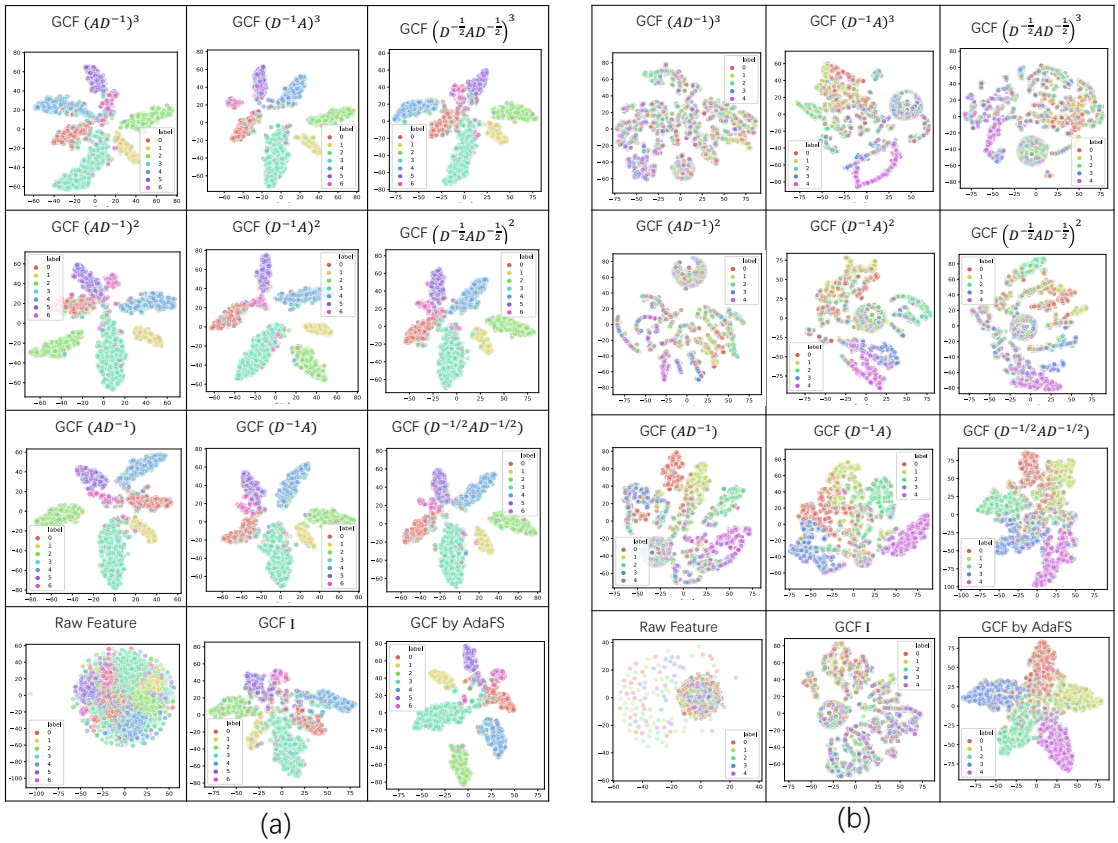


Figure 3.4: Visualization Results with Different GCFs on (a) Cora and (b) Squirrel.

features obtained by AdaFS for Cora and Squirrel in Figure 3.4 (a)(b) respectively. These visualizations show that the GCF obtained by AdaFS always leads to the best performance compared to other GCFs across different datasets, and the soft-selection may even have the potential to learn a combination of base GCFs that is more powerful than simply selecting one of them.

The Role of Laplacian Loss. We investigate the impact of the combination weight λ in the loss function $\ell = \lambda\ell_{CE} + (1-\lambda)\ell_{Lap}$ on performance. We select five benchmark

datasets as examples and vary λ from 0 to 1 in intervals of 0.1. The results are shown in Figure 3.3(b)(e), both sub-figures depict the sensitivity of accuracy to λ, but (b) provides the full view, while (e) focuses on the curves with $\lambda \geq 0.4$ to give a clearer view of the performance gain achieved by incorporating the Laplacian loss (as cases with $\lambda \geq 0.3$ are clearly less competitive compared to the others). We find that, when $\lambda = 0$, corresponding to training the embedding $\boldsymbol{H}$ solely with the Laplacian loss, the performance is poor. This observation is not surprising and matches our intuition that it is important to have a supervised loss to guide both embedding and classifier training. When $\lambda = 1$, corresponding to training the model solely with the cross-entropy loss, the performance is already quite good and outperforms many baselines. However, combining both losses yields even better results. Please note that as the $\lambda = 0$ case is too bad, the benefit of incorporating the Laplacian loss may not be very obvious. To address this, we present Table 3.3 to underscore the significance of the Laplacian loss. In addition, we also consider taking SIGN as the backbone model since it also applies multiple GCFs. For AdaFS, comparing the results with (AdaFS-Lap) and without (AdaFS) ℓ_{Lap}, we find that Laplacian loss improves the classification performance. Comparing the performance gain brought by the Laplacian loss between SIGN and AdaGS, we observe that AdaFS benefits more from the Laplacian loss while SIGN gets less. This might be because the purpose of Laplacian loss is to assist the learning with an enlarged filter space, so for models with a fixed filter such as SIGN, it is less helpful. Besides, we also provide the loss curves in Figure 3.5 (g)(h) for the setting with and without ℓ_{Lap} respectively. We find that AdaFS trained with Laplacian loss is more stable.

Influence of Maximum Hop Number t. We examine the influence of the maximum hop number t on classification accuracy. We select four benchmark datasets

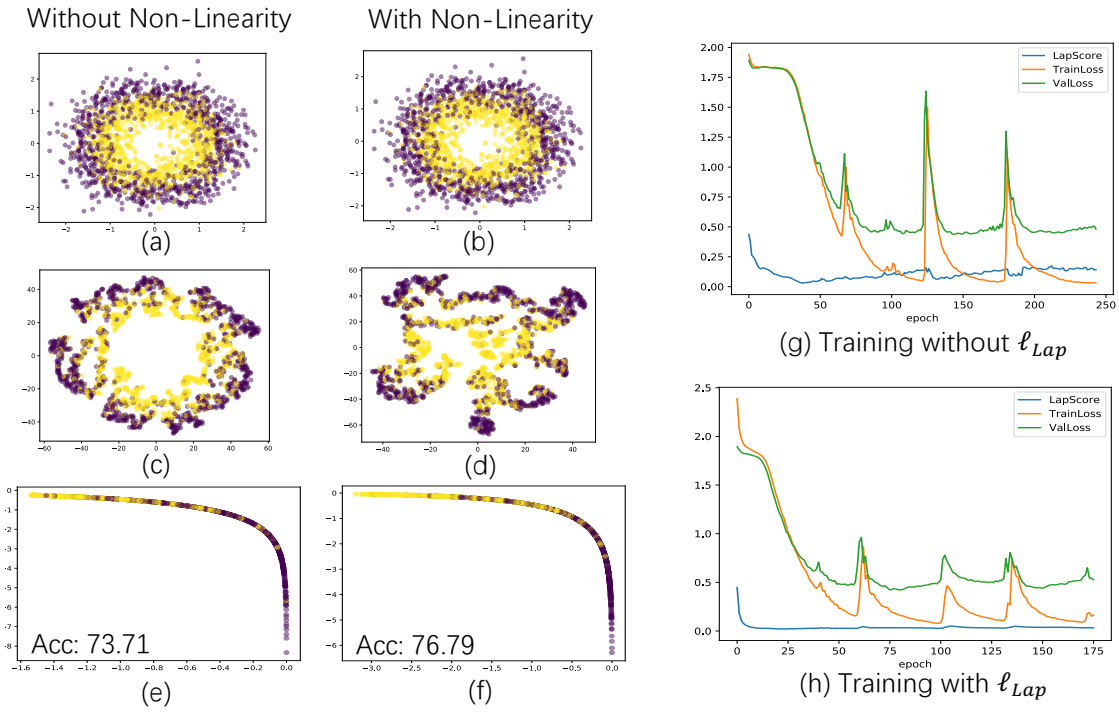


Figure 3.5: Importance of Non-linearity and Importance of the Laplacian Loss.

and vary t from 2 to 20, showing the results in Figure 3.3(c). We observe that, with $t = 20$, where we need to learn a soft selection from 61 filters, AdaFS can still learn an effective combination of filters and achieve competitive performance. The performance does not show a clear difference when varying t from 2 to 20. This demonstrates that AdaFS, trained along with the Laplacian loss, can mitigate the overfitting caused by the increased filter space. Furthermore, we present 3.3(f) as an extended version to investigate AdaFS's performance with a deeper architecture, which presents the results for AdaFS with $t = 2, 3, 5, 8, 10, 16, 20, 32, 64, 128$ on Cora and Wisconsin datasets. We find that, even with a larger t up to 128, we can still observe the identical phenomenon that, when hop number t goes large, AdaFS does not suffer a performance drop, which indicates it can go deeper.

Importance of Including Non-linearity in ϕ_i. We present Figure 3.5 (a)$\sim$(f) to highlight the importance of incorporating non-linearity in the design of ϕ functions. In the figure, (a) and (d) are the same synthetic two-circle graphs with two classes and 1000 nodes for each class. (b) and (e) are the TSNE visualization of $\boldsymbol{H}$ learned by AdaFS with linear mapping as ϕ_i and MLP as ϕ_i respectively. (c) and (f) are the corresponding predicted 2-D label vectors. By comparing the t-SNE visualizations of $\boldsymbol{H}$ learned by AdaFS without non-linearity (Figure 3.5(b)) and with non-linearity (Figure 3.5(e)), we find that, the visualization without non-linearity still exhibits a two-circle structure, and this leads to a lower classification performance, as shown in Figure 3.5(c). In contrast, the visualization that incorporates non-linearity is shown to be capable of making the two classes more separable, and demonstrates a higher classification accuracy, as indicated in Figure 3.5(f). These findings emphasize the necessity of incorporating non-linearity in the design of ϕ_i functions in order to achieve better classification performance.

Results on Running Time. In order to compare the actual running time, we take the dataset Squirrel and Chameleon as examples to compare AdaFS without ℓ_{Lap}, AdaFS, SIGN, and H2GCN. We examine the 3-layer network, and the results are the following: AdaFS without ℓ_{Lap} takes 94.2653s on Squirrel and 62.9830s on Chameleon; AdaFS takes 179.4837s on Squirrel and 84.5602s on Chameleon; SIGN takes 81.4807s on Squirrel and 59.7841s on Chameleon; H2GCN takes 254.s on Squirrel and 90.0227s on Chameleon.

3.5 Conclusion

In this work, we aim to give a comprehensive study of the GCF selection. We begin by highlighting the GCF for GNN should not be data-agnostic and task-agnostic and thus an adaptive filter design should be explored. We argue that exploring an expanded GCF family with diverse filters is crucial, and incorporating an auxiliary training objective would be important for effective filter learning with this enlarged filter space. To address these considerations, we formally present the AdaFS framework for graph representation learning, and instantiate it with a GCF set consisting of GCFs varies in neighbor scopes and weighting schemes. Furthermore, we take the adapted Laplacian score in the instantiation as the regularizer. Besides, we seamlessly build the connection between the mature MKL problem and the filter selection problem, providing theoretical support for AdaFS. Through extensive experiments, AdaFS is shown to becapable of generating appropriate GCFs to accommodate the given task.

CHAPTER 4

SMASH: Scalable Meta-path Aggregation baSed Heterogeneous Graph Neural Networks

4.1 Introduction

Many real-world data take the form of heterogeneous graphs - a very informative graph type that contains various types of nodes and edges and can model complex realistic systems, such as academic graphs and social networks. Heterogeneous Graph Neural Networks are a family of heterogeneous graph embedding models adapted from the GCN kipf2016semi and have shown their great capability by taking the performance of tasks on heterogeneous graphs to the next level.

As a heterogeneous graph has multi-typed nodes and relations, when conducting the message passing on heterogeneous graphs, the question of how to propagate information among different node types via different relation types arises naturally. Based on the perspective to consider this question, most of the existing Heterogeneous GNNs can be categorized into two types: relation-based and metapath-based models.

Relation-based models [82, 121, 123] usually constructs homogeneous subgraphs for each single relation type or for each sub-set of relation types for message passing, then combine the obtained embeddings on each subgraph to get the final

embedding on the full graph. However, it is very challenging for relation subsets to cover all the important meta-paths in the full graph, and therefore, they may suffer inevitable semantic information loss.

Metapath-based methods [101, 33, 45, 125] consider message passing for each important metapath either explicitly or implicitly, then combine the obtained embeddings to get the final embeddings on the full graph. But as the number of metapaths usually goes exponentially to the length of metapath, these models would either suffer scalability challenges, or heavily depend on the domain expertise to pre-select the important metapaths. Besides, in the final combination step for both types of models, we need to carefully consider the importance of each combined component.

With the aim of the aforementioned challenges, we carefully think about how we can design an effective and scalable Heterogeneous GNN model. This leads us towards designing a model with the goal of covering more metapaths to keep richer semantic information while giving the metapaths enough flexibility to make them distinguishable. In this work, We focus on the node classification task on heterogeneous graphs, and propose the SMASH model, which consists of 1) Metapath Feature Propagation step which conducts message passing for matapaths obtained by a Breadth-First-Search (BFS) to avoid semantic information loss; 2) Metapath Feature Aggregation step which is a 3-stage operation to combine the obtained embedding from previous step stage by stage in a hierarchical manner with a heuristic based metapath aggregator, an attention-based metapath aggregator, and a concatenation-based aggregator; 3) Transformation step which projects the representation in a better embedding space with a few MLP layers.

We summarize our main contributions as follows:

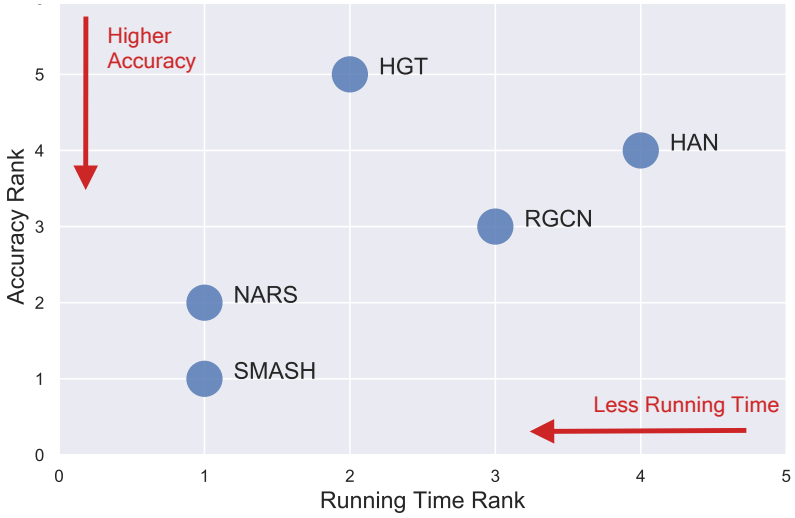


Figure 4.1: Runtime and Accuracy Rank of SMASH and Different HeterHeterogeneous GNNs on ACM Dataset.

- We systematically discuss how to develop powerful metapath-based heterogeneous GNNs by proposing and answering the three fundamental questions regarding metapath aggregation: what metapaths to use, when to aggregate, and how to aggregate.

- We propose a multi-stage metapath aggregation module that can get aware of the metapath importance, and then propose the Scalable Metapath Aggregation-baSed Heterogeneous GNN (SMASH) model.

- We demonstrate the power of SMASH on benchmark datasets. SMASH achieves top-1 performance on ACM, OAG-L1, and OAG-Venue dataset, and also gets competitive results on OGB-MAG. We provide a glance at the high-level comparison in terms of running time and accuracy of our SMASH with existing methods in Figure 4.1.

4.2 Preliminaries

4.2.1 Node Classification for Heterogeneous Graphs

A heterogeneous graph [89] $\mathcal{G}(\mathcal{V}, \mathcal{E}, \Phi, \Psi)$ is a type of directed graph which contains multiple types of nodes and relations which help to represent interconnected data, where $\mathcal{V}$ is a multi-type node set, $\mathcal{E}$ is a multi-type edge set, $\Phi : \mathcal{V} \to \mathcal{A}$ is a mapping function that associates each node v in the node set $\mathcal{V}$ to a node type a in node type set $\mathcal{A}$, and $\Psi : \mathcal{E} \to \mathcal{R}$ is a mapping function that associate each edge e in edge set $\mathcal{E}$ to an relation type r in relation type set $\mathcal{R}$.

Our work focuses on the node classification task on attributed heterogeneous graphs. The task specifies a target node type, and has a set of labeled and unlabeled nodes within this type respectively. The labeled nodes are used as the training supervision to learn a model that predicts the labels for the unlabeled ones. We provide a detailed formal task description as follows:

Given the heterogeneous graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \Phi, \Psi)$, let $\mathbf{X}_{a_i} \in \mathbb{R}^{n_{a_i} \times d}$ be the feature matrix for node type a_i, where n_{a_i} is the number of nodes with type a_i and d is the initial feature dimension (we assume all node types have same initial feature dimension), the v-th row $\mathbf{X}_{a_i}(v)$ is the initial feature vector for node v satisfies $\Phi(v) = a_i$. Let $\mathbf{A}_p \in \mathbb{R}^{n_{a_k} \times n_{a_1}}$ be the adjacency matrix for metapath p starts from a a_1 node type and ends with a a_k node type, where $\mathbf{A}_p(v_k, v_1) = 1$ if and only if there is a metapath instance for metapath p connecting node v_k with node type a_k and node v_1 with node type a_1. Let a^* be the target node type and $\mathcal{V}^*$ be the set of type-a^* nodes, let Y^* be the class assignment vector for all the nodes in $\mathcal{V}^*$, C be the number of classes, then $Y^*(v) \in \{1, \cdots, C\}$ would be the label for node v in $\mathcal{V}^*$.

Let $\mathcal{P}^*$ be the full metapath set end with node type a^*. Then the node classification task is to learn a mapping function $f : \mathcal{V}^* \to \{1, \cdots, C\}$ by leveraging node features $\{\mathbf{X}_{a_i} : a_i \in \mathcal{A}\}$, adjacency matrices $\{\mathbf{A}_p : p \in \mathcal{P}^*\}$ and the labeled nodes, to predict the class labels for the unlabeled nodes.

4.2.2 Heterogeneous Graph Emebedding

Extensive studies have been done on heterogeneous graph embedding [114].

Shallow Methods. Before the deep learning era, various powerful shallow models were proposed. Metapath2vec [29] is a shallow effective model inspired by Deepwalk [77], and its main idea is to first conduct metapath-based random walks and generate a set of random walk sequences, then use heterogeneous skip-gram to obtain the embeddings. Hin2vec [32] is another random-walk-based shallow model, which exploits various types of relations among nodes by conducting multiple prediction tasks jointly. The work PTE [92] is a proximity-based approach that is designed based on the idea that similar nodes' embeddings should be close, it decomposes the heterogeneous graph into multiple bipartite graphs for each relation type and considers the log-likelihoods over all of them. All these shallow methods are shown to be powerful on various real-world benchmark datasets, however, they only utilize the structural information, so they may lose important information from node attributes. In addition, these methods heavily rely on the quality of the graph structure and therefore would be more vulnerable when the graph structure is noisy or irrelevant to the downstream node classification task.

Deep Methods. Graph neural networks (GNNs) have emerged as the state-of-art family of models for graph representation learning and related tasks [55, 110, 116, 82].

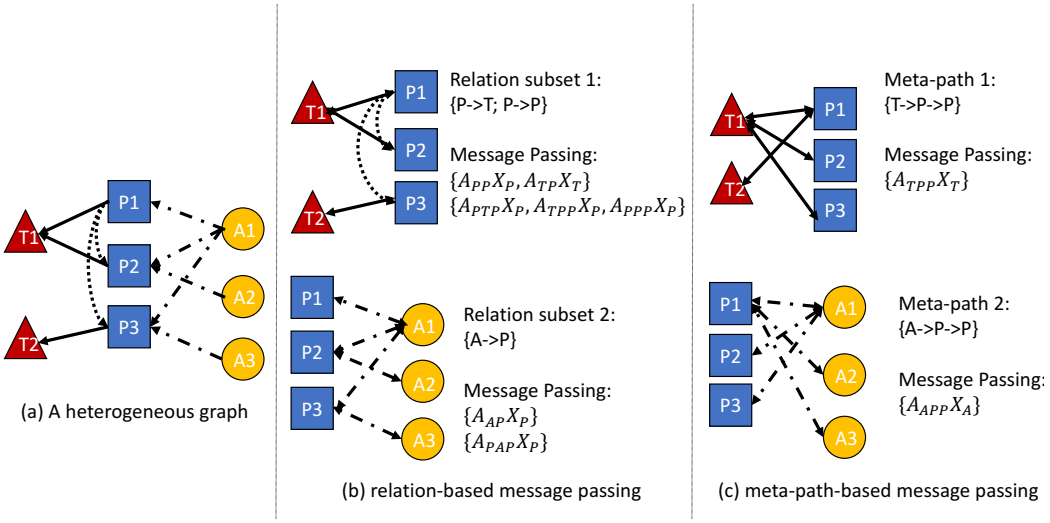


Figure 4.2: An Illustration Example for the Difference Between Relation-based Message Passing and Metapath-based Message Passing.

Different GNN architectures have been proposed [55, 95, 110] based on the message-passing framework to generate powerful embeddings by fusing the node attributes and the graph structural information. In recent years, researchers have made efforts to generalize GNNs to heterogeneous graphs, and developed dozens of powerful heterogeneous GNN architectures. These deep methods can be categorized into two different types in general: relation-based methods and metapath-based methods, based on what perspective the model considers for message passing.

Relation-based models [82, 123, 121] usually construct homogeneous sub-graphs for each relation type or for each relation subsets, then propagate information on the sub-graphs to update the representations for the target nodes type. RGCN [82] is a simple but effective extension of GCN [55], which conducts message passing for each single relation's in-coming and out-going edges respectively, then combines the generated representations using degrees as combination weights to obtain the

representation for one RGCN layer. R-HGNN [121] proposes a cross-relation message passing module to improve the interactions of node representations across different relations after obtaining the representations on each sub-graph and thus would be able to capture more semantics information. However, for complex heterogeneous graphs with a large number of relations, RGCN and R-HGNN scale poorly. NARS [123] introduces neighbor averaging over relation sub-graphs, which randomly samples sub-graphs based on relation subsets and then simply operates on graph-smoothed node features and therefore would be scalable on large-scale graphs. Despite their good effectiveness, the relation-based models may lose some important semantic information in the metapaths.

For example, suppose we have a heterogeneous graph shown in Figure 4.2 (a), whose most important semantic information is encoded in metapath T→P→P and A→P→P, and the target node type is P. Assume we consider two relation-subsets or two metapaths for message passing, and we are considering 2-hop neighborhood. Then, using a relation-based method such as NARS [123] can lead to getting the sampled relation-subsets as shown in (b), which fails to propagate information through A→P→P, one of the key metapaths. Therefore target node P1 would not be able to get important information propagated from node A3 and as a result may fail the downstream node classification task. Constructing the sub-graph using the full relation subset is a way to avert this issue - in fact other relation-based methods like RGCN use all the relations - however, this would mix all the relations together at an early stage and make them undistinguishable, and therefore may include some noise.

Metapath-based methods [101, 45, 33, 125] consider message passing and feature propagation through each individual or each group of metapath(s), then aggregate the propagated features. HAN [101] proposes to utilize metapaths to model higher-

order proximity and introduce attention mechanisms to heterogeneous GNNs. It utilizes a node-level attention module and a semantic-level attention module to produce representations aware of the neighbor importance and metapath importance respectively. MAGNN [33] improves HAN by considering both the metapath-based neighborhood and the nodes along the metapath during the propagation. Though these models are powerful and can identify important metapaths explicitly, however, as the number of possible metapaths usually goes exponentially to the length of metapath, these two methods need a manual selection of metapaths in a pre-processing stage, which requires domain expertise and would bring some challenges. HGT [45] proposes to use each edge's type to parameterize a transformer-like self-attention architecture, so that the metapaths can be learned implicitly and therefore the semantic information can be captured. GTN [125] proposes to composite relations and generate useful multi-hop metapaths implicitly and automatically learn to leverage information from different metapaths. Though these two methods tried to solve the aforementioned problem by implicitly learning to aggregate information from different metapaths, they are not very efficient and do not have good interpretability. Figure 4.2 (c) provides an example of explicit metapath-based message passing, that covers the important metapaths in the scenario, but still relies on manual metapath selection.

4.3 Methodology: SMASH

In the following, we propose our solution: SMASH. It includes 3 steps: 1) Metapath Feature Propagation for message passing on each metapaths; 2) Metapath Feature Aggregation to aggregate the propagated features for the metapaths stage by stage; and 3) Transformation which projects the obtained aggregated features into proper

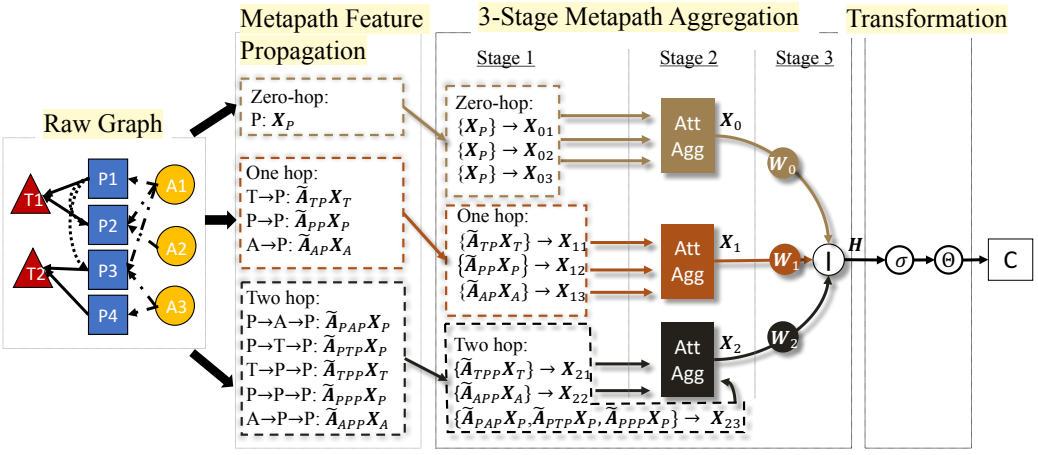


Figure 4.3: Overview of SMASH.

space for the downstream tasks. Figure 4.3 shows its workflow overview, in this example, we set the maximum metapath length to 2, and set the number of metapath subsets to 3 per path length. σ is a non-linear transformation and Θ is an MLP whose layer number is a hyper-parameter in the implementation. C is the classifier for the downstream task, we set C to be the softmax classifier in our implementation.

4.3.1 Details for Components in SMASH

Metapath Feature Propagation To avoid semantic information loss, with a pre-defined length K, we enumerate all the metapaths $\{p : p \in \mathcal{P}^*\}$ that end with the target node type a_* and are of a length less than or equivalent to K by running a Breadth-First-Search (BFS) on the heterogeneous meta graph. Then for each metapath p with source node type a_p, we would have an adjacency matrix $\mathbf{A}_p$ indicating the connectivity between nodes of type a_* and nodes of type a_p, and a feature matrix for nodes of type a_p: $\mathbf{X}_{a_p}$. Then, with the normalized adjacency matrix

$\tilde{\mathbf{A}}_p = \mathbf{D}_p^{-1}\mathbf{A}_p$, where $\mathbf{D}_p^{-1} = diag[1/d_p(v)]$, we conduct message passing via all the metapath instances of metapath p as:

$$\tilde{\mathbf{A}}_p\mathbf{X}_{a_p}(v) = \frac{1}{|\mathcal{N}_{v,p}|}\sum_{u\in\mathcal{N}_{v,p}} X_p(u) \tag{4.1}$$

where $\mathcal{N}_{v,p}$ is the metapath-based neighborhood for node v with metapath p, and $|\mathcal{N}_{v,p}|$ is its size.

Metapath Feature Aggregation. To aggregate the metapath propagated node features $\{\tilde{\mathbf{A}}_p\mathbf{X}_{a_p}|p \in \mathcal{P}^*\}$, we propose a three-stage aggregation module. We use different aggregation operators for different stages, with the purpose of maintaining the representation learning power of the model while avoiding expensive computation costs.

In the first stage, we split metapaths into subsets based on their length and source node type. For each path length $k \in \{1, ..., K\}$, we generate S subsets (so in total there would be $K \times S$ subsets.), and denote the subsets as $\mathcal{P}_{k1}, \mathcal{P}_{k2}, ..., \mathcal{P}_{kS}$. For each metapath p of length k, we randomly put it to a subset $\mathcal{P}_{ks}$, with the constraint that all metapaths in the same subset $\mathcal{P}_{ks}$ should have the same source node type. Next, we generate a feature matrix for each metapath subset $\mathcal{P}_{ks}$ by aggregating the propagated node features $\{\tilde{\mathbf{A}}_p\mathbf{X}_{a_p}|p \in \mathcal{P}_{ks}\}$ according to their degrees, i.e.:

$$\mathbf{X}_{ks}(v) = \sum_{p\in\mathcal{P}_{ks}} \frac{d_p(v)}{\sum_{p_i\in\mathcal{P}_{ks}} d_{p_i}(v)} \cdot (\tilde{\mathbf{A}}_p\mathbf{X}_{a_p})(v) \tag{4.2}$$

where $d_p(v)$ is the number of metapath p reachable neighbors for node v, and $(\tilde{\mathbf{A}}_p\mathbf{X}_{a_p}(v))$ is the corresponding row in $\tilde{\mathbf{A}}_p\mathbf{X}_{a_p}$ for node v.

In the second stage, we design an attention-based aggregator to combine the subset-specific features $\{\mathbf{X}_{ks}|s \in \{1, 2...S\}\}$ and generate path length specific representations, i.e. for each k, our aggregator adaptively learns the importance for each $\mathbf{X}_{ks}$. We propose two aggregators for different scenarios: 1) **Feature-wise attention** aggregator which learns the importance of each feature channel:

$$\mathbf{X}_k(v) = \sum_{s=1}^{S} \alpha_{ks} \odot \mathbf{X}_{ks}(v) \tag{4.3}$$

where $\alpha_{ks} \in R^d$ is a learnable attention vector for subset s of length k metapaths, and $\odot$ is the element-wise multiplication operator. For a specific feature channel, all target nodes share the same attention weight on the metapath subsets; 2) **Node-wise attention** aggregator that learns different metapath subset importance for different nodes:

$$\mathbf{X}_k(v) = \sum_{s=1}^{S} \beta_{ks}(v)\mathbf{X}_{ks}(v) \tag{4.4}$$

where $\beta_{ks} \in R^{n_{a^*}}$ is a learnable vector, and each entry $\beta_{ks}(v) \in \mathbb{R}$ is the attention weight for node v. For a target node, all feature channels share the attention weight for the metapath subset.

In the last stage, we aggregate the path-length specific representations to obtain the target nodes representations that are ready for further transformations. For each $\mathbf{X}_k$, we learn a weight matrix $\mathbf{W}_k$ to which is used to $\mathbf{X}_k\mathbf{W}_k$ Then, we concatenate the to obtain the aggregated node representation:

$$\mathbf{H} = [\mathbf{X}_1\mathbf{W}_1, \mathbf{X}_2\mathbf{W}_2, ...\mathbf{X}_k\mathbf{W}_k] \tag{4.5}$$

Transformation. $\mathbf{H}$ is our metapath augmented node representation and in order to obtain model predictions, we pass $\mathbf{H}$ through a Multi-Layer Perceptron (MLP) Θ to transform the aggregated representations for the downstream node classification task.

Finally, we feed the representations into an MLP classifier, to get the predicted node labels for target nodes. The SMASH model described in this pipeline is trained with the cross-entropy loss.

4.3.2 Advantages of SMASH.

We summarize the advantages of SMASH in effectiveness, scalability, and explainability. For the effectiveness side, our method can prevent information loss as we cover full metapath sets. In addition, SMASH is capable of learning to gather information from diverse metapaths and paying attention to those informative ones with a 3-stage aggregation scheme, which further enhances effectiveness.

For the efficiency side, our method scales well. At the first aggregation stage, we group similar metapaths into subsets with smaller set sizes to address the scaling issue. Then, we aggregate different groups together based on the metapath length with an attention-based sum aggregator. This step-by-step aggregation captures the crucial metapath information while being able to alleviate the pressure of maintaining the distinguishability of metapaths, whose numbers grow exponentially with the metapath length. As for the training complexity, the feature propagation and the

stage 1 aggregation are one-time processes before the training; stage 2 aggregation takes $\mathcal{O}(Kmdn_{a^*})$ time where m is the metapath subset number, and K is the maximum metapath length; stage 3 aggregation takes $\mathcal{O}(Kd^2n_{a^*})$, here, for simplicity, we assume all the hidden layers has dimension d; the transformation module takes $\mathcal{O}(Ld^2a^*)$ where L is the number of MLP layers.

In terms of interpretability, our method is capable of getting aggregated representations that are aware of the importance of metapaths. We learn the importance by adaptive learning how metapath subsets should be aggregated either for each feature columns or for each node. In addition, within GNN each metapath subsets, the SMASH model emphasizes the metapath that connects a target node to the most metapath reachable neighbors. Therefore by considering both the adaptively-learned importance for each metapath subset and the metapath significance inside each group, we would be able to identify the important metapaths.

4.3.3 Distinguishing SMASH with Existing Models.

In the end, we distinguish our SMASH from existing methods. First, we want to highlight all the concerns for Heterogeneous GNN embedding learning. On the one side, we have effectiveness concerns, (1) we don't want to exclude any metapaths at the beginning, as it's hard to decide which metapath is important; also, (2) some of the metapaths are highly correlated, and consider them separately might be harmful to the model performance; in the other side, we have efficiency concern, (3) number of metapaths grow exponentially with metapath length, and some heterogeneous graphs may have a large number of relations. In addition, (4) we want the model to be interpretable instead of just being a powerful "black-box" model. None of the

existing models can satisfy all 4 points.

For the baselines, RGCN would be less efficient with a large number of relations and cannot adaptively learn metapath importance. HAN and MAGNN handle challenge(1) by manually selecting important metapaths that require domain expertise, and their complicated attention mechanism limits their scalability. HGT can only implicitly learn the metapath importance and therefore has less interpretability. NARS may exclude some important metapaths and would suffer the challenge (1). GTN needs to do feature propagation for each training epoch, thus having higher complexities in both time and memory.

In SMASH, we consider a full metapath set to handle (1), we propose grouping-based metapath feature propagation (Stage 1) to address (2) and (3), and by combining our heuristic-based importance (stage 1) within GNN each metapath group and the learnable attention-based importance (stage 2) among each metapath group, we can identify important metapaths and give good interpretations for (4).

4.4 Experiment

4.4.1 Experimental Setup

Datasets. For the experiments, we evaluate the SMASH model on four widely used academic graphs for the node classification task:

- ACM [101] is a citation graph based on ACM papers. The node classification task has paper nodes as the target node to predict its category.
- OGB-MAG [43] is an academic network constructed from Microsoft Academic

Table 4.1: Statistics of Benchmark Datasets for Node Classification in Heterogeneous Graphs.

Dataset	**Nodes**	**Relations**	**Features (init dim → dim, how features are generated)**	**Splits**
ACM	P: 4,025	P-A: 13,407	P: 1903, Bag-of-words features	Random Split
	A: 17,431	P-F: 4,025	A: 128→1903, Randomly projected TransE features	Train: 808
	F: 73		F: 128→1903, Randomly projected TransE features	Val: 201
				Test: 2816
OGB-MAG	P: 736,389	P-A: 7,145,660	P: 128→256, Randomly projected Word2Vec features	Time-based Split
	A: 1,134,649	P-P: 5,416,271	A: 256, TransE features	Train: 629,571
	F: 59,965	P-F: 7,505,078	F: 256, TransE features	Val: 64,879
	I: 8,740	A-I: 1,043,998	I: 256, TransE features	Test: 41,939
OAG-L1	P: 119,483	P-A: 340,959	P: 768, pre-trained XLNet features	Time-based Split
	A: 510,189	P-P: 329,703	A: 400→768, Randomly projected TransE features	Train: 81,071
	V: 6,934	P-V: 119,483	V: 400→768, Randomly projected TransE features	Validation: 16,439
	I: 9,079	A-I: 612,872	I: 400→768, Randomly projected TransE features	Test: 21,973
OAG-Venue	P: 166,065	P-A: 477,676	P: 768, pre-trained XLNet features	Time-based Split
	A: 510,189	P-P: 851,644	A: 400→768, Randomly projected TransE features	Train: 106,058
	F: 45,717	P-F: 1,700,497	F: 400→768, Randomly projected TransE features	Validation: 24,255
	I: 9,079	A-I: 612,872	I: 400→768, Randomly projected TransE features	Test: 35,752

Graph (MAG). The target node type is paper and the node classification task is to predict the venue.

- OAG-L1 and OAG-Venue [45] are both constructed from the largest public academic graph Open Academic Graph (OAG, [85]) by using nodes from the CS domain. They use paper as the target node type and the goal is to predict the L1-field of each paper and the venue of each paper respectively.

For all these four datasets, for the node types that do not have raw features, follow the work of [123], and use the TransE model to generate initial features for them, before projecting the features for all node types to the same dimension. The detailed dataset statistics are summarized in Table 4.1.

Baselines. We compare SMASH against the following two types of baseline

Table 4.2: Performance of SMASH and Baselines on Benchmark Datasets. OOM indicates out of memory.

Dataset	Metric	R-GCN	NARS	HAN	HGT	| SMASH
ACM	Acc	0.930±0.002	0.931±0.004	0.922±0.002	0.919±0.003	| **0.934±0.008**
OGB-MAG	Acc	0.500±0.001	**0.521±0.004**	OOM	0.498±0.001	| 0.507±0.001
OAG-L1	NDCG	0.852±0.002	0.868±0.001	OOM	0.868±0.002	| **0.873 ± 0.001**
	MRR	0.843±0.002	0.857±0.003	OOM	0.849±0.003	| **0.863 ± 0.001**
OAG-Venue	NDCG	0.481±0.004	0.520±0.003	OOM	0.498 ± 0.014	| **0.527±0.007**
	MRR	0.302±0.005	0.342±0.003	OOM	0.322±0.014	| **0.349±0.010**

models: Relation-based models and metapath-based models. We take R-GCN [82] and HAN [101] as the representative classic models for each type, while NARS [123], and HGT [45] are taken as an example for the state-of-the-art methods for each.

Training Settings. In terms of the evaluation metrics, for ACM and OGB-MAG datasets, we report the test accuracy of all baselines and our SMASH model; for OAG-L1 and OAG-Venue datasets, we report the NDCG and MRR for all models. For each dataset, we regard the model with the best validation performance as the "best model" evaluate its performance on the test dataset. We run the model 5 times and record the mean and standard deviation. For R-GCN, NARS, HAN, and HGT, we directly use their reported results.

4.4.2 Results

We present out main results of SMASH and baseline models on the four benchmark datasets in Table 4.2. As shown in the results, our proposed SMASH model outperforms all the baselines on ACM, OAG-L1, and OAG-Venue and gets the best

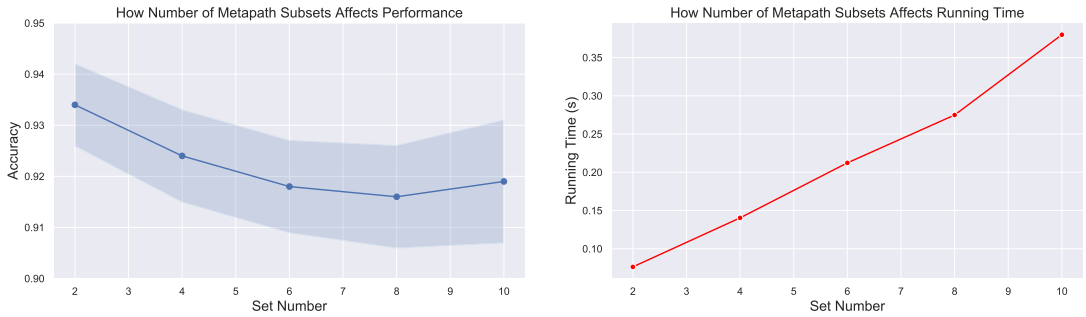


Figure 4.4: The Influence of the Number of Metapath Subsets on Accuracy (left) and Running Time (right) with ACM Datasset.

performance, for the OGB-MAG dataset, SMASH also achieved competitive top-tier performance. In addition, we provide the following 2 ablation studies to give a more comprehensive justification of SMASH.

The effect of the number of metapath subsets. We examine how the number of metapath subsets would influence the model performance in terms of accuracy and per epoch running time. We use the ACM dataset as an example, and conduct experiments with different subset numbers. Figure 4.4.2 shows how the performance would change with respect to the change in the number of metapath subsets. As ACM is a small dataset with only 2 different relation types, we find that it prefers a smaller number of metapath subsets, because we won't have too many metapaths, and therefore when the number of subsets is too large, there would be a lot of duplicates in the metapath subsets and this redundancy may lead to some performance drop. In addition, we can observe that the variance grows when the subset number grows, this is also a cons bring by the redundancy. Figure 4.4.2 shows that the per epoch running time grows linearly with related to the metapath subset number.

Table 4.3: Importance of Multi-stage Aggregation on ACM dataset.

	SMASH	SMASH$_{Sum}$	SMASH$_{Att}$	SMASH$_{Cat}$
Acc	0.934	0.7525	0.8849	0.8707
Epoch Time (s)	0.0670	0.0548	0.0556	0.0551
Epoch Num	67	47	754	373

Importance of multi-stage aggregation. We also examine the power of the multi-stage aggregation on the ACM dataset. The results are summarized in Table 4.3. We compare our SMASH model with three variants:

- SMASH$_{Sum}$: replaces the 3-stage metapath aggregation module in SMASH with a simple sum aggregator to aggregate all the metapaths
- SMASH$_{Att}$: uses the attention-based aggregator that automatically learns the importance weight for each metapath and conducts weighted summation.
- SMASH$_{Cat}$: simply concatenates all the metapaths in the aggregation module.

According to the experimental results, we find that the multi-stage aggregation has the best performance and is also very efficient. The sum aggregator has the smallest running time and needs the fewest epoch numbers to get fully converged, but its performance is the worst. The attention-based aggregator and the concatenation aggregator can obtain better performance than the sum aggregator, but require more epochs to get converged and need more total running time.

4.5 Conclusion

In this work, we aim to design a Heterogeneous GNN model that can alleviate the challenges for relation-based models and metapath-based models, which are

potential semantic information loss and the scalability issue plus the domain-expertise requirement respectively. We claim that the key is to cover more metapaths with critical semantics and keep their distinguishability. We then propose the scalable and effective SMASH model, which achieves top-level performance on all the benchmark datasets and performs the best on three of them. For further research, it is worth exploring whether we can improve the grouping stage in the aggregation module to enhance the model performance.

Part II

Efficiency

CHAPTER 5

Decoupled Greedy Learning of Graph Neural Networks

5.1 Introduction

Graph Neural Networks (GNN) have been shown to be highly effective in graph-related tasks, such as node classification [55], graph classification [117], graph matching [3], and recommender system [116]. Given a graph of arbitrary size and attributes, GNNs obtain informative node embeddings by first conducting a graph convolution operation to aggregate information from the neighbors of each node, and then transforming the aggregated information. As a result, GNNs can fuse together the topological structure and node features of a graph, and have thus become dominant models for graph-based applications. Despite its superior representation power, however, training GNNs raises challenges in run time and memory.

The first challenge is that the graph convolution operation has been shown to be expensive when GNNs become deep and wide [15]. Therefore, training a deep GNN model is challenging for large and dense graphs. Since deep and wide GNNs are becoming increasingly important with the emergence of complicated classification tasks such as [44], and semantic segmentation tasks as introduced in [60], we focus

here on studying methods for alleviating computational burdens associated with large-scale GNN training.

Several strategies have been proposed during the past years to alleviate this computation issue of large-scale GNNs. GraphSAGE [38] took the first step to use a neighborhood sampling strategy for GNNs training, which only aggregates a sampled subset of neighbors of each node in the graph convolution operation. However, though this method helps reduce memory and time costs for shallow GNNs, it computes the representation of a node recursively, and the node's receptive field grows exponentially with the number of GNN layers, which may make the memory and time costs even go larger for deeper GNNs when the sample number is big. The work of [15, 17, 144] developed sampling-based stochastic training methods to train GNNs more efficiently and avoid this exponential growth problem. [20] proposed a batch-learning algorithm by exploiting the graph clustering structure. Beyond the aforementioned methods, recently, [120] proposed a layer-wise sequential training algorithm for GNNs, which decouples the aggregation and transformation operations in the per-layer feed-forward process and reduces the time and memory cost during training while not sacrificing too much model capability, this indicates the GNN layers do not have to be learned jointly. However, the sequential design in current layerwise training hinders the further enhancement of efficiency.

The second challenge is the inefficiency caused by the sequential nature of standard backpropagation [5], which is shared by all the neural networks. As pointed out in [49], backpropagation for deep neural networks suffers an update-locking problem, which means each layer heavily relies on upper layers' feedback to update itself, and thus, it must wait for the information to propagate through the whole network before updating. This would be a great obstacle for GNN layers to be trained in parallel

to alleviate computation pressure under time and memory constraint, and would prohibit the GNN training to be trained in an asynchronous setting.

In this work, using semi-supervised node classification as an example, we show that greedy learning would help to decouple the optimization of each layer in GNNs and enable GNNs to achieve update-unlocking, i.e., allow the GNN layers to update without getting any feedback from the later layers. By using this decoupled greedy learning for GNNs, we can achieve parallelization of the network layers, which would make the model training much more efficient and would be very important for time or memory-limited applications. Moreover, we propose to use a lazy-update scheme during training, which is to exchange information between layers after a certain number of epochs instead of every epoch, this will further improve the efficiency while not sacrificing much performance. We theoretically analyze the computation complexity of our proposed method, and analogue our method to the classic block coordinate descent optimization to enable further analysis. We run a set of experiments to justify our model, and show its great efficiency on all benchmark datasets. On the newly proposed large OGBN-arxiv dataset, when training a 7-layer model, our proposed method even saves 85% time and 66% per-GPU memory cost of the conventionally trained GCN.

Our main contributions can be summarized as follows. **First**, we introduce a decoupled greedy learning algorithm for GNNs that achieves update-unlocking and enables the GNN layer to be trained in parallel. **Next**, we propose to leverage a lazy-update scheme to improve the training efficiency. We evaluate our proposed training strategy thoroughly on benchmark datasets, and demonstrate it has superior efficiency while not sacrificing much performance. **Finally**, our method is not limited to the GCN and the node classification task, but can be combined with other

scalability-enhancing GNNs and can be applied to other graph-related tasks.

5.2 Related Work

Before discussing our proposed approach, we review here related work on efficient training strategies for GNNs. The computational complexities of the discussed methods are summarized in Table 5.1. Here $\bar{D}$ denotes the average degree, b denotes the batch size, s_{node} and s_{layer} are the number of sampled neighbors in NS and IS respectively, K is the dimension of embedding vectors (for simplicity, assume it is the same across all layers), L is the number of layers, N is the number of nodes in the graph, $\boldsymbol{A}$ is the adjacency matrix, T is the number of iterations, T_{wait} is the waiting time for LU-DGL-GCN.

5.2.1 Deep Graph Convolutional Network (DeepGCN)

Graph convolutional network [55] is one of the most popular models for graph-related tasks. Given an undirected graph $\mathcal{G}$ with node feature matrix $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ and adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ where N is node number and D is feature dimension, let $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$, $\tilde{\boldsymbol{D}}$ be a diagonal matrix satisfying $\tilde{\boldsymbol{D}}_{i,i} = \sum_{j=1}^{N} \tilde{\boldsymbol{A}}_{i,j}$, and $\boldsymbol{F} = \tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}$ be the normalized $\tilde{\boldsymbol{A}}$, then, the l-th GCN layer will have the output $\boldsymbol{H}^{(l)}$ as $\boldsymbol{H}^{(l)} = \sigma(\boldsymbol{F}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l)})$, where σ is the non-linear transformation, and $\boldsymbol{W}^{(l)}$ is the trainable weight matrix at layer l.

As pointed out in [64], when GCN becomes deep, it will suffer a severe over-smoothing problem, which means the nodes will become not distinguishable after stacking too many network layers. However, for applications such as semantic

segmentation [60] or classification tasks on large datasets [44], we do need deeper GCN models. Therefore, we follow the work of [60], alleviating the over-smoothing problem by adding residual links between GCN layers and obtain the deepGCN model. The l-th layer of our network model will be $\boldsymbol{H}^{(l)} = \sigma(\boldsymbol{F}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l)}) + \boldsymbol{H}^{(l-1)}$.

5.2.2 Efficient GNN Training

To alleviate the expensive computation issue of GNN introduced in the previous section, a lot of sampling-based algorithms were proposed to train GNNs more efficiently.

GraphSAGE [38] introduced a node sampling strategy (NS), which is to randomly sample s neighbors for each node at each layer, then, for each node, instead of aggregating embeddings of all its neighbors, we only aggregate the sampled ones. VRGCN [15] also followed this NS strategy, but it further proposed to leverage history activation to reduce the variance of the estimator. Though the NS scheme has a smaller complexity compared to full-batch GNN, there exists redundant computation and the complexity grows exponentially with the layer number.

Layer-wise importance sampling strategy (IS) would be a more advanced method for efficient GNN training. FastGCN [17] proposed to sample nodes for each layer with a degree-based sampling probability in order to solve the scalability issue in NS. The work of LADIES [144] leveraged the IS idea as well, but it proposed a layer-dependent importance sampling scheme, which enjoys a smaller variance while maintaining the same level of complexity as FastGCN. Though IS is better than NS in general, we still have to trade off the complexity with the performance (i.e. accuracy for classification tasks) via sample size, as a larger sample size will enhance

Table 5.1: Summary of Complexity.

Methods	Memory (per GPU)	Time
GCN, GIN [55, 111]	$\mathcal{O}(LNK + LK^2)$	$\mathcal{O}(TL\|\boldsymbol{A}\|_0 K + TLNK^2)$
GraphSage [38]	$\mathcal{O}(bKs_{node}^{L-1} + LK^2)$	$\mathcal{O}(bTKs_{node}^{L} + bTK^2 s_{node}^{L-1})$
VR-GCN [15]	$\mathcal{O}(LNK + LK^2)$	$\mathcal{O}(b\bar{D}TKs_{node}^{L-1} + bTK^2 s_{node}^{L-1})$
FastGCN [17]	$\mathcal{O}(LKs_{layer} + LK^2)$	$\mathcal{O}(TLKs_{layer}^{2} + TLK^2 s_{layer})$
LADIES [144]	$\mathcal{O}(LKs_{layer} + LK^2)$	$\mathcal{O}(TLKs_{layer}^{2} + TLK^2 s_{layer})$
ClusterGCN [20]	$\mathcal{O}(bLK + LK^2)$	$\mathcal{O}(TL\|\boldsymbol{A}\|_0 K + TLNK^2)$
LGCN [120]	$\mathcal{O}(NK + 2K^2)$	$\mathcal{O}(L\|\boldsymbol{A}\|_0 K + 2TLNK^2)$
LU-DGL-GCN (ours)	$\mathcal{O}(NK + 2K^2)$	$\mathcal{O}(T\|\boldsymbol{A}\|_0 K/T_{wait} + 2TNK^2)$

the performance but also increase the computation cost and vice versa.

Except for the aforementioned methods, we also have ClusterGCN [20], which proposes to partition the graph into several clusters and then randomly select multiple clusters to form a batch to train the GNN. Though this would allow us to train much deeper GCN without much time and memory overhead, the stability of the performance of this approach would be hard to guarantee, since the performance heavily depends on the graph clustering settings.

5.2.3 Layer-wise GNN

Layerwise learning for neural networks is first introduced and well-discussed in [**?** 8]. The work of [5, 6] explored the layerwise CNNs and achieved impressive results.

Recently, [120] proposed a layerwise algorithm for GNN training. The key idea is to train GNNs layer by layer sequentially. Figure 5.1 illustrates the sequential training

framework for layerwise GNN. For a L-layer GNN, its first layer is trained with an auxiliary classifier. Once the current layer $l-1$ is fully converged, the next layer l will be optimized. The process will be repeated until the L-th layer is optimized. This layerwise training saves us a lot of memory, since this method only requires us to focus on one layer and only need to store one layer's activation results. Besides the clear memory saving, this layerwise training scheme also saves us a lot of time. During the learning process, it can decouple the two key components in the per-layer feed-forward graph convolution: aggregation and transformation. Then for each layer, it only needs to conduct the aggregation once at the beginning of the training, and then only needs to do the transformation step at each iteration, which greatly reduces the time cost. According to the reported results, this sequentially-trained layerwise method is more efficient than the joint learning strategy and can get us good performance. However, its sequential scheme would bring some inefficiency because one layer has to wait until its previous layers get fully converged to start training. In our work, we solve this problem and enable the model parallelization to further improve efficiency.

5.3 Proposed Approach

5.3.1 Model Architecture

As mentioned in section 5.2.1, we introduce our proposed algorithm with deepGCN model (i.e., the GCN model with residual link) since the residual link would help to alleviate GCN's over-smoothing problem when it goes deep, which would be important for large scale scenarios.

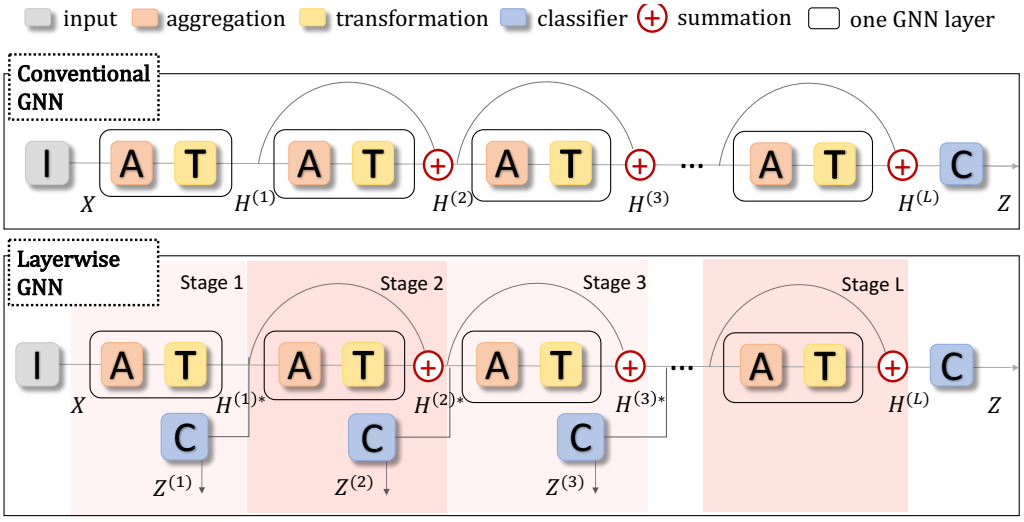


Figure 5.1: High-level Framework of Conventional DeepGCN (upper) and Layerwise DeepGCN (lower).

There exist two ways to train such GCN model: conventional training and sequential layerwise training. We illustrate these two strategies with the high-level framework shown in Figure 5.1. The aggregation step (A) corresponds to $\boldsymbol{F}\boldsymbol{H}^{(l-1)}$ and the transformation step corresponds to $\sigma(\cdot\boldsymbol{W}^{(l)})$. In conventional training, both steps are done for every iteration. For sequential layerwise training, we can decouple these two operations to conduct the transformation step in every iteration, and the aggregation step *only once* at the beginning of each layer, which results in the demonstrated time-saving. For conventional training, the learnable parameters in all layers and in the classifier are jointly optimized. For layerwise training, the training for a L-layer GNN is decomposed into L sequential stages, each stage has to wait for all its previous layers to get fully converged to start training.

Note that, sequential layerwise training has the advantage that it can save time and memory while not compromising too much performance, this suggests its promising

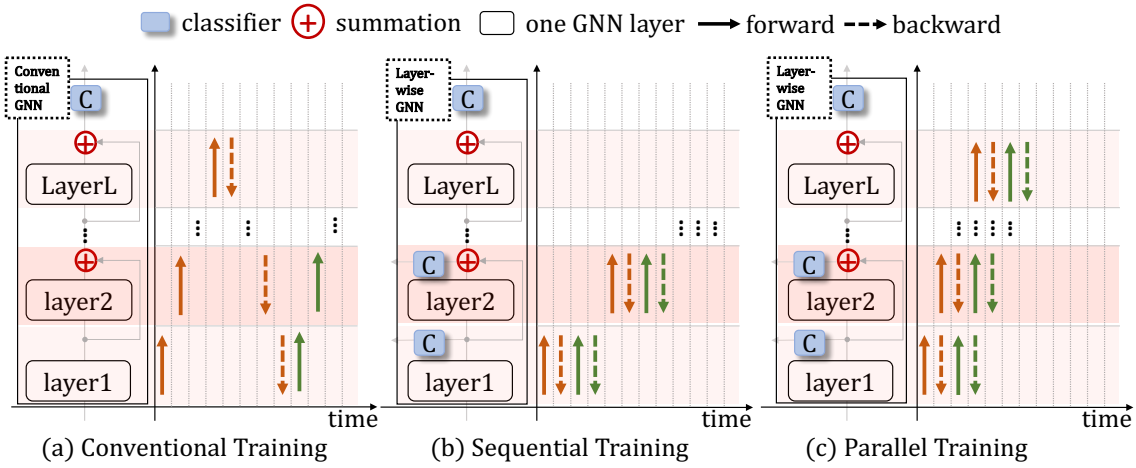


Figure 5.2: Signal Propagation Process for Conventionally Trained GNN, Sequentially Trained Layerwise GNN, and the Parallel Trained GNN.

applications in large-scale models under hardware and time constraints. We now consider, *whether we can extend it to a parallel version, so that the efficiency can be further improved?* Interestingly, as shown in the following sections, we find the answer is affirmative.

5.3.2 Decoupled Greedy Learning Algorithm

To enable parallel GNN training, the most challenging problem is update-locking. Before updating one layer, we have to wait until after the signal has been passed through all its successors, which would bring inefficiency. To alleviate this problem, we follow the design of layerwise GNN: decoupling the GNN model into different layers, associating each layer with an auxiliary classifier, which is an MLP layer with softmax activation, and assigning a per-layer greedy objective. Then, with the output

activation of a given layer, we can leverage the auxiliary classifier to optimize the per-layer objective and therefore can update the current layer without any feedback from its successors while the rest layers are still in the forward process. We name our training strategy as **D**ecoupled **G**reedy **L**earning of **GNN**s (DGL-GNN).

With our DGL-GNN, we achieve update-**un**locking, and therefore can enable parallel training for layerwise GNNs. For clarity, we provide Figure 5.2 to compare the signal propagation process of the conventionally trained GNN, sequentially trained layerwise GNN, and the parallel trained GNN. Arrows of different colors represent different batches of data (can be either mini-batch or full-batch). We assume the forward process and backward process have the same time cost, we also assume the auxiliary classifier computation is negligible, these are only for simpler illustration purposes. With this illustration, we can observe that the parallel training of layerwise GNN can avoid the case in which one layer is forwarding or back-propagating the signal while other layers are idle. Therefore, given the same number of batches of data, the parallel version would finish training much earlier than the conventional and the sequential version.

Following our previous notations, we denote by $\boldsymbol{F}$ the normalized adjacency matrix, $\boldsymbol{H}^{(l)}$ the output activation of l-th layer, and $\boldsymbol{W}^{(l)}$ the learnable parameters for l-th layer. Plus, let $\boldsymbol{Y}$ be the labels, $\Theta^{(l)}$ be the parameters for l-th layer's classifier, and $loss$ be the cross-entropy loss which is frequently used for classification tasks. Then, we have the per-layer objective function: $loss_{(\boldsymbol{W}^{(l)},\Theta^{(l)})}(\boldsymbol{Y}, \boldsymbol{H}^{(l-1)})$. We now formally define our DGL-GNN training method in algorithm 1. Note that, the inner for-loop can be done in a parallel manner, i.e., when the $l-$th layer is working on the backward process as given in line 5, the $(l+1)-$th layer can start forward propagation as given in line 4. Therefore, we claim our DGL-GNN algorithm can

Algorithm 1 Decoupled Greedy Learning (DGL) of GNNs

Require: Normalized Adjacency Matrix $\boldsymbol{F}$; Feature Matrix $\boldsymbol{X}$; Labels $\boldsymbol{Y}$; Total Number of Iterations T; Total Number of Layers L.

Initialize: $\boldsymbol{H}^{(0)} = \boldsymbol{X}$;
for $t = 1$ to T **do**
  for $l = 1$ to L **do**
    $\boldsymbol{H}^{(l)} = \sigma(\boldsymbol{F}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l)})$ // *Forward.*
    $(\boldsymbol{W}^{(l)}, \Theta^{(l)}) \leftarrow$ Update with $\nabla loss_{(\boldsymbol{W}^{(l)}, \Theta^{(l)})}(\boldsymbol{Y}, \boldsymbol{H}^{(l-1)})$ // *Backward.*
  end for
end for

achieve update-**un**locking.

We then empirically observed that, without passing the signal to the next layer immediately after the forward process, we still get the same-level performance. Thus, we find that the efficiency of DGL-GNN can be further improved by leveraging an **L**azy **U**pdate scheme (LU-DGL-GNN). Instead of using the up-to-date activation output from its predecessor, one layer can use the history activation to learn its parameters and only update the history activation a few times during the overall training process. Then, same as sequential trained layerwise GNN, we only need to conduct the aggregation step for one time after each update, which saves us a lot of time. We denote by $\hat{\boldsymbol{H}}^{(l)}$ the aggregated stored history activation for layer l. We now formally define the LU-DGL-GNN method in algorithm 2, we marked its difference with DGL-GNN in blue.

To sum up, our proposed DGL-GNN and LU-DGL-GNN methods enjoy a very high efficiency because (1) we introduce an auxiliary greedy objective for each layer

Algorithm 2 Decoupled Greedy Learning (DGL) of GNNs with Lazy Update Scheme

Require: Normalized Adjacency Matrix $\boldsymbol{F}$; Feature Matrix $\boldsymbol{X}$; Labels $\boldsymbol{Y}$; Total Number of Iterations T; Total Number of Layers L; Waiting time T_{lazy}.

Initialize: $\hat{\boldsymbol{H}}^{(0)} = \boldsymbol{F}\boldsymbol{X}$;
for $t = 1$ to T **do**
  for $l = 1$ to L **do**
    $\boldsymbol{H}^{(l)} = \sigma(\hat{\boldsymbol{H}}^{(l-1)}\boldsymbol{W}^{(l)})$ // *Forward.*
    $(\boldsymbol{W}^{(l)}, \Theta^{(l)}) \leftarrow$ Update with $\nabla loss_{(\boldsymbol{W}^{(l)}, \Theta^{(l)})}(\boldsymbol{Y}, \hat{\boldsymbol{H}}^{(l-1)})$ // *Backward.*
    if $(t \ mod \ T_{lazy} == 0)$ **then**
      $\hat{\boldsymbol{H}}^{(l)} = \boldsymbol{F}\boldsymbol{H}^{(l)}$ // *Message Passing.*
    end if
  end for
end for

and thus achieve update-**un**locking; (2) we then decouple the model into layers and therefore enable the model to be trained in parallel; and (3) we finally propose to leverage the lazy-update scheme, with which we can avoid redundant computation in the aggregation step and further reduce the training time.

5.4 Analysis

In this section, we provide some theoretical justification for our proposed decoupled greedy learning algorithm. We first analyze time and memory complexity for both DGL-GCN and LU-DGL-GCN methods, then we connect our methods to block coordinate descent, which would be beneficial for further analysis.

5.4.1 Complexity Analysis

As shown in Table 5.1, our proposed methods achieve a lower complexity compared to the conventional training and other baselines. Note that DGL-GCN can be regarded as a special case in which $T_{wait} = 1$, i.e. we update the stored activation every epoch. Therefore, we focus on the complexity justification for LU-DGL-GCN.

For time complexity, first, we know that the training process consists of two fundamental operations: aggregation and transformation. The time complexity of aggregation is $\mathcal{O}(\|\boldsymbol{A}\|_0 K)$, and the time complexity of the transformation step is $\mathcal{O}(NK^2)$. Then, we note that, for LU-DGL-GCN, during the full learning process, for each layer, we have to do aggregation T/T_{wait} times, and we have to do the transformation $2T$ times because this step should be conducted for both the GNN layer and the auxiliary classifier. Since the computation for each layer is done in parallel, we know that the overall time complexity for LU-DGL-GCN should be $\mathcal{O}(T\|\boldsymbol{A}\|_0 K/T_{wait} + 2TNK^2)$. In practice, if we put different layers on different GPUs and do the training in parallel, there would be some extra non-negligible time cost for GPU communication.

For memory complexity, it also consists two components. We have to store two things for each layer during the training: the history activation and the intermediate learnable weight matrices for GNN and for the auxiliary classifier. The activation takes $\mathcal{O}(NK)$, and the two types of weight matrix take $\mathcal{O}(2K^2)$ space. Again, when we do the training in a parallel fashion and assign the layers to different machines, the per-GPU memory would only be $\mathcal{O}(NK + 2K^2)$, which is significantly reduced compared to most of the existing baselines.

5.4.2 Analogy to block coordinate descent

To justify the rationality of the proposed model, we present here an analogy of our decoupling approach to the classic coordinate descent algorithm variants.

Coordinate descent (CD) is a classic iterative optimization algorithm that solves an optimization problem by minimizing the objective along each coordinate direction successively. In each iteration, it would choose one variable, fix the other components, and then optimize the objective with respect to only the single variable. By doing so, we only need to solve a lower dimensional minimization problem at each iteration, which would be easier. CD algorithm has been discussed in various kinds of literature and has been used in applications for a long time [106, 108, 84]. Block coordinate descent (BCD) is an extension of the CD method. The difference between BCD and the conventional CD is that BCD will do the searching along a coordinate hyperplane instead of a single coordinate direction [4], i.e. it groups variables into blocks, and minimizes the objective with respect to only one block of variables at each iteration while fixing the others. For the BCD algorithm, there exist its parallel implementations. As introduced in the work of [106], we can categorize them into two types: synchronous and asynchronous. For synchronous parallel BCD, we partition the computation into pieces and put different pieces on different processors, each processor will update a part of the variables in parallel, then a synchronization step should be conducted to guarantee the consistency of the information shared among all processors before further computation. For asynchronous settings, the difference is we do not have to do the synchronization.

We observe that, the sequential layerwise GNN shares a similar high-level idea with the BCD method. We regard each layer and its associated auxiliary classifier as

a module. Then, for each module, all its parameters can be treated as a coordinate block, we order the coordinate block according to which layer it corresponds to. Note that, in BCD, for each iteration, we choose one coordinate block and optimize the overall training objective with respect to the chosen block. So if we keep choosing the first coordinate block until it fully converges, then keep choosing the second coordinate block, etc., until the last coordinate block fully converges, then this optimization process is the same as the learning process of a sequentially trained layerwise GNN. We should note that, there are slight differences between BCD and the layerwise training. For BCD, one may optimize towards each coordinate block for multiple rounds, but in the layerwise training, we only optimize towards each coordinate block once. As long as we finish the training of one layer, we won't go back and re-optimize this layer in the latter stage. One may also doubt that, for BCD optimization, when optimizing towards one coordinate block, though later blocks are fixed, they would still be useful in defining the optimization objective. But in the layerwise training, it seems that each layer has its own greedy objective. For this question, we can consider the global optimization objective for layerwise training as the summation of the greedy objective at each layer and the classification objective for the downstream task. Then, when we optimize one layer, though other layers' parameters are fixed, they still contribute to the global optimization objective.

We then observe that, the DGL-GCN can be analog to the synchronous parallel BCD and the LU-DGL-GCN can be regarded as an analogy of asynchronous parallel BCD. Note that our DGL-GCN and LU-DGL-GCN can be implemented in a parallel fashion, and their key difference is whether all the layers share consistency and up-to-date information. Therefore, if we make the same analogy of learnable parameters and the coordinate blocks as in the above sequential version, then it would be easy to

find the similarity between parallel BCD and our decoupled greedy learning methods. With such an analogy, it would allow us to leverage existing theorems for BCD optimization to better understand and analyze the DGL-GCN and LU-DGL-GCN.

5.4.3 Convergence Guarantee

We then justify the convergence guarantee for our proposed algorithm 1. We first show the rationality of some standard assumptions in [11], then we compare our settings to the parallel CNN setting in [5] and show the convergence theorem applies to our setting as well.

We consider our algorithm to be optimized by the stochastic gradient descent method and denote by $\eta(t)$ the learning rate (i.e. step size) at iteration t. Then we state the three standard assumptions as follows:

Assumption 1 (γ-smoothnes, [11]) The loss function $loss_{(\boldsymbol{W}^{(l)},\Theta^{(l)})}(\boldsymbol{Y}, \boldsymbol{H}^{(l-1)})$ is differentiable, and its gradient is γ-Lipschitz.

Assumption 2 (Robbins-Monro conditions, [11]) The learning rate satisfies $\sum_t \eta(t) = \infty$, $\sum_t \eta(t)^2 < \infty$.

Assumption 3 (Finite Variance) There exists constant μ, such that $\forall t$, we have $\|\nabla loss_{(\boldsymbol{W}^{(l)},\Theta^{(l)})}(\boldsymbol{Y}, \boldsymbol{H}^{(l-1)})\|^2 \leq \mu$. If we use batch training, this formula becomes $\mathbb{E}[\|\nabla loss_{(\boldsymbol{W}^{(l)},\Theta^{(l)})}(\boldsymbol{Y}_B, \boldsymbol{H}_B^{(l-1)})\|^2] \leq \mu$, where $\boldsymbol{Y}_B$ and $\boldsymbol{H}_B^{(l-1)}$ correspond to the sampled batch.

Note that in the parallel training process shown in Algorithm 1, the input for each layer (except the first layer) is evolving, following [5], when analysing the convergence guarantee for layer l, we need one more assumption for the convergence of the previous layer $l-1$. Let $\boldsymbol{Z} = (\boldsymbol{H}^{(l-1)}, \boldsymbol{Y})$, denote by $p_t^{(l)}(\boldsymbol{Z})$ the input data distribution for

layer l at iteration t, and denote by $p^{(l)*}(\boldsymbol{Z})$ the input data distribution for layer l when the $(l-1)$-th layer gets fully converged. Let $d^{(l)}(t) = \int_Z |p_t^{(l)}(\boldsymbol{Z}) - p^{(l)*}(\boldsymbol{Z})| d\boldsymbol{Z}$ be the distance between the input data distribution for layer l at iteration t and the fully converged input data distribution for layer l. Then, the assumption should be:

Assumption 4 (Convergence of $(l-1)$-th layer, [5]) We assume that $\sum_t d^{(l)}(t) < \infty$, which guarantees the previous layer can get fully converged.

Let $\mathcal{L}(\boldsymbol{W}^{(l)}, \Theta^{(l)}) = \mathbb{E}_{p^{(l)*}}[loss_{(\boldsymbol{W}^{(l)}, \Theta^{(l)})}(\boldsymbol{Y}, \boldsymbol{H}^{(l-1)})]$. With the aforementioned assumptions, we find that the Theorem in [5] applies to our graph setting as well, and thus we can apply the following theorem as our convergence guarantee:

Theorem ([5]) With assumptions 1$\sim$4, we have:

$$\sum_t \eta(t) \mathbb{E}[\|\nabla \mathcal{L}(W^{(l)}, \Theta^{(l)})\|^2] \leq \mathbb{E}[\|\nabla \mathcal{L}(W^{(0)}, \Theta^{(0)})\|^2] + \mu \sum_t \eta(t) (\sqrt{2d^{(l)}(t)} + \frac{\gamma \eta(t)}{2})$$

In the above inequality, by leveraging the assumptions and the Cauchy-Schwartz inequality, we can know the right-hand side is bounded, therefore, we can justify the convergence of our model.

5.5 Experimental Results

We evaluate our proposed algorithms with the multi-class node classification task. However, it should be noted that the decoupled greedy learning method can also be applied in other graph-related tasks and is not limited to the node classification task.

Table 5.2: Statistics of Benchmark Dataset

Dataset	**Cora**	**Citeseer**	**Pubmed**	**Reddit**	**OGBN-arxiv**
Nodes	2,708	3,327	19,717	232,965	169,343
Edges	5,429	4,732	4,4338	11,606,919	1,166,243
Classes	7	6	3	41	40
Feature	1,433	3,703	500	602	100

5.5.1 Experiment Settings

Dataset. We use the following public datasets for evaluation: cora, citeseer, pubmed [83], Reddit [38], and OGBN-arxiv [44]. We summarize the dataset statistic in Table 5.2.

Baseline. In the main experiment, we compare our method against several baseline models: GCN [55], FastGCN [17], LADIES [144], and LGCN [120]. We do not consider node-wise sampling such as GraphSage and VRGCN since it is shown to be less efficient compared to Layer-wise sampling. For all the methods, we use the same deepGCN model architecture and set all the hidden dimensions as 128. For Cora, Citeseer, Pubmed and Reddit, we use a 5-layer model, and for OGBN-Arxiv, we use a 7-layer model. We follow the public implementations of the baselines, and use their parameter settings.

Metrics. We evaluate the performance of different methods with the following evaluation metrics: **Accuracy (%)**: The micro F1-score of the test data at the convergence point. **Memory (MiB)**: The maximum per-GPU memory cost during training. **Total Running Time (s)**: The total training time (exclude validation) before convergence.

Training Settings. We conduct the training 10 times and take the mean and

variance of the evaluation results. For each running time, we apply batch training, run each model 200 epochs and 500 epochs for small datasets (Cora, Citeseer, Pubmed) and large datasets (OGBN, Reddit) respectively, and guarantee convergence. We use the parameters that can achieve best validation performance to do the test. We run the experiments on Tesla V100 GPUs (16GB). For the sampling-based methods: FastGCN and LADIES, we set the sample number as 64, increasing this number would improve the accuracy, but would increase the computation cost. In addition, for the sampling-based method, we set the batch number as 10, and set the batch size as 512.

5.5.2 Main Results

We summarize the node classification performance results in Table 5.3, which demonstrates the efficacy of our approach. Here, for the sampling-based methods (especially FastGCN), we can sacrifice the training time to obtain a higher accuracy by using a larger sample size. We find that the decoupled greedy learning method can greatly reduce the time and memory cost of GCN without a huge performance drop. Compared to the sampling-based method, the accuracy of our method is more stable, also, in our later ablation study, we show our method can be combined with the sampling-based method to further boost efficiency. We should also note that T_{lazy} is important in the LU-DGL-GCN method, increasing T_{lazy} can reduce running time, but may lead to unsatisfactory accuracy performance (like in Citeseer).

Table 5.3: Comparison of DGL-GCN and LU-DGL-GCN With Baseline Methods on Benchmark Datasets. Set $T_{lazy} = 50$ for LU-DGL-GCN.

Dataset	Method	Accuracy(%)	Total Time(s)	Mem(MiB)
Cora	GCN	77.8 ± 1.3	42.2 ± 1.0	31.7
	LADIES(64)	78.8 ± 0.8	31.5 ± 0.8	3.1
	FastGCN(64)	55.2 ± 4.8	36.8 ± 2.1	3.1
	LGCN	80.4 ± 0.9	119.7 ± 11.5	6.9
	DGL-GCN (ours)	78.0 ± 1.3	17.3 ± 1.2	6.9
	LU-DGL-GCN (ours)	65.4 ± 14.1	14.1 ± 0.4	6.9
Citeseer	GCN	65.6 ± 2.4	33.1 ± 1.2	67.9
	LADIES(64)	66.6 ± 1.2	32.5 ± 1.2	5.9
	FastGCN(64)	36.0 ± 1.0	34.5 ± 1.7	5.9
	LGCN	67.1 ± 1.7	107.9 ± 5.2	14.7
	DGL-GCN (ours)	64.8 ± 3.2	15.8 ± 0.9	14.7
	LU-DGL-GCN (ours)	54.8 ± 6.3	13.9 ± 0.2	14.7
Pubmed	GCN	74.8 ± 2.6	46.9 ± 2.0	137.9
	LADIES(64)	77.9 ± 2.4	33.8 ± 1.5	1.9
	FastGCN(64)	41.2 ± 0.5	34.6 ± 1.4	1.9
	LGCN	76.2 ± 1.6	141.8 ± 12.8	29.1
	DGL-GCN (ours)	75.5 ± 0.8	18.6 ± 1.1	29.1
	LU-DGL-GCN (ours)	74.9 ± 0.9	14.9 ± 1.0	29.1
OGBN-arxiv	GCN	71.9 ± 0.2	162.9 ± 23.4	1568.7
	LADIES(64)	49.1 ± 2.8	50.4 ± 16.8	3.1
	FastGCN(64)	21.6 ± 0.0	24.0 ± 6.3	3.1
	LGCN	68.8 ± 0.1	122.5 ± 17.2	385.2
	DGL-GCN (ours)	69.1 ± 0.3	98.3 ± 10.2	385.2
	LU-DGL-GCN (ours)	69.3 ± 0.3	31.2 ± 0.1	385.2
Reddit	GCN	69.6 ± 0.5	436.0 ± 18.4	2370.5
	LADIES(64)	70.0 ± 0.4	63.3 ± 1.2	3.8
	FastGCN(64)	67.1 ± 3.2	40.6 ± 0.2	3.8
	LGCN	69.1 ± 0.4	544.2 ± 3.3	588.6
	DGL-GCN (ours)	72.4 ± 4.5	133.8 ± 0.4	588.6
	LU-DGL-GCN (ours)	73.7 ± 1.5	65.7 ± 0.9	588.6

Table 5.4: Comparison of LU-DGL-GCN With Different T_{lazy}.

Dataset	Performance	$T_{lazy} = 1$	$T_{lazy} = 5$	$T_{lazy} = 10$	$T_{lazy} = 20$	$T_{lazy} = 50$
OGBN-arxiv	Accuracy (%)	69.1 ± 0.3	69.4 ± 0.3	69.5 ± 0.2	69.4 ± 0.2	69.3 ± 0.3
	Total Time (s)	98.3 ± 10.2	63.5 ± 2.1	44.1 ± 0.8	38.6 ± 0.2	31.2 ± 0.1
	Memory (MiB)	385.2	385.2	385.2	385.2	385.2

5.5.3 Ablation Study

In the following, we provide ablation studies to further justify the superiority of our proposed method.

Importance of Lazy Update Scheme. We illustrate the advantage of the Lazy Update Scheme and show how the waiting time T_{lazy} will influence the performance. We use the OGBN-arxiv as an example, and follow previous model architectures and parameter settings. We summarize the results in Table 5.4. Note that, $T_{lazy} = 1$ corresponds to the DGL-GNN model. We find that, with the lazy update scheme, we can greatly reduce the time cost, and a proper waiting time T_{lazy} may even improve the accuracy a little bit since it can alleviate overfitting.

Sequential Training v.s. Parallel Training. Finally, we briefly compare the sequential training and our parallel training of the greedy objective. Again, we use Cora and OGBN-arxiv as examples and follow the above experiment settings. We present the results in Figure 5.3, the Left column shows how the downstream task's accuracy changes with epochs, the middle column shows how the current layer's accuracy changes with time, and the right column shows how last layer's accuracy changes with time (therefore for the sequential layer-wise training, the accuracy is 0 at early period). In terms of accuracy, we observe that parallel training can quickly catch up with sequential training, and in terms of running time, parallel training is

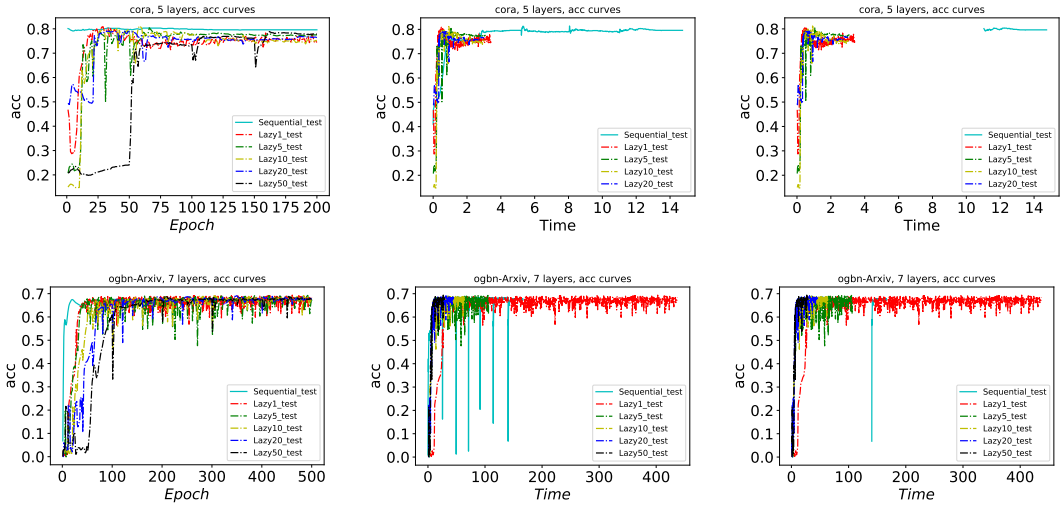


Figure 5.3: Comparison of Sequential and Parallel Training.

Table 5.5: The Decoupled Greedy Learning Method Can Also Be Combined With the Graph Isomorphism Network Model.

		Cora	Pubmed
GIN	Accuracy (%)	75.7 ± 1.1	73.5 ± 1.5
	Total Time (s)	6.2 ± 0.4	6.8 ± 0.3
	Mem (MiB)	31.7	137.9
LGIN	Accuracy (%)	79.9 ± 0.9	73.7 ± 2.6
	Total Time (s)	2.2 ± 0.1	1.8 ± 0.0
	Mem (MiB)	6.9	29.1
DGL-GIN	Accuracy (%)	77.0 ± 1.6	74.3 ± 1.5
	Total Time (s)	2.4 ± 0.0	1.8 ± 0.0
	Mem (MiB)	6.9	29.1

Table 5.6: The Decoupled Greedy Learning Method Can Also Be Combined With the Sampling-based Method.

		Reddit	Pubmed
FastGCN(64)	Accuracy (%)	67.1 ± 3.2	41.2 ± 0.5
	Total Time (s)	40.6 ± 0.2	34.6 ± 1.4
	Mem (MiB)	3.8	1.9
LADIES(64)	Accuracy (%)	70.0 ± 0.4	77.9 ± 2.4
	Total Time (s)	63.3 ± 1.2	33.8 ± 1.5
	Mem (MiB)	3.8	1.9
LSAMPLE(64)	Accuracy (%)	68.8 ± 0.5	78.4 ± 0.6
	Total Time (s)	102.3 ± 2.5	66.7 ± 0.7
	Mem (MiB)	1.1	0.5
DGL-SAMPLE(64)	Accuracy (%)	77.3 ± 1.4	71.8 ± 1.5
	Total Time (s)	17.1 ± 1.2	14.8 ± 0.3
	Mem (MiB)	1.1	0.5

much faster.

Decoupled Greedy Learning with GIN. We use GIN as an example to show the decoupled greedy learning method is not limited to GCN, but can be combined with other GNN models. We use Cora and Pubmed as examples, using full-batch training, run the experiments 10 times to record the mean and variance, and compare the results of GIN, layer-wise sequential GIN, and the decoupled greedy learning of the GIN model. According to the results in Table 5.5, with the decoupled greedy learning method, we can boost the efficiency of the GIN model and may even improve its performance.

Decoupled Greedy Learning with Sampling. We now show the decoupled greedy learning method is not orthogonal to the sampling method, but can be complementary. We use Reddit and Pubmed as examples, and follow the main

experiment setting. As shown in Table 5.6, the decoupled greedy learning method can further enhance the efficiency of the sampling-based method. We should note that, since we decouple the layers, we only use importance sampling but not layer-dependent sampling, that's why we have a performance drop compared with LADIES.

5.6 Conclusions

In this paper, we focus on the efficiency issue of GNN training in large-scale applications and present a decoupled greedy GNN learning strategy. Our proposed DGL-GNN model achieves update-**un**locking by introducing greedy auxiliary objectives during training, and enables parallelization by decoupling the GNN into smaller modules. We also propose to leverage a lazy-update scheme during training to further improve the model efficiency. We empirically analyze our proposed model and demonstrate its effectiveness and superior efficiency through a range of experiments. We note that while we introduce our proposed method with GCN model and use the semi-supervised node classification task as an example, the method is not limited to this setting, and can be applied to other GNNs and graph-related downstream tasks. Further, while here we focus on comparing the decoupled approach as an alternative to other sampling-based methods respect to their accuracy and efficiency, these approaches can be regarded as complementary to each other. By combining the decoupled greedy learning method with other scalability-enhancing improvements of GNN training, the computation cost would be further reduced, which poses a promising direction for future work.

CHAPTER 6

BLADS: Bi-Level Adaptive Sampling for Heterogeneous Information Network Training

6.1 Introduction

Many real-world datasets such as e-commerce or academic networks contain multiple types of entities interacting with diverse relations. Heterogeneous Graph (HetG) [90] is a multi-relational data type that can perfectly model these highly interactive data. Therefore, significant research attention has been put to developing models to learn representations for entities in HetGs to obtain better performance applications like classifying an object's category and predicting the connectivity of an object pair [100]. Heterogeneous Graph Neural Networks (Heterogeneous GNNs) are the State-of-the-Art family of models for learning on heterogeneous graphs and top most existing benchmark leaderboards [44, 71, 100].

Mini-batch training is typically used to scale Heterogeneous GNNs training to very large graphs but can still be ineffectual with the existence of high-degree nodes that lead to uncontrollable neighbor expansion in the minibatch graph. Sampling is an effective way to improve scaling efficiency for training the Graph Neural Networks (GNNs) for homogeneous graphs (HomG), as it limits the neighbor fanout [38] for

each node during training.

An intuitive idea to apply sampling to a HetG is to uniformly randomly sample (Uniform) a limited number of neighbors for each node, i.e. in a node-wise sampling setting, for each node, given a sampling budget, regardless of its node type and its neighbors' node type, we sample the neighbor node with the same probability. We show that this approach fails with a 'noisy' graph, where for the target nodes, most of the connected neighbor types have a large number of instances that are uninformative for the downstream task, while the valuable neighbor type only has a few instances.

We can address the shortcoming described above with a type-aware uniform random sampling (TAUniform), i.e., in the node-wise sampling setting, for each node, given the sampling budgets for each neighbor type, we sample the neighbor nodes of the same type with the same probability. However, we show that when the given HetG has a complex schema and some node types have numerous connected neighbor types, then applying this TAUniform requires a large total sampling budget, which is less efficient and is counter to the motivation of using sampling to reduce the computation cost.

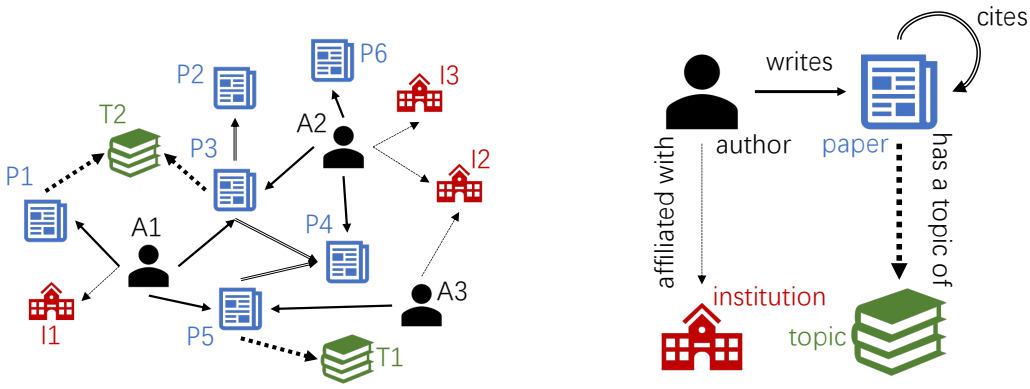
Pioneering works such as VRGCN [15], LADIES [144], and SHADOW [127] have laid foundations on non-uniform sampling on graphs. The work of PASS [118], ASGCN [46], and GCN-BS [70] even explores adaptive sampling distributions which allow the sampling probability on each neighbor node to be learned according to the downstream tasks. However, they focus on HomGs, and lack a discussion on how to take care of the heterogeneous semantics in HetGs. The work of HetGNN [129] and HGSampling [45] took steps in sampling for HetGs, however, they either consider a uniform random walk, or consider uniformly sample from each neighbor

type, and therefore are still within the scope of Uniform and TAUniform. Therefore, the question of how to effectively conduct sampling on HetG for Heterogeneous GNNs training still lacks comprehensive discussion and remains open.

In this work, we examine the Uniform and TAUniform in Heterogeneous GNNs training, and identify their challenges with case studies. Then, we develop a sampling method, Bi-Level Adaptive Sampling (BLADS), to address these tough cases by performing sampling from both schema and entity levels. The schema-level sampler learns to generate more samples of informative neighbor types while the entity level uses a uniform random sampler for each neighbor type to sample diverse instances. With the framework, we learn a sampling policy customized to the downstream task - the Heterogeneous GNNs gets more informative neighbor samples, while the sampler learns an adaptive policy with the high-quality embeddings learned by Heterogeneous GNNs as input - improving task performance. Further, the sampling probability on each relation type learned by the schema-level sampler in BLADS can be interpreted as the importance score on each neighbor type, and therefore provides the side benefit of explainability.

6.2 Preliminaries

In this section, we formally define our problem setup, and provide an in-depth discussion of existing sampling methods on graphs.



(a) Partial HetG for OGBN-MAG Dataset (b) Schema for OGBN-MAG Dataset

Figure 6.1: An Illustration Example on Partial OGBN-MAG.

6.2.1 HetG and Notations

In this work, we focus on the Attributed Heterogeneous Graphs $\mathcal{G}(\mathcal{V}, \mathcal{E}, \Phi, \Psi, \{\mathbf{X}_{(t)}\})$ [88], which contains multiple types of entities (also referred to as nodes), relations (also referred to as edges), and node attributes. Figure 6.1(a) provides a partial graph example for the OGBN-MAG dataset [44], Figure 6.1(b) is its schema. To make all the notations clear, we summarize all the major symbols in the paper in Table 6.1.

6.2.2 Node Classification and Link Prediction with Heterogeneous GNNs

In this work, we take the node classification task and the link prediction task on HetG as example downstream tasks. For the node classification task, we are given a HetG $\mathcal{G}(\mathcal{V}, \mathcal{E}, \Phi, \Psi, \{\mathbf{X}_{(t)}\})$, a target entity type t_{tar}, and a subset of the labeled entity within this target type $\mathcal{V}_{(t_{tar},l)}$. Let the number of classes be $\mathcal{C}$, and let the full entity set of type t_{tar} be $\mathcal{V}_{(t_{tar})} = \mathcal{V}_{(t_{tar},l)} \cup \mathcal{V}_{(t_{tar},u)}$, where $\mathcal{V}_{(t_{tar},u)}$ indicates the unlabeled entity set of type t_{tar}. Then, the objective of the node classification task is

	Notation	Description
HetG	$\mathcal{G}$	Heterogeneous graph
	$\mathcal{V}, v$	Full node set with multi-typed nodes for $\mathcal{G}$, a node instance
	$\mathcal{V}_{(t)}$	Node set of type t
	$\mathcal{E}, e$	Full edge set with multi-typed edges for $\mathcal{G}$, an edge instance
	$\mathcal{T}$,t, t_{tar}	Node type set, a node type, target node type for classification
	$\mathcal{R}, r$	Edge type set, an edge type
	$\Phi : \mathcal{V} \to \mathcal{T}$	Mapping function, projects node from $\mathcal{V}$ to type from $\mathcal{T}$
	$\Psi : \mathcal{E} \to \mathcal{R}$	Mapping function, projects edge from $\mathcal{E}$ to type from $\mathcal{R}$
	$\mathbf{X}_{(t)} \in \mathbf{R}^{n_t \times d_0}$	Attribute matrix for node of type t
	n_t	Number of nodes of type t
	d_0	Initial node attribute dimension
Task	$\mathcal{V}_{(t_{tar},l)}$	Labeled node set of the target node type
	$\mathcal{V}_{(t_{tar},u)}$	Unlabeled node set of target node type
	C	Number of classes
	y_v	Label for node v
	$\hat{y}_v$	Predicted label for node v
Sampler	$\mathcal{N}(v), \mathcal{N}^{(S)}(v)$	Set of neighbors of v, set of sampled neighbors of v
	$\mathcal{N}_t(v), \mathcal{N}_t^{(S)}(v)$	Set of type-t neighbors of v, set of sampled type-t neighbors of v
	$\mathcal{N}_r(v)$	Set of neighbors of v connected with relation type r
	$\mathcal{N}_r^{(S)}(v)$	Set of sampled neighbors of v connected with relation type r
	$\lvert\mathcal{N}(v)\rvert, \lvert\mathcal{N}_t(v)\rvert$	Number of neighbors of v, number of type-t neighbors of v
	$\mathcal{B}$	Sampling budget per node per layer
	$\mathcal{B}_{t^*,t}$	Sampling budget for type-t neighbors of node type t^*
	β_r	a learnable scaler for relation r in BLADS sampler
Heterogeneous GNN	$h_v^l \in \mathbf{R}^{d_l}$	Hidden embedding learned by Heterogeneous GNN's layer l
	d_l	Output dimension of layer l in Heterogeneous GNN
	L	Total layer number
	$\alpha_r^{(l+1)} \in \mathbf{R}^{d_l \times d_{l+1}}$	learnable linear projection parameter for $(l+1)-$th Heterogeneous GNN layer

Table 6.1: Notation Summary

to learn a mapping function $f : \mathcal{V}_{(t_{tar})} \rightarrow \{1, \cdots, C\}$ to predict the class labels for the unlabeled entities in $\mathcal{V}_{(t_{tar},u)}$. For the link prediction task, we would be given the HetG $\mathcal{G}$, and we aim to learn the scoring function $f : (u, r, v) \rightarrow s_{uv}^{(r)}$ in which the score $s_{uv}^{(r)}$ indicates the probability of a type-r edge existence between node pair (u, v). As Heterogeneous GNNs are shown to be the State-of-the-Art HetGs embedding methods that get the top performance on existing benchmarks [100, 114], we briefly review existing Heterogeneous GNN models.

RGCN [82] proposes to extend the seminal GCN model[55] to HetGs. At each network layer, RGCN runs a separate GCN layer for each involved relation type (the in-coming and out-going edges between a pair of nodes are treated as two different relation types), then use a problem-specific normalization coefficient to sum up the embeddings obtained by each relation types, the coefficient can either be learned or chosen in advance. R-HGNN [122] proposes to first run a GCN module (consisting of several GCN layers) for each relation type, then design a cross-relation message passing module to allow the interactions across the learned relation-specific node representations and enable the model to capture more heterogeneous semantics. HGT [45] proposes to add attention to the edges by designing node-type dependent and relation-type dependent parameters to characterize a transformer-like architecture, enabling HGT to maintain specific representation spaces for nodes and edges of different types. Though these groundbreaking works have achieved great performance in HetG embedding learning, they may scale poorly for complex large-scale HetGs due to uncontrollable neighbor expansion.

In addition to the aforementioned, some Heterogeneous GNN models go beyond immediate neighbors and explore a broader neighborhood at each network layer, such as NARS [123], HAN [101], GTN [124], and MAGNN [33]. However, in this work, we

propose a neighbor sampling method that should be built on top of Heterogeneous GNNs that consider immediate neighbors, so these models exceed our discussion scope, and we leave addressing them to future work.

6.2.3 Related Works: Sampling on Graphs

Sampling has been shown to be an effective technique for speeding up network training on graphs [67] and we briefly review existing works for sampling on graphs. Based on the graph type the sampler considers and the sampling distribution (also referred to as sampling policy) the sampler uses, we categorize these samplers into three groups as follows.

6.2.3.1 Heuristic-based Sampling on Homogeneous Graphs

. The work of *GraphSAGE* [38] took the first step to apply sampling on HomGs for GNN training, it considers neighbor sampling for each node. For each network layer, for each node, it uniformly randomly samples a fixed number of its neighbors, i.e., for a node v has $|\mathcal{N}(v)|$ neighbors, the probability to sample each of its neighbors would be $1/|\mathcal{N}(v)|$. *VRGCN* [15] improves GraphSAGE by reducing the embedding variance by using historical activation. *FastGCN* [17] considers a layer-wise sampling, for each network layer, it samples from the full node set with a degree-based probability distribution, the higher degree a node has, the higher chance it can get sampled. *LADIES* [144] improves FastGCN by restricting the sampling for the current network layer to be within the immediate neighbor set of the later network layer, i.e. any node that has the opportunity to get sampled should be connected to at least one sampled nodes for the successor network layer. *ClusterGCN* [20], *GraphSAINT* [128], and

SHADOW [127] considers sampling from subgraph perspective. These works first sample nodes or edges, then construct an induced subgraph, and use this subgraph to train the current iteration. In addition to the aforementioned pioneering works, many other samplers explore different aspects of sampling on HomGs [67, 116, 23]. However, these works are task-agnostic and would apply the same sampling heuristics regardless of the downstream task, which limits their capability to generate customized samples to enjoy the benefit in task performance.

6.2.3.2 Adaptive Sampling on Homogeneous Graphs

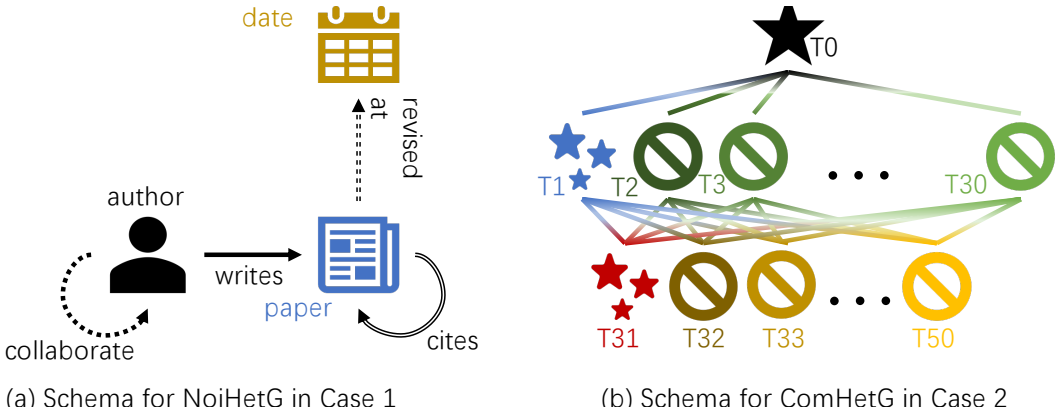
. With the aim of learning to generate task-specific samples to improve network training, several research works propose adaptive samplers on HomGs for GNN training. The work of *AS-GCN* [46] considers layerwise sampling similar to the FastGCN, but it proposes to use the adaptive sampling distribution instead of using a heuristic-based one, and is shown to be capable of learning to give higher weights to nodes that are more connected to the samples in the later network layer. *GCN-BS* [70] aims to learn an adaptive sampler to reduce the sampling variance, and formulates the variance minimization as an adversary bandit problem. *PASS* [118] proposes a performance-adaptive sampling method to learn to sample by explicitly optimizing the task performance. Though these works address the scalability problem as well as learn to sample for the sake of benefiting the downstream tasks, they focus only on HomGs. We may treat the HetGs as HomGs and apply these samplers, but this often leads to a performance drop, so these samplers are unable to be simply generalized to HetGs without careful consideration of the heterogeneity.

6.2.3.3 Heuristic-based Sampling on Heterogenous Graphs

. Heterogeneous GNNs training for HetGs confronts the same neighbor expansion concerns as GNN training on HomGs. From previous discussions, we know that the heterogeneous nature of HetGs requires additional treatment for Heterogeneous GNNs sampling. *HetGNN* [129] proposes to use random walks with restart (RWR) to sample the nodes of different types in the neighborhood based on the random walk frequency. *HGSampling* [45] adapting the LADIES sampler to generate dense subgraphs for efficient Heterogeneous GNNs training, and samples the same number of neighbors of each neighbor type. Despite their thoughtful treatment of the heterogeneous semantics in HetGs, they face similar limitations as in the Heuristic-based Sampling on HomGs, i.e. they are task-agnostic and lack the flexibility to adapt to the downstream task. Therefore, a systematic discussion of how to sample on HetGs to benefit Heterogeneous GNNs training on both efficiency and performance sides is still needed.

6.3 Method

First, we identify two challenging cases to demonstrate the limits of uniform random sampling for training Heterogeneous GNNs. Then, we propose BLADS, a bi-level adaptive treatment that addresses the identified challenge cases.



(a) Schema for NoiHetG in Case 1

(b) Schema for ComHetG in Case 2

Figure 6.2: Schema for Synthetic HetGs in the Challenge Cases.

6.3.1 Random Sampling with Uniform Distribution

6.3.1.1 Case 1: Noisy HetG

To begin, we examine an intuitive sampling method, which we term Uniform, that extends the idea of GraphSAGE [38] to HetGs. For an entity v in HetG with the total number of neighbors $|\mathcal{N}(v)|$, each of its neighbor nodes regardless of node type can get sampled with probability $1/|\mathcal{N}(v)|$ with Uniform. Applying Uniform for HetG sampling ensures each node has the same neighbor distribution as in the original full HetG.

However, in some cases, like a noisy HetG where most of the relation types are not relevant for the downstream task but have more edge instances in the graph than the few relation types that are valuable, preserving the neighbor-type distribution does not help the downstream model performance. When applying Uniform on such a graph, the sampler misses the critical neighbors in the generated computational graph.

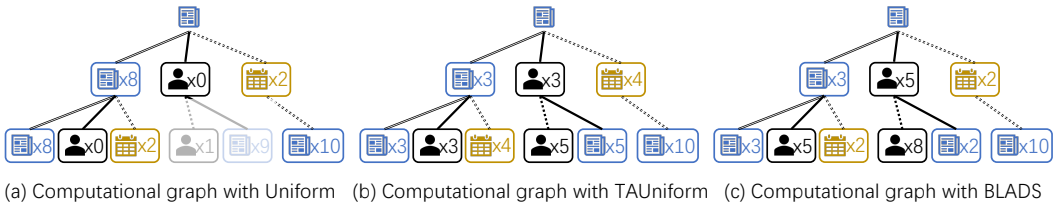


Figure 6.3: Computational Graphs Generated with Different Sampler on NoiHetG.

In order to provide a concrete instance for this case, we generate a synthetic graph called NoiHetG (Abbr. for Noisy HetG) with schema shown in Figure 6.2(a) and perform a classification task. The details of the synthetic graph generation are provided in the following:

Let us consider the task of predicting the paper node's published venue, and it is a binary classification for the venue CVPR and KDD. In total, there are 2000 paper nodes, 1000 date nodes, and 100 author nodes. The node attributes for paper nodes and for date nodes are randomly generated following a 32-dim multi-variate Gaussian distribution $\mathcal{N}(\mathbf{1}, I)$ with the 32-dim all-one vector $\mathbf{1}$ as mean, and a diagonal identity matrix $I \in \mathbb{R}^{32\times 32}$ as the variance. The node attributes for author nodes are generated with two different 32-dim multi-variate Gaussian distributions $\mathcal{N}(\mathbf{1}, I)$ and $\mathcal{N}(\mathbf{0}, I)$, where $\mathbf{0}$ indicates 32-dim all-zero vector, and each multi-variate Gaussian takes care of 50 author nodes.

The entities in NoiHetG interact as follows: For any pair of entities (date, paper), they are connected with probability 0.005. For any pair of entities (paper, paper), they are connected with probability 0.02. For any pair of entities (author, author), they can only get connected with probability 0.08 if their attributes are generated with the same multi-variate Gaussian distribution, and can never get connected

on the contrary. For any pair of entities (paper, author), if the paper's label is CVPR and the author's attribute is generated with $\mathcal{N}(\mathbf{0}, I)$, or the paper's label is KDD and the author's attribute is generated with $\mathcal{N}(\mathbf{1}, I)$, they can get connected with probability 0.06, otherwise, they can never get connected. Hence, with these connectivity generation rules, clearly, we know that the most crucial neighbor type for a paper node would be the author nodes, and the author node also takes itself as the most helpful neighbor type. However, there are only a few edge instances of the critical (paper, author) and (author, author) types in NoiHetG, and therefore, these valuable links are hard to get captured with Uniform.

Let us consider the binary classification task of predicting the T0 node's class. In total, there are 1000 nodes of type T0, and 100 nodes of other types. The node attributes for node type R1 and R31 are generated with two different 32-dim multi-variate Gaussian distributions $\mathcal{N}(\mathbf{1}, I)$ and $\mathcal{N}(\mathbf{0}, I)$, where $\mathbf{0}$ indicates 32-dim all-zero vector, $\mathbf{1}$ indicates 32-dim all-one vector, and $I \in \mathbb{R}^{32\times 32}$ is a diagonal identity matrix. Each multi-variate Gaussian takes care of 50 author nodes in each node type. For other node types, we generate their node instances' attributes with the 32-dim multi-variate Gaussian $\mathcal{N}(\mathbf{1}, I)$.

In ComHetG, we connect the entities with the following rules. First of all, as we aim to classify the type-T0 nodes, we start by describing how type-T0 nodes interact with others. To add edges between nodes of type T0 and T1, connections are not allowed if the T0-type node is of class 0 and the T1-type node instance is generated with $\mathcal{N}(\mathbf{1}, I)$, or the T0-type node is of class 1 and the T1-type node instance is generated with $\mathcal{N}(\mathbf{0}, I)$. Then with this pre-requisite, for each type-T0 node instance, we uniformly randomly select 3 T1 node instances to add the edges. For a node with type T1 T30, for each type-T0 node instance, we uniformly randomly select 3 node

instances of each type, and add the connections. Then, we clarify how to connect a pair of node instances without any type-T0 nodes. For a node pair of type (T1, T31), we only allow connections if the feature of these two nodes are generated with the same multi-variate Gaussian, and for each node type of T1, we uniformly randomly select 3 node instances of qualified T31-type nodes and get them connected. For a node pair of type (T1, T31), we only allow connections if the feature of these two nodes are generated with the same multi-variate Gaussian, and for each node type of T1, we uniformly randomly select 3 node instances of qualified T31-type nodes and get them connected. For a node pair of (T2, T31), for each node instance of type T2, if most of its connected type-T0 nodes belong to class 1, then we define the candidate sampling set as all the T31-type nodes whose features are generated with distribution $\mathcal{N}(\mathbf{1}, I)$, on the contrary, we define the candidate sampling set as all the T31-type nodes whose features are generated with distribution $\mathcal{N}(\mathbf{0}, I)$. Then, we uniformly randomly select 3 node instances from the candidate sampling set, and build the connections. For other node pairs of type (Ti, Tj) where $1 \leq i \leq 30$ and $31 \leq j \leq 50$, for each node of type Ti, we uniformly randomly select 3 instances of type Tj, and add the edges between the instance pair. Then, with these connection rules, we know that the most important neighbor node we should capture for T0 nodes would be the type T1 nodes, type-T2 neighbors are the second important neighbor type, as they can act as a bridge between the T0-type nodes and the critical T31-type nodes. For T1 and T2 nodes, the most important neighbor type is both T31.

Suppose our sampling budget is 10, Figure 6.3(a) provides the computational graph constructed with Uniform. The neighbor type distribution remains the same as in the un-sampled NoiHetG, but the computational graph fails to capture the author nodes in the sampled neighborhood for paper nodes. Figure 6.3(b)(c) also provides

the computational graph constructed with other two different samplers (TAUniform and BLADS, we will give details in later sections), by comparing these computational graphs, we can observe the limitations of Uniform under such kinds of noisy HetGs more clearly.

Next, we examine a modified version of Uniform, which we refer to as Type Aware Uniform or TAUniform, that ensures each neighbor type is equally represented. TAUniform can be regarded as applying the GraphSAGE [38] sampler on the connected instances of each different node type in a node's neighborhood. For an entity v in HetG with $|\mathcal{N}_t(v)|$ number of neighbors of type t, when we sample the neighbor nodes of type t, each of its type-t neighbor nodes can get sampled with probability $1/|\mathcal{N}_t(v)|$. As shown in Figure 6.3(b), applying TAUniform for HetG sampling can address the limitation of Uniform under the aforementioned challenge case 1, as it guarantees each neighbor type is sampled, so the important neighbor type can be captured even it only has a few instances.

However, TAUniform also has its own significant limitation. Suppose we have a schema-complex HetG that includes a large number of connected neighbor types. To apply TAUniform, the sampling budget should be relatively large to ensure each neighbor type is covered in the sampling process. This means that some of the sampling budgets are spent on neighbor types that do not contribute to the downstream task since TAUnifrom treats each neighbor type equally. Thus, applying TAUniform for the schema-complex HetGs makes less efficient use of the sampling budget and limits the effectiveness of using sampling to significantly reduce the overall computation cost.

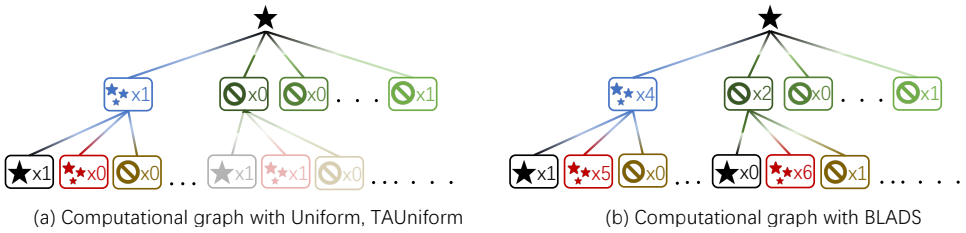


Figure 6.4: Computational Graphs Generated with Different Sampler on ComHetG.

6.3.1.2 Case 2: Schema-Complex HetG

To present this challenging case explicitly, we generate a synthetic graph called ComHetG (Abbr. for Complex HetG) with the schema shown in Figure 6.2(b), note that, due to the considerable amount of entity and relation types, we only provide a partial schema that covers all the critical entity and relation types, and omit others with dots. As this ComHetG includes 51 node types and 330 relation types, we only present partial schema with key node types and relation type, and neglect the others. Let us consider the binary classification task of predicting the T0 node's class. For the current case, we set our sampling budget as 10, and Figure 6.4(a) shows the computational graph constructed with TAUniform, note that, due to the huge number of node types and relation types, we only present partial computational graphs of key neighbor node types. Then, due to the sampling budget constraint, the computational graph constructed with TAUniform fails to capture the T31-type nodes in the sampled neighborhood for T1-type nodes, and also fails to include T2-type nodes in the T0-type nodes' neighborhood, which disables some important information to pass from the valuable T31-type nodes to our target T0-type nodes through T2-type nodes. Therefore, the Heterogeneous GNN model trained on this sampled HetG is weaker. Figure 6.3(b) also provides the computational graph constructed with the BLADS

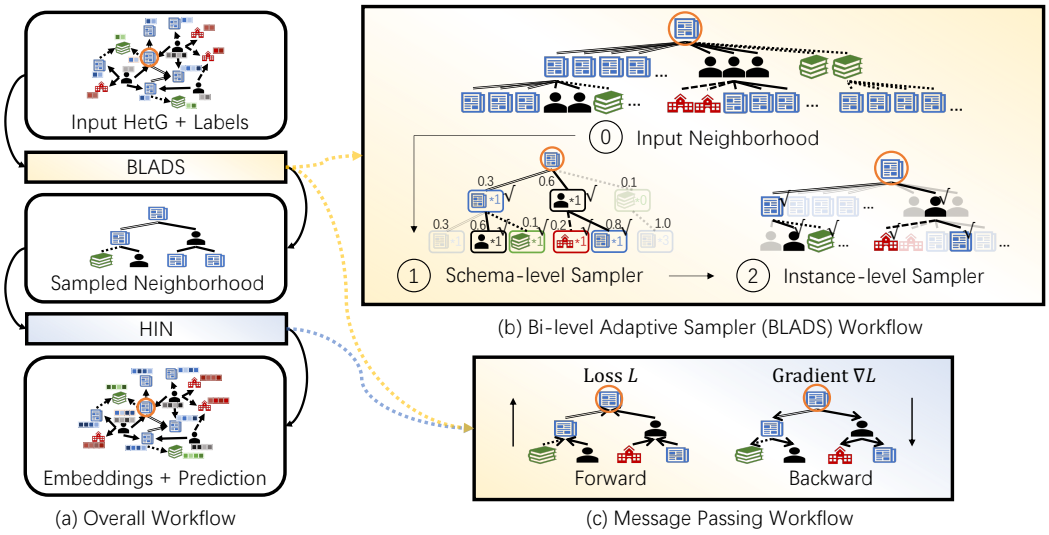


Figure 6.5: An Illustration for Applying BLADS With a Heterogeneous GNN. Schema is Taken From OGBN-MAG in Fig6.1.

sampler, and we should note that ComHetG, Uniform, and TAUniform have the same number of neighbors of different types. By comparing these computational graphs, we can observe the limitations of TAUniform under such kinds of schema-complex HetGs clearly.

6.3.2 BLADS: Bi-Level Adaptive Sampling

In the following, we present our BLADS method in detail. First, in the current subsection, we describe the workflow of the BLADS sampler and illustrate how it generates samples and constructs the computational graph. The next sections will cover how BLADS works together with Heterogeneous GNNs and how to optimize BLADS and Heterogeneous GNNs. The overall workflow is described in Figure 6.5, in which subfigure (a)presents the overall workflow, subfigure (b) presents how

the BLADS sampler works in the form of a computational graph, since the full neighborhood of the target node (with an orange circle) is too big, we only present a partial neighborhood in the input neighborhood and in the instance-level sampler, and subfigure (c) presents the message passing flow, this forward and backward process will update both Heterogeneous GNNs and BLADS.

The case studies shown above for Uniform and TAUniform, demonstrate that uniform random sampling is not enough for complex HetGs. To address the aforementioned tough cases, our sampler explicitly identifies important neighbor types for each node type. The main idea of our method is to conduct a bi-level sampling. Given the sampling budget $\mathcal{B}$, for each node type t^*, the schema-level sampler will learn to determine the share of each neighbor type $\mathcal{B}_{t^*,t}$ and assign more of the budget to the critical neighbor types, then, for each neighbor type t, the instance-level sampler will generate $\mathcal{B}_{t^*,t}$ neighbor instances, where we have $\sum_t \mathcal{B}_{t^*,t} = \mathcal{B}$. Figure 6.5(b) illustrates the workflow for our proposed BLADS.

6.3.2.1 Schema-level Sampler

The schema-level sampler determines, for each node type sampled in the successor Heterogeneous GNN layer, the share of each neighbor type for the current Heterogeneous GNN layer. In BLADS, we make this schema-level sampler adaptive to the given downstream task, so for each node type, it learns the importance of different neighbor types to get higher-quality sampled neighbors.

To achieve this, first, for each relation type r in HetG, we have a learnable scaler β_r. We require $0 \leq \beta_r \leq 1$ for all r. Then, for each node type t^*, we compute a probability distribution $p_{t^*}(\cdot)$ for all the neighbor type t in t^*'s neighborhood as

follows:

$$p_{t^*}(t) = \beta_{r_{t,t^*}} / \sum_{t' \in \mathcal{N}_{t^*}} \beta_{r_{t',t^*}} \tag{6.1}$$

where r_{t,t^*} indicates the relation type that takes t and t^* as source and destination types respectively, and $\mathcal{N}_{t^*}$ indicates the neighbor type set for node type t^*. Then, with this sampling policy, as shown in Figure 6.5(b)'s schema-level Sampler, BLADS can learn different sampling probabilities (which can be interpreted as importance score) on each neighbor type, and then can distribute the budget to each neighbor type accordingly. We may extend the schema-level Sampler by enabling a learnable β_r^l for each layer, but in this work, we fix the schema-level sampling parameters for all layers, and leave the flexible version for future work.

6.3.2.2 Instance-level Sampler

For each node instance v of type $\Phi(v)$ sampled in the successor Heterogeneous GNN layer, for each of its neighbor type t, with the budget $\mathcal{B}_{\Phi(v),t}$ determined by the schema-level sampler, the instance-level sampler selects $\mathcal{B}_{\Phi(v),t}$ instances from node v's type-t neighbors for the current Heterogeneous GNN layer. In BLADS, different from the schema-level sampler which we design to be adaptive, the instance-level sampler applies the uniform distribution for sampling that gives each neighbor node of the same node type the same chance to get selected. This generates diverse instances and alleviates efficiency concerns as learning a customized sampling distribution for each node instance becomes expensive and runs contrary to the goal of reducing the total running time.

6.3.2.3 Sampling with BLADS

The schema-level Sampler and the Instance-level sampler make up BLADS. For each node instance v of type $\Phi(v)$ sampled in the successor Heterogeneous GNN layer, the probability for its neighbor node u of $\Phi(u)$-type to get sampled in the current Heterogeneous GNN layer would be:

$$p(v|u) = p_{\Phi(v)}(\Phi(u)) * 1/|\mathcal{N}_{\Phi(v)}(v)| \tag{6.2}$$

which is the probability to sample type $\Phi(u)$ multiplies the probability to sample the instance u among the type-$\Phi(u)$ neighbors.

6.3.3 Combining BLADS with Heterogeneous GNNs

In this subsection, we explain how can BLADS be applied to Heterogeneous GNNs, and present the forward and backward process of the training workflow. First of all, we describe the backbone Heterogeneous GNN model in detail. In this work, we use the classic RGCN [82] model to present the concrete explanation, but BLADS is designed for any message-passing-based Heterogeneous GNNs where each network layer uses the one-hop neighborhood.

For node v, let $h_v^{(l+1)} \in \mathbf{R}^{d_{l+1}}$ be its learned embedding at layer $l+1$ with dim d_{l+1}, then, the RGCN-layer has the following form:

$$h_v^{(l+1)} = \sum_{r \in \mathcal{R}} \sum_{u \in N_r(v)} \frac{1}{c_{v,r}} \alpha_r^{(l+1)} h_u^{(l)} \tag{6.3}$$

where $c_{v,r}$ is a problem-specific constant that can be customized for normalization purposes, and $\alpha_r^{(l+1)} \in \mathbf{R}^{d_l \times d_{l+1}}$ is the learnable parameter that transforms the current hidden representation of node u.

6.3.3.1 Forward

With our BLADS sampler, the RGCN backbone model will use the sampled heterogeneous neighborhood to update the hidden embedding learned at each layer. Besides, we set $c_{v,r}$ as a constant value for each v, r, to enable a more important relation type with more samples to have a larger influence during the learning process. Let $\mathcal{N}_r^{(S)}(v)$ be the sampled set of neighbor nodes connected to v through type-r relation, the RGCN layer after BLADS sampling has the form in the following. Figure 6.5(c) presents this forwarding message passing process.

$$h_v^{(l+1)} = \frac{1}{\mathcal{B}} \sum_{r \in \mathcal{R}} \sum_{u \in N_r^{(S)}(v)} \alpha_r^{(l+1)} h_u^{(l)} \tag{6.4}$$

6.3.3.2 Backward

After the forward process, then we update the parameters BLADS sampler and the backbone RGCN model with the gradient back-propagation shown in Figure 6.5(c). However, we may encounter a challenge here since the sampling operation is non-differentiable. To address this issue, we use the widely applied log derivative trick [104]. Let $\mathcal{L}$ be the loss function for the downstream task, with the chain rule, we can derive the gradients for the BLADS sampler in the following:

$$\frac{d\mathcal{L}}{d\beta_r} = \frac{d\mathcal{L}}{dh_v^{(l+1)}} \frac{dh_v^{(l+1)}}{d\beta_r} \tag{6.5}$$

To compute the second term and obtain the gradient of $h_v^{(l+1)}$ w.r.t. β_r, we could

have the derived gradient in the following:

$$\begin{aligned}\frac{dh_v^{(l+1)}}{d\beta_r} &= \frac{d\frac{1}{\mathcal{B}}\sum_{r\in\mathcal{R}}\sum_{u\in N_r^{(S)}(v)}\alpha_r^{(l+1)}h_u^{(l)}}{d\beta_r} \\ &= \frac{d(\mathbf{E}_{u\sim p(u|v)}[\alpha_r^{(l+1)}h_u^{(l)}])}{d\beta_r} \\ &= \frac{d(\mathrm{NORM}(\beta_r)\frac{1}{\mathcal{B}_r}\sum_{u\in N_r^{(S)}(v)}\alpha_r^{(l+1)}h_u^{(l)})}{d\beta_r} \\ &= \nabla_{\beta_r}\mathrm{NORM}(\beta_r)h_{v,r}^{(l+1)}\end{aligned}$$

where $\mathrm{NORM}(\beta_r)$ is the normalized schema-level sampling probability.

6.3.4 Overall Algorithm and Optimization

Algorithm With the BLADS and the forward and backward process described in the previous two sections, the overall workflow of our algorithm is presented in Fig 6.5(a). Given a HetG, the BLADS sampler will first sample neighbors for each node at each layer based on the manually specified sampling budget, then the Heterogeneous GNN will use message passing over the sampled subgraph to compute the node embeddings and loss, and finally, both the Heterogeneous GNN and BLADS parameters get updated with backpropagation. We summarize this process in Algorithm 3.

Optimization For the node classification task, we minimize the cross-entropy loss:

$$\mathcal{L}_{NC} = -\sum_{v\in\mathcal{V}_{(t_{tar},l)}} y_v ln\hat{y}_v \tag{6.6}$$

For the link prediction task, we follow the work of [82] and minimize the cross-entropy loss through negative sampling:

$$\mathcal{L}_{LP} = -\sum_{(u,v)\in\mathcal{E}_S} \mathbb{1}_{(u,v)\in\mathcal{E}_r} log\frac{\exp(h_u^{LT}\cdot h_v^L)}{\sum_{u'\in\mathcal{V}}\exp(h_u'^{LT}\cdot h_v^L)} \tag{6.7}$$

Algorithm 3 Training RGCN With BLADS

Input: a minibatch $\{v_i\}$, sample budget $\mathcal{B}$

Output: embedding $h_{v_i}^{(L)}$ on each node, importance score β_r on each relation r

Training for one batch: $\mathcal{V}_0 = \{v_i\}$

for $l = 0$ **to** $L - 1$ **do**

    $\mathcal{V}_{l+1} = \{\}$

    for $v_i \in \mathcal{V}_l$ **do**

        Sample $\mathcal{N}^{(S)}(v)$ with BLADS

        $\mathcal{V}_{l+1} = \mathcal{V}_{l+1} \cup \mathcal{N}^{(S)}(v)$

        Update embedding $h_{v_i}^{(l)}$ with Formula 6.4

    end for

    Compute loss for downstream task

    Update RGCN with backpropagation

    Update BLADS with backpropagation

end for

To optimize the BLADS sampler and the RGCN backbone model, we consider an alternative optimization process, in which we optimize the BLADS sampler for one epoch, and optimize the RGCN for four epochs.

The BLADS sampler relies on the quality of the learned embeddings which is low at the beginning of training. Thus, we split the sampling budget $\mathcal{B}$, and take $\gamma\mathcal{B}$ samples from BLADS and $(1-\gamma)\mathcal{B}$ samples from a uniform random sampler, where γ is the weight coefficient. There are two extreme cases, $\gamma = 0$ corresponds to TAUniform, while $\gamma = 1$ corresponds to using BLADS for the full budget. At the beginning of the training process, it is better to focus on the Heterogeneous GNNs learning, therefore, we use a dynamic γ that increases during the training process, and set $\gamma = \frac{n_{\text{EPO}}}{N_{\text{EPO}}}$, where n_{EPO} is the current training epoch id and N_{EPO} is the total number of training epochs.

Complexity. BLADS with RGCN has a time complexity of $\mathcal{O}(bd\mathcal{B}^L + bd^2\mathcal{B}^{L-1})$, where d is the average dimension of the intermediate representations, and b is the batch size. Compared to the mini-batch full-neighborhood RGCN whose complexity is $\mathcal{O}(bd\mathcal{D}^L + bd^2\mathcal{D}^{L-1})$ where $\mathcal{D}$ is the average degree for the full-neighborhood, applying BLADS would lead to a great complexity reduction as we usually have $\mathcal{B} < \mathcal{D}$. Compared to the Uniform and TAUniform sampling, BLADS has the same time complexity.

6.4 Experiments

In this section, we evaluate our BLADS sampler with the node classification and link prediction tasks.

Dataset	# Entity Type	# Relation Type	# Entity	# Relation	Splits	# Target Node Class
OGBN-MAG	4	4	1,939,743	21,111,007	629,571/64,879/41,939	349
Freebase	8	36	180,098	1,057,688	24%/6%/70%	7
WN18	N/A	18	40,943	151,442	141,442/5,000/5,000	N/A
FB15K	N/A	1,345	14,951	592,213	483,142/50,000/59,071	N/A

Table 6.2: Benchmark Dataset Statistics

6.4.1 Experimental Setup

Datasets. We evaluate the BLADS sampler on the following two widely-used benchmark graphs and two synthetic graphs for the *node classification task*: OGBN-MAG [44] is an academic network whose target node type is paper and the node classification task is to predict the paper's venue. Freebase [114, 71] is a huge knowledge graph in which we set the book nodes as targets and aim to classify their category. We also evaluate BLADS on the two synthetic datasets NoiHetG and ComHetG as described in section 6.3.1. As for the *link prediction task*, we consider evaluating BLADS on two benchmark graphs: WN18 [82] which is a subset of the WordNet that consists of lexical relations between words, and FB15K [82] which is a subset of the Freebase dataset. We summarize all the dataset statistics in Table 6.2.

Baselines. We compare BLADS against the Uniform and TAUniform sampler to demonstrate the benefits of an adaptive bi-level sampler. Besides, we also compare to the full-neighborhood training to demonstrate the efficiency of BLADS.

Other Experimental Set-up. For OGBN-MAG and the two synthetic datasets, we report the test accuracy, and for the Freebase dataset, we report the macro-F1 and micro-F1 scores. For the two link prediction datasets WN18 and FB15K, we report MRR, Hits@1, Hits@3, and Hits@10 as the standard evaluation in [82]. For each dataset, we regard the model with the best validation performance as the "best

	ogbn-MAG	Freebase		NoiHetG	ComHetG
	Acc	macro-F1	Micro-F1	Acc	Acc
RGCN	42.31±0.54	45.27±0.90	52.14±1.34	87.60±0.40	92.10±0.35
UniRanS	39.78± 0.71	36.60±1.29	46.13±1.47	80.45±0.94	80.55±1.28
TAUniRanS	41.33±0.63	39.42±1.17	48.92±1.26	83.4±0.62	80.55±1.28
BLADS	41.49±0.88	43.02±1.59	51.22±1.62	87.20±0.99	91.25±1.06

Table 6.3: Results for Node Classification Task.

model" and evaluate its performance on the test dataset. We train the model 5 times and record the mean and standard deviation of the performance metric. For the RGCN backbone model, we consider a 2-layer network. We set the sampling budget to be 30 for each of the benchmark datasets.

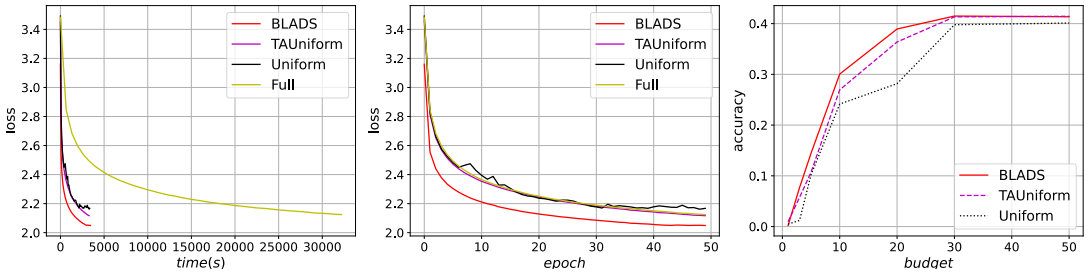
6.4.2 Results

We present our main results of BLADS and baseline models for the node classification task in Table 6.3[1], and for the link prediction task in Table 6.4. As shown in the results, BLADS outperforms Uniform and TAUniform sampler on all the datasets and all the tasks, indicating the importance of assigning budget according to the neighbor type importance. We show the loss convergence in terms of running time and number of epochs in Figure 6.6(a)(b) respectively. We can observe from the curves that BLADS achieves a similar convergence speed as Uniform and TAUniform, and is much faster than full-neighborhood. In the following, we show some ablation studies to provide more detailed views of BLADS, and discuss the importance of budget, weight coefficient, and initialization to BLADS's performance.

[1]We should note that, for the ComHetG, since the relation distribution is the uniform distribution, Uniform and TAUniform are identical in this HetG, and thus have the same results.

	WN18				FB15K			
	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
RGCN	81.63±0.75	66.79±0.79	92.04±0.76	95.58±0.78	65.67±0.57	52.38±0.64	72.44±0.60	81.09±0.62
UniRanS	77.94±0.91	61.83±0.82	88.73±0.87	93.35±0.88	62.09±0.67	49.16±0.73	70.24±0.72	78.85±0.69
TAUniRanS	79.48±0.89	64.50±0.84	89.91±0.86	94.21±0.85	62.80±0.69	50.22±0.70	70.85±0.71	79.94±0.71
BLADS	81.33±0.96	65.12±1.07	91.06±1.04	95.36±1.03	64.12±0.90	50.97±0.87	71.60±0.91	81.17±0.96

Table 6.4: Results for Link Prediction Task



(a) OGBN-MAG Loss (time) (b) OGBN-MAG Loss (epoch) (c) OGBN-MAG acc (budget)

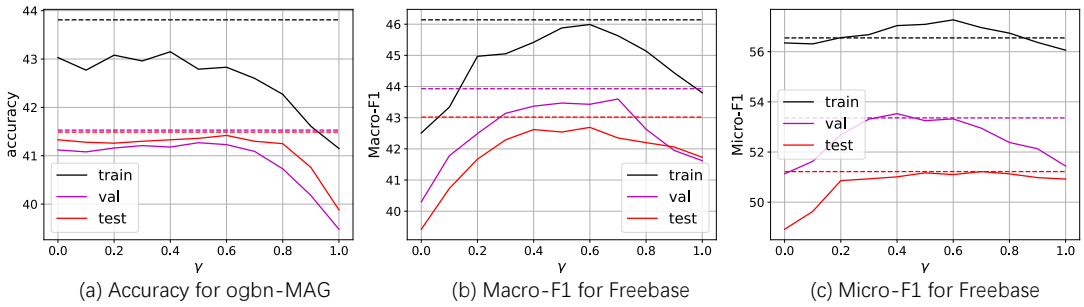
Figure 6.6: (Validation Loss Change In Terms of Wall-time (a) and Epoch (b). The Budget's Effects on the Accuracy (c).

Budget. First, we consider the question: *How does the sampling budget affect BLADS performance?* According to Figure 6.6(c), we observe that in general, increasing the sampling budget improves the performance of all three samplers, because more neighbors bring more information during the message-passing process. Crucially, we find that when the budget is limited, BLADS tends to perform better than the Uniform and TAUniform sampler since it can focus on the important neighbor types.

Weight Coefficient Next, we examine the question: *How does the γ affect BLADS*

	Init1	Init2	Init3	Model	Init1	Init2	Init3
Uniform	36.73	41.41	41.33	P(A,F,P)	(0.40, 0.33, 0.27)	(0.35, 0.3, 0.35)	(0.33, 0.33, 0.33)
BLADS	40.51	41.85	41.49	A(I,P)	(0.93, 0.07)	(0.18, 0.82)	(0.50, 0.50)

Table 6.5: Initialization's Effect on BLADS's Performance.



(a) Accuracy for ogbn-MAG (b) Macro-F1 for Freebase (c) Micro-F1 for Freebase

Figure 6.7: γ's Influence on BLADS Performance. The Dashed Lines Are For the Dynamic Schedule in 3.4. The Solid Lines Are For Results With Different Fixed γ.

performance? With the results shown in Figure 6.7, we find that a fixed $0 < \gamma < 1$ performs better than the extreme cases in which $\gamma = 1$ and $\gamma = 0$. However, a dynamic γ makes BLADS stronger, since by doing so, we can first take advantage of the uniform sampler to obtain a reliable RGCN backbone, then this reliable backbone model would help to learn a more powerful BLADS sampler, and finally, BLADS and the backbone model would be able to enhance each other by providing higher quality samples and higher quality embeddings respectively.

Initialization. Finally, we consider: *How does the schema-level sampling probability initialization affect BLADS performance?* As presented in Table 6.5, we examine the performance of Uniform and BLADS with three different initialization, we find that BLADS is capable of consistently providing competitive performance, while the performance of Uniform sampler largely depends on the initial relation distribution. This demonstrates the power of BLADS in learning the relation importance according to the task regardless of the initialization.

6.5 Conclusion

In this work, we investigate existing sampling approaches for Heterogeneous GNNs, we point out that uniform random sampling is not enough for large-scale heterogeneous graphs with complex schemas and a limited sampling budget, especially in noisy graphs where most instances are not informative for a downstream task. We propose the BLADS sampler that operates at both schema and entity levels to address this problem. The schema-level sampler learns to generate more samples in critical relation types, while the entity-level sampler uses a uniform random sampler for each node at each network layer for speed concerns. According to our experimental results, the BLADS sampler is shown to be able to speed up the Heterogeneous GNNs training without sacrificing the performance on both node classification and link prediction tasks.

CHAPTER 7

Conclusion and Future Directions

This dissertation explores and expands the capabilities of GNNs. We summarize the key contributions regarding solutions to improve GNN effectiveness and efficiency on both homogeneous and heterogeneous graphs as follows:

- We propose the AdaFS framework and the Laplacian regularizer, an innovative solution for soft-selecting optimal graph convolutional filters, greatly enhancing the effectiveness of GNNs on homogeneous graphs and addressing the challenges of over-smoothing and heterophilic challenge.
- We propose the SMASH model as a solution, which adeptly balances the demands of accuracy and scalability for heterogeneous graphs. SMASH performs a 3-stage aggregation which allows it to explore the full metapath set and then identify and focus on the most important metapath for the given task.
- Addressing efficiency for homogeneous GNNs, we provide the DGL-GNN framework which allows for concurrent layer updates and reducing computational costs. This decoupled, greedy learning approach could be integrated with other existing scalability solutions such as sampling, which further improves efficiency.
- We propose the BLADS to enhance heterogeneous GNNs training efficiency. BLADS is a bi-level sampling technique for heterogeneous GNNs that performs

schema-level sampling and instance-level sampling in two stages and is capable of learning the distributions to accommodate the task.

While this dissertation has explored a lot towards a more effective and more efficient GNN, the related research needs further investigation. In terms of effectiveness, developing and deploying GNNs in real-world applications presents notable challenges. Practical training environments are seldom ideal and often include complicating factors such as noisy data (pertaining to features, labels, and graph structures) and out-of-distribution instances. Thus, dedicated efforts are necessary to accommodate these adversities. On the efficiency front, while current research predominantly addresses million- and billion-size graphs, the exploration of training techniques capable of handling trillion-node graphs represents an ambitious and potentially rewarding frontier. Such advancements could set new benchmarks in the scalability and utility of GNN technologies, pushing the boundaries of what is currently achievable.

Bibliography

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019.

[2] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48, 2013.

[3] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei 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, 2019.

[4] Amir Beck and Luba Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM journal on Optimization*, 23(4):2037–2060, 2013.

[5] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled greedy learning of cnns. *arXiv preprint arXiv:1901.08164*, 2019.

[6] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to imagenet. In *International conference on machine learning*, pages 583–593. PMLR, 2019.

[7] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled greedy learning of cnns. In *International Conference on Machine Learning*, pages 736–745. PMLR, 2020.

[8] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[9] Maciej Besta and Torsten Hoefler. Parallel and distributed graph neural networks: An in-depth concurrency analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[10] Rui Bing, Guan Yuan, Mu Zhu, Fanrong Meng, Huifang Ma, and Shaojie Qiao. Heterogeneous graph neural networks analysis: a survey of techniques, evaluations and applications. *Artificial Intelligence Review*, 56(8):8003–8042, 2023.

[11] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[12] Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. *arXiv preprint arXiv:2102.01350*, 2021.

[13] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900, 2015.

[14] Fenxiao Chen, Yun-Cheng Wang, Bin Wang, and C-C Jay Kuo. Graph representation learning: a survey. *APSIPA Transactions on Signal and Information Processing*, 9:e15, 2020.

[15] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568*, 2017.

[16] Jie Chen, Shouzhen Chen, Mingyuan Bai, Jian Pu, Junping Zhang, and Junbin Gao. Graph decoupling attention markov networks for semisupervised graph node classification. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[17] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[18] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR, 2020.

[19] Ting Chen, Song Bian, and Yizhou Sun. Are powerful graph neural nets necessary? a dissection on graph classification. *arXiv preprint arXiv:1905.04579*, 2019.

[20] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266, 2019.

[21] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019.*, pages 257–266, 2019.

[22] Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S Dhillon. Node feature extraction by self-supervised multi-scale neighborhood prediction. *arXiv preprint arXiv:2111.00064*, 2021.

[23] Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1393–1403, 2020.

[24] Andreea Deac, Marc Lackenby, and Petar Veličković. Expander graph propagation. In *Learning on Graphs Conference*, pages 38–1. PMLR, 2022.

[25] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[26] Nima Dehmamy, Albert-László Barabási, and Rose Yu. Understanding the representation power of graph neural networks in learning graph topology. In *Advances in Neural Information Processing Systems*, pages 15387–15397, 2019.

[27] Li Deng and Yang Liu. *Deep learning in natural language processing*. Springer, 2018.

[28] Ivan Derevitskii, Oksana Severiukhina, and Klavdiya Bochenina. Clustering interest graphs for customer segmentation problems. In *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 321–327. IEEE, 2019.

[29] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 135–144, 2017.

[30] Lun Du, Xiaozhou Shi, Qiang Fu, Xiaojun Ma, Hengyu Liu, Shi Han, and Dongmei Zhang. Gbk-gnn: Gated bi-kernel graph neural networks for modeling both homophily and heterophily. In *Proceedings of the ACM Web Conference 2022*, pages 1550–1558, 2022.

[31] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

[32] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1797–1806, 2017.

[33] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*, pages 2331–2341, 2020.

[34] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

[35] Tong Geng, Chunshu Wu, Yongan Zhang, Cheng Tan, Chenhao Xie, Haoran You, Martin Herbordt, Yingyan Lin, and Ang Li. I-gcn: A graph convolutional network accelerator with runtime locality enhancement through islandization. In *MICRO-54: 54th annual IEEE/ACM international symposium on microarchitecture*, pages 1051–1063, 2021.

[36] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[37] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[38] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[39] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[40] Dongxiao He, Chundong Liang, Huixin Liu, Mingxiang Wen, Pengfei Jiao, and Zhiyong Feng. Block modeling-guided graph convolutional neural networks. In

Proceedings of the AAAI conference on artificial intelligence, volume 36, pages 4022–4029, 2022.

[41] Xiaofei He, Deng Cai, and Partha Niyogi. Laplacian score for feature selection. *Advances in neural information processing systems*, 18, 2005.

[42] Yixuan He, Michael Perlmutter, Gesine Reinert, and Mihai Cucuringu. Msgnn: A spectral graph neural network based on a novel magnetic signed laplacian. In *Learning on Graphs Conference*, pages 40–1. PMLR, 2022.

[43] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[44] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[45] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pages 2704–2710, 2020.

[46] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems*, 31, 2018.

[47] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe:

Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[48] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. Scaling up graph neural networks via graph coarsening. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 675–684, 2021.

[49] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International conference on machine learning*, pages 1627–1635. PMLR, 2017.

[50] Dejun Jiang, Zhenxing Wu, Chang-Yu Hsieh, Guangyong Chen, Ben Liao, Zhe Wang, Chao Shen, Dongsheng Cao, Jian Wu, and Tingjun Hou. Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models. *Journal of cheminformatics*, 13:1–23, 2021.

[51] Di Jin, Rui Wang, Meng Ge, Dongxiao He, Xiang Li, Wei Lin, and Weixiong Zhang. Raw-gnn: Random walk aggregation based graph neural network. *arXiv preprint arXiv:2206.13953*, 2022.

[52] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. Graph condensation for graph neural networks. *arXiv preprint arXiv:2110.07580*, 2021.

[53] Wei Ju, Zheng Fang, Yiyang Gu, Zequn Liu, Qingqing Long, Ziyue Qiao, Yifang

Qin, Jianhao Shen, Fang Sun, Zhiping Xiao, et al. A comprehensive survey on deep graph representation learning. *Neural Networks*, page 106207, 2024.

[54] Seungbae Kim, Jyun-Yu Jiang, Jinyoung Han, and Wei Wang. Influencerrank: Discovering effective influencers via graph convolutional attentive recurrent neural networks. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 17, pages 482–493, 2023.

[55] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[56] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems*, pages 13333–13345, 2019.

[57] Kezhi Kong, Jiuhai Chen, John Kirchenbauer, Renkun Ni, C Bayan Bruss, and Tom Goldstein. Goat: A global transformer on large-scale graphs. In *International Conference on Machine Learning*, pages 17375–17390. PMLR, 2023.

[58] Runlin Lei, Zhen Wang, Yaliang Li, Bolin Ding, and Zhewei Wei. Evennet: Ignoring odd-hop neighbors improves robustness of graph neural networks. *Advances in Neural Information Processing Systems*, 35:4694–4706, 2022.

[59] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2018.

[60] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns:

Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9267–9276, 2019.

[61] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

[62] Jiayu Li, Tianyun Zhang, Hao Tian, Shengmin Jin, Makan Fardad, and Reza Zafarani. Sgcn: A graph sparsifier based on graph convolutional networks. In *Advances in Knowledge Discovery and Data Mining: 24th Pacific-Asia Conference, PAKDD 2020, Singapore, May 11–14, 2020, Proceedings, Part I 24*, pages 275–287. Springer, 2020.

[63] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM computing surveys (CSUR)*, 50(6):1–45, 2017.

[64] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[65] Xiao Li, Li Sun, Mengjie Ling, and Yan Peng. A survey of graph neural network based recommendation in social networks. *Neurocomputing*, 549:126441, 2023.

[66] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. *arXiv preprint arXiv:1901.01484*, 2019.

[67] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, and Dongrui Fan.

Sampling methods for efficient training of graph convolutional networks: A survey. *IEEE/CAA Journal of Automatica Sinica*, 9(2):205–234, 2021.

[68] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, Dongrui Fan, Shirui Pan, and Yuan Xie. Survey on graph neural network acceleration: An algorithmic perspective. *arXiv preprint arXiv:2202.04822*, 2022.

[69] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. Pick and choose: a gnn-based imbalanced learning approach for fraud detection. In *Proceedings of the web conference 2021*, pages 3168–3177, 2021.

[70] Ziqi Liu, Zhengwei Wu, Zhiqiang Zhang, Jun Zhou, Shuang Yang, Le Song, and Yuan Qi. Bandit samplers for training graph neural networks. *Advances in Neural Information Processing Systems*, 33:6878–6888, 2020.

[71] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1150–1160, 2021.

[72] Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. Simplifying approach to node classification in graph neural networks. *Journal of Computational Science*, 62:101695, 2022.

[73] Hoang Nt and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

[74] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

[75] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114, 2016.

[76] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.

[77] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[78] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

[79] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[80] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

[81] T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

[82] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*, pages 593–607. Springer, 2018.

[83] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[84] Hao-Jun Michael Shi, Shenyinying Tu, Yangyang Xu, and Wotao Yin. A primer on coordinate descent algorithms. *arXiv preprint arXiv:1610.00040*, 2016.

[85] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Hsu, and Kuansan Wang. An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th international conference on world wide web*, pages 243–246, 2015.

[86] Yunchong Song, Chenghu Zhou, Xinbing Wang, and Zhouhan Lin. Ordered gnn: Ordering message passing to deal with heterophily and over-smoothing. *arXiv preprint arXiv:2302.01524*, 2023.

[87] Stavros Souravlas, Sofia Anastasiadou, and Stefanos Katsavounis. A survey on the recent advances of deep community detection. *Applied Sciences*, 11(16):7179, 2021.

[88] Yizhou Sun and Jiawei Han. Mining heterogeneous information networks: principles and methodologies. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 3(2):1–159, 2012.

[89] Yizhou Sun and Jiawei Han. Mining heterogeneous information networks: a structural analysis approach. *Acm Sigkdd Explorations Newsletter*, 14(2):20–28, 2013.

[90] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003, 2011.

[91] Shyam A Tailor, Javier Fernandez-Marques, and Nicholas D Lane. Degree-quant: Quantization-aware training for graph neural networks. *arXiv preprint arXiv:2008.05000*, 2020.

[92] Jian Tang, Meng Qu, and Qiaozhu Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1165–1174, 2015.

[93] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 807–816, 2009.

[94] Shanshan Tang, Bo Li, and Haijun Yu. Chebnet: Efficient and stable constructions of deep neural networks with rectified power units via chebyshev approximations. *arXiv preprint arXiv:1911.05467*, 2019.

[95] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[96] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[97] Cheng Wan, Youjie Li, Ang Li, Nam Sung Kim, and Yingyan Lin. Bns-gcn: Efficient full-graph training of graph convolutional networks with partition-parallelism and random boundary node sampling. *Proceedings of Machine Learning and Systems*, 4:673–693, 2022.

[98] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. Improving graph attention networks with large margin-based constraints. *arXiv preprint arXiv:1910.11945*, 2019.

[99] Tao Wang, Di Jin, Rui Wang, Dongxiao He, and Yuxiao Huang. Powerful graph convolutional networks with adaptive propagation mechanism for homophily and heterophily. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 4210–4218, 2022.

[100] Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and S Yu Philip. A survey on heterogeneous graph embedding: methods, techniques, applications and sources. *IEEE Transactions on Big Data*, 9(2):415–436, 2022.

[101] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The world wide web conference*, pages 2022–2032, 2019.

[102] Yewen Wang, Ziniu Hu, Yusong Ye, and Yizhou Sun. Demystifying graph neural network via graph filter assessment. 2019.

[103] Oliver Wieder, Stefan Kohlbacher, Mélaine Kuenemann, Arthur Garon, Pierre Ducrot, Thomas Seidel, and Thierry Langer. A compact review of molecular property prediction with graph neural networks. *Drug Discovery Today: Technologies*, 37:1–12, 2020.

[104] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pages 5–32, 1992.

[105] Zhen Hao Wong, Hansi Yang, Xiaoyi Fu, and Quanming Yao. Loss-aware curriculum learning for heterogeneous graph neural networks. *arXiv preprint arXiv:2402.18875*, 2024.

[106] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[107] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019.

[108] Tong Tong Wu, Kenneth Lange, et al. Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics*, 2(1):224–244, 2008.

[109] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[110] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[111] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[112] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.

[113] Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 1287–1292. IEEE, 2022.

[114] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. Heterogeneous network representation learning: A unified framework with survey and benchmark. *IEEE Transactions on Knowledge and Data Engineering*, 34(10):4854–4873, 2020.

[115] Liqi Yang, Linhan Luo, Lifeng Xin, Xiaofeng Zhang, and Xinni Zhang. Dagnn: Demand-aware graph neural networks for session-based recommendation. *arXiv preprint arXiv:2105.14428*, 2021.

[116] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018.

[117] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable

pooling. In *Advances in neural information processing systems*, pages 4800–4810, 2018.

[118] Minji Yoon, Théophile Gervet, Baoxu Shi, Sufeng Niu, Qi He, and Jaewon Yang. Performance-adaptive sampling strategy towards fast and accurate graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2046–2056, 2021.

[119] Ronghui You, Shuwei Yao, Hiroshi Mamitsuka, and Shanfeng Zhu. Deepgraphgo: graph neural network for large-scale, multispecies protein function prediction. *Bioinformatics*, 37(Supplement_1):i262–i271, 2021.

[120] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2127–2135, 2020.

[121] Le Yu, Leilei Sun, Bowen Du, Chuanren Liu, Weifeng Lv, and Hui Xiong. Heterogeneous graph representation learning with relation awareness. *arXiv preprint arXiv:2105.11122*, 2021.

[122] Le Yu, Leilei Sun, Bowen Du, Chuanren Liu, Weifeng Lv, and Hui Xiong. Heterogeneous graph representation learning with relation awareness. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[123] Lingfan Yu, Jiajun Shen, Jinyang Li, and Adam Lerer. Scalable graph neural networks for heterogeneous graphs. *arXiv preprint arXiv:2011.09679*, 2020.

[124] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J

Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

[125] Seongjun Yun, Minbyul Jeong, Sungdong Yoo, Seunghun Lee, Sean S Yi, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks: Learning meta-path graphs to improve gnns. *arXiv preprint arXiv:2106.06218*, 2021.

[126] Hanqing Zeng and Viktor Prasanna. Graphact: Accelerating gcn training on cpu-fpga heterogeneous platforms. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 255–265, 2020.

[127] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. Decoupling the depth and scope of graph neural networks. *Advances in Neural Information Processing Systems*, 34:19665–19679, 2021.

[128] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

[129] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 793–803, 2019.

[130] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018.

[131] Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old mlps new tricks via distillation. *arXiv preprint arXiv:2110.08727*, 2021.

[132] Shichang Zhang, Atefeh Sohrabizadeh, Cheng Wan, Zijie Huang, Ziniu Hu, Yewen Wang, Jason Cong, Yizhou Sun, et al. A survey on graph neural network acceleration: Algorithms, systems, and customized hardware. *arXiv preprint arXiv:2306.14052*, 2023.

[133] Jialin Zhao, Yuxiao Dong, Ming Ding, Evgeny Kharlamov, and Jie Tang. Adaptive diffusion in graph neural networks. *Advances in neural information processing systems*, 34:23321–23333, 2021.

[134] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

[135] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. Robust graph representation learning via neural sparsification. In *International Conference on Machine Learning*, pages 11458–11468. PMLR, 2020.

[136] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. Distdgl: distributed graph neural network training for billion-scale graphs. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 36–44. IEEE, 2020.

[137] Xin Zheng, Yixin Liu, Shirui Pan, Miao Zhang, Di Jin, and Philip S Yu.

Graph neural networks for graphs with heterophily: A survey. *arXiv preprint arXiv:2202.07082*, 2022.

[138] Yizhen Zheng, He Zhang, Vincent Lee, Yu Zheng, Xiao Wang, and Shirui Pan. Finding the missing-half: Graph complementary learning for homophily-prone and heterophily-prone graphs. In *International Conference on Machine Learning*, pages 42492–42505. PMLR, 2023.

[139] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[140] Kaixiong Zhou, Xiao Huang, Daochen Zha, Rui Chen, Li Li, Soo-Hyun Choi, and Xia Hu. Dirichlet energy constrained learning for deep graph neural networks. *Advances in Neural Information Processing Systems*, 34:21834–21846, 2021.

[141] Yu Zhou, Haixia Zheng, Xin Huang, Shufeng Hao, Dengao Li, and Jumin Zhao. Graph neural networks: Taxonomy, advances, and trends. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(1):1–54, 2022.

[142] Hao Zhu and Piotr Koniusz. Simple spectral graph convolution. In *International conference on learning representations*, 2020.

[143] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in neural information processing systems*, 33:7793–7804, 2020.

[144] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in neural information processing systems*, 32, 2019.