Geometric Learning on Graph Structured Data

W. O. K. Asiri Suranga Wijesinghe

A thesis submitted for the degree of
Doctor of Philosophy
The Australian National University

December 2022

– 22 December 2022

Certificate Of Authorship/ Originality

I, Asiri Wijesinghe, declare that this thesis is submitted in fulfilment of the requirements for the degree of Doctor of Philosophy, in the School of Computing at the Australian National University.

I certify that this thesis is my own original work, except where otherwise indicated. I also certify that this document has not been submitted for obtaining an award at any other academic institution.

Dedicated to the people in Sri Lanka, who contributed towards the free education I received for 17 years.

– 22 December 2022

Acknowledgments

I would like to express my sincere gratitude and respect to my primary supervisor A/prof. Qing Wang, for her continuous support, guidance and patience throughout my doctoral research. She has been a tremendous mentor for me and without her encouragement and constant feedback, this wouldn't have been achievable. It has been a great privilege to have you as my supervisor and I will forever be grateful to you for your immense support during my PhD study.

I would like to extend my deepest appreciation to my other supervisors, Prof. Stephen Gould, Prof. Brendon McKay and Dr. Yu Lin who had provided me with great insights. Thank you for sharing your expertise towards completion of my research work.

My sincere thanks goes to all my colleagues from ANU for their support in various ways. It has been fun working with them and I will always cherish the memories we had working together. I would also like to acknowledge with appreciation, the help I received from admin team in College of Engineering and Computer Science, HDR team and IT support in ANU.

My humble gratitude goes to my supervisors and lecturers from University of Colombo School of Computing, Sri Lanka for their encouragement and endorsements which helped me to pursue higher studies.

Last but not least, I would like to show my deepest gratitude to my parents Lal Wijesinghe and Padmini Wijesinghe, my wife Dilmi Jayasena, siblings Dhanushka Wijesinghe and Isuru Wijesinghe and my relatives for their unconditional support and patience during the difficult times throughout my life.

Publications

Primary Publications

Contributions from work presented in this thesis have been published across multiple peer reviewed top-tier conferences. A list of publications in chronological order is given below:

- A. Wijesinghe and Q. Wang. A New Perspective on "How Graph Neural Networks Go Beyond Weisfeiler-Lehman?". The 10th International Conference on Learning Representations (ICLR), 2022.

  The concepts and formulations of this spatial Graph Neural Network model, theoretical analysis, model architecture and evaluations of this paper are presented in Chapter 3.

- A. Wijesinghe and Q. Wang. DFNets: Spectral CNNs for Graphs with Feedback-Looped Filters. The 33rd Conference on Neural Information Processing Systems (NeurIPS), 2019.

  The concepts and formulations of this spectral Graph Neural Network model, theoretical analysis, model architecture and evaluations of this paper are presented in Chapter 4.

- A. Wijesinghe and Q. Wang. Dynamic PageRank for Graph Neural Networks. Under review.

  The concepts and formulations of this diffusion Graph Neural Network model, theoretical analysis, model architecture, representational power, convergence properties and evaluations of of this model are presented in Chapter 5.

- A. Wijesinghe, Q. Wang and S. Gould. A Regularized Wasserstein Framework for Graph Kernels. The 21st IEEE International Conference on Data Mining (ICDM), 2021.

  The concepts and formulations of this regularized optimal transport graph kernel framework, theoretical analysis, algorithm implementation and evaluations of this paper are presented in Chapter 6.

Secondary Publications

One more paper is published as a join work with other collaborators. However, contributions from this publication is not closely related and not presented in this thesis. This paper is:

- J. Shao, Q. Wang, A. Wijesinghe and E. Rahm. ERGAN: Generative Adversarial Networks for Entity Resolution. The 20th IEEE International Conference on Data Mining (ICDM), 2020.

Abstract

Graphs provide a ubiquitous and universal data structure that can be applied in many domains such as social networks, biology, chemistry, physics, and computer science. In this thesis we focus on two fundamental paradigms in graph learning: representation learning and similarity learning over graph-structured data. Graph representation learning aims to learn embeddings for nodes by integrating topological and feature information of a graph. Graph similarity learning brings into play with similarity functions that allow to compute similarity between pairs of graphs in a vector space. We address several challenging issues in these two paradigms, designing powerful, yet efficient and theoretical guaranteed machine learning models that can leverage rich topological structural properties of real-world graphs.

This thesis is structured into two parts. In the first part of the thesis, we will present how to develop powerful Graph Neural Networks (GNNs) for graph representation learning from three different perspectives: (1) spatial GNNs, (2) spectral GNNs, and (3) diffusion GNNs. We will discuss the model architecture, representational power, and convergence properties of these GNN models. Specifically, we first study how to develop expressive, yet efficient and simple message-passing aggregation schemes that can go beyond the Weisfeiler-Leman test (1-WL). We propose a generalized message-passing framework by incorporating graph structural properties into an aggregation scheme. Then, we introduce a new local isomorphism hierarchy on neighborhood subgraphs. We further develop a novel neural model, namely *GraphSNN*, and theoretically prove that this model is more expressive than the 1-WL test. After that, we study how to build an effective and efficient graph convolution model with spectral graph filters. In this study, we propose a spectral GNN model, called *DFNets*, which incorporates a novel spectral graph filter, namely *feedback-looped filters*. As a result, this model can provide better localization on neighborhood while achieving fast convergence and linear memory requirements. Finally, we study how to capture the rich topological information of a graph using graph diffusion. We propose a novel GNN architecture with dynamic PageRank, based on a learnable transition matrix. We explore two variants of this GNN architecture: *forward-euler solution* and *invariable feature solution*, and theoretically prove that our forward-euler GNN architecture is guaranteed with the convergence to a stationary distribution.

In the second part of this thesis, we will introduce a new optimal transport distance metric on graphs in a regularized learning framework for graph kernels. This optimal transport distance metric can preserve both local and global structures between graphs during the transport, in addition to preserving features and their local variations. Furthermore, we propose two strongly convex regularization terms to theoretically guarantee the convergence and numerical stability in finding an optimal assignment between graphs. One regularization term is used to regularize a Wasserstein distance

between graphs in the same ground space. This helps to preserve the local clustering structure on graphs by relaxing the optimal transport problem to be a cluster-to-cluster assignment between locally connected vertices. The other regularization term is used to regularize a Gromov-Wasserstein distance between graphs across different ground spaces based on degree-entropy KL divergence. This helps to improve the matching robustness of an optimal alignment to preserve the global connectivity structure of graphs. We have evaluated our optimal transport-based graph kernel using different benchmark tasks. The experimental results show that our models considerably outperform all the state-of-the-art methods in all benchmark tasks.

Contents

List of Figures

List of Tables

Notation and Terminology

Notations	Meaning
$G = (V, E, \mathbf{A})$	An undirected and weighted graph G with the set of vertices V, edges E and an adjacency matrix $\mathbf{A}$
$\mathcal{N}(v)$	Set of neighbors of a vertex v
S_v	Neighborhood subgraph of a vertex v
S_{vu}	Overlap subgraph between vertices v and u
$S_i \simeq_{subgraph} S_j$	Subgraph-isomorphism between S_i and S_j
$S_i \simeq_{overlap} S_j$	Overlap-isomorphism between S_i and S_j
$S_i \simeq_{subtree} S_j$	Subtree-isomorphism between S_i and S_j
h_v	Feature vector of a vertex $v \in V$
$\{\!\{h_v \mid v \in \mathcal{N}(i)\}\!\}$	A multiset of feature vectors of neighborhoods of vertex v
$\mathcal{S}^*$	Set of overlap subgraphs in G
$\omega(S_v, S_{vu})$	Structural coefficients for each vertex v and its neighbors
$\mathbf{X}$	A matrix with node features (graph signal matrix)
$m_a^{(t)}$	A message aggregated from the neighbors of v and their structural coefficients
$m_v^{(t)}$	An adjusted message from vertex v to account for structural effects from its neighbors
$h_v^{(t+1)}$	New feature vector by combining $m_v^{(t)}$ and $m_a^{(t)}$
$W^{(t)}, \theta_1^{(t)}, \theta_2^{(t)}$	Learnable weight matrices
σ	A non-linear activation function
$\alpha_{vu}^{(t)}$	Attention coefficient between vertex v and u
$\tilde{A}$	A matrix of normalized structural coefficients
x	Graph signal
$\mathbf{D}$	Degree matrix
$\mathbf{I}$	Identity matrix
$\mathbf{L}$	Graph Laplacian matrix
$\mathbf{U}$	Graph Fourier basis
$\mathbf{U}^{\mathrm{H}}$	Hermitian transpose of $\mathbf{U}$
Λ	Diagonal eigenvalue matrix
λ_{min}	Smallest eigenvalue
λ_{max}	Largest eigenvalue
$\hat{x}$	Graph Fourier coefficient
$h(\lambda_i)$	Frequency response of spectral filters
$\hat{\mathbf{A}}$	Adjacency matrix after adding self-loop

– Continued from the previous page

Notations	Meaning
$\hat{\mathbf{L}}$	Normalized Laplacian matrix
$\tilde{\mathbf{L}}$	Scaled-normalized Laplacian matrix
b	Bias
ψ, ϕ	Vectors of complex coefficients
λ_{cut}	Cut-off frequency
$\acute{e}(\tilde{\lambda}_i)$	Frequency response error between desired and actual frequency responses
ϵ	A parameter that controls the trade-off between convergence efficiency and approximation accuracy
$y(t)$	A random walk distribution at a time point t
$x(t)$	Time-dependent teleportation vector
$\mathbf{P}$	PageRank transition matrix
y_{ppr}	Personalized PageRank vector
y_{hk}	Heat kernel vector
α	PageRank damping factor
$\mathbf{Z}_i$	A dynamic PageRank diffusion scheme
$\oplus$	Vector concatenation
$\tilde{\pi}$	A stationary distribution
$\Delta(\mathbf{X})$	Local variation matrix of vertex features of a graph
μ	A prior marginal distribution of graph G_1
ν	A prior marginal distribution of graph G_2
$\mathbf{1}_n$	A n-dimensional vector of ones
$\pi(\mu, \nu)$	A set of probabilistic couplings between graphs G_1 and G_2
$\mathbf{C}$	A cost function matrix which measures the cost of moving a probability mass from μ to ν
e_i	An embedding vector of i^{th} vertex
$\mathbf{C}^N(i,j)$	A neighbourhood similarity matrix between e_i and e_j, where *i-th* vertex of G_1 and the *j-th* vertex of G_2
$\mathbf{C}^P(i,j)$	A pairwise similarity matrix of a graph between the node embeddings of the *i-th* vertex and the *j-th* vertex
$LW(\mu, \nu)$	Local barycentric Wasserstein distance
$\langle \cdot, \cdot \rangle_F$	Frobenius dot product
γ	Optimal transport coupling matrix
$\mathbf{E}_\mu$	A node embedding matrix of distribution μ
$\hat{\mathbf{E}}_\mu$	Local barycentric embedding matrix
$\Omega_\mu(\gamma)$	Spatially localized barycentric term
$\nabla^2\Omega_\mu(\gamma)$	Hessian of $\Omega_\mu(\gamma)$
$\otimes$	Kronecker product
$\odot$	Hadamard product
$GW(\mu, \nu)$	Global connectivity Wasserstein distance
$KL(\gamma \Vert \gamma')$	KL divergence between two distributions
$RW(\mu, \nu)$	Regularized Wasserstein (RW) discrepancy

– Continued from the previous page

Notations	Meaning
$\Delta\gamma$	Descent direction
$H(\gamma)$	A composite objective function
$\nabla H(\gamma)$	Gradient of $H(\gamma)$
$\mathbf{K}_{\mu\nu}$	A regularized Wasserstein kernel (RWK)
δ_i	Suboptimality gap

Table 1: A list of notations.

Chapter 1

Introduction

1.1 Background

Graphs are indispensable mathematical objects that allow us to model complex relationships (i.e., edges) between entities (i.e., nodes). With the advancement of science, graphs have been receiving considerable attention in various disciplines, including social science, chemistry, physics, medicine, bioinformatics, computer science, and many other related fields. For instance, molecular graphs, drug-drug interaction networks, friendship networks, recommender systems, transportation networks, protein-protein interaction networks, document link networks, finite element meshes, and 3D scene graphs are some examples from different domains [10, 91, 32, 187, 94].

In the last two decades, machine learning methods have achieved an enormous success in analyzing graph-structured data [294, 268, 131]. Graph learning plays a vital role to capture useful insights from hidden patterns of networks in many real-world applications. For instance, molecular property prediction in chemistry, advertising a new product to users in recommender systems, 3D object detection in computer graphics & computer vision, social interaction prediction in social science, and traffic forecasting in transportation [74, 268, 131, 180, 288, 236, 227, 200]. Typically, graph learning tasks compute low-dimensional vectors for entities or an entire graph by considering node features and structural properties that relate to the local neighborhood of entities and their relationships. Then. these low-dimensional vectors can be used to perform prediction on a desired learning task on graphs.

There are two fundamental learning problems in machine learning on graphs: (1) graph representation learning, and (2) graph similarity learning. The main idea of graph representation learning is to learn low-dimensional vector representations for graphs (or their nodes) that reflect the properties of interests, e.g., local neighborhood structure. The main idea of graph similarity learning is to learn a similarity function that enables to measure how similar or different pairs of graphs are in a low-dimensional vector space.

1.1.1 Graph Representation Learning

Representation learning is not a new paradigm, which has been explored since past several decades in various domains such as natural language processing, signal

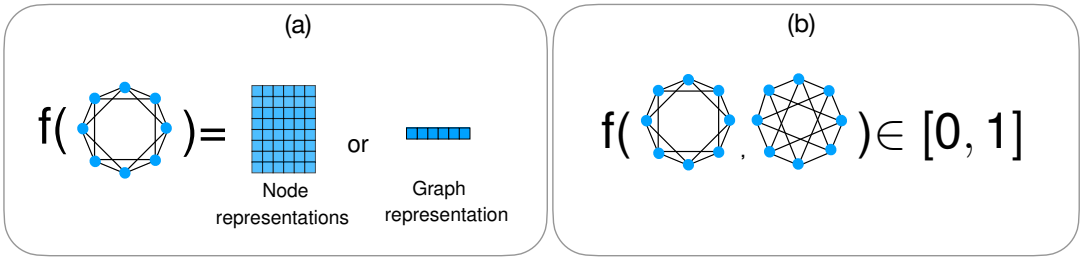


Figure 1.1: (a) Representation learning learn a function that encodes each node or entire graph structure into a low-dimensional embedding; (b) Similarity learning learn a similarity function that measures the similarity between pairs of graphs.

processing, and computer vision by machine learning research community [14]. The evolution of the field of representation learning [27] has been influenced by the studies on the Euclidean domain. Studies of the Euclidean domain are concerned about the geometry of flat surfaces such as grids in a 1-dimensional (1D) or 2-dimensional (2D) space, which follows a simple relational regular structure. Audio signals and text are good examples of 1D grid structured data that can be modelled by Recurrent Neural Networks (RNNs); RNNs can encode sequential relationships to build representations for signals and texts on speech recognition and machine translation tasks [282, 213, 139, 51, 50, 45]. On the other hand, images and videos are good examples for 2D grid structured data that can be modelled by Convolution Neural Networks (CNNs); CNNs are parameterized neural networks with spatial locality and shift-invariance properties to learn representations for images on image classification and image segmentation & reconstruction [260, 206, 287, 142].

Traditional machine learning approaches for graph-structured data primarily rely on graph statistics such as clustering coefficients or node degrees [19], or use hand-craft features for local neighborhood structure [147]. These approaches have several limitations. For example, they are inflexible and cannot adapt to the learning process; extracting graph statistics may be expensive and time consuming. A number of studies have been devoted to address these issues through learning representations that encode the structural information of graphs. More concretely, graph representation learning learns embeddings for entities such that structural relationships between entities are preserved in an embedding space. Then, these learned graph representations can be used as features for downstream machine learning tasks such as node classification [283], link prediction [52, 61], clustering [41, 132], and visualization [32]. Take link prediction [52, 61] in social networks for example, pairwise properties between nodes are encoded into node representations such that similarities between nodes are preserved.

The existing graph representation learning methods generally fall into two categories: (1) shallow embedding methods, and (2) deep learning methods [57, 32].

Shallow embedding methods. The central idea of shallow embedding methods is to map graph entities into low-dimensional latent embeddings such that local neigh-

borhood similarities between entities are preserved [196, 241, 257, 258]. Previously, there was a growing interest to map high dimensional structural properties into a lower dimensional vector space by taking into consideration short random walks over a graph. A number of works have been developed such as DeepWalk, LINE, and node2vec [196, 241, 95] which linearize a given graph using random walks. However, these methods have several limitations. DeepWalk [196] uses uniform random walks for finding neighbors; hence, it cannot control over the explored neighborhoods. In LINE [241], the authors proposed a breadth-first strategy to sample nodes such that the likelihood is maximized independently over 1-hop and 2-hop neighbors. Hence, LINE has no flexibility in exploring nodes in further depths. In node2vec [95, 201], it avoids these limitations by exploring graph neighborhoods through the higher order biased random walks. However, node2vec takes into account truncated random walks to capture the local neighborhoods of nodes. Hence, this method ignores long-distance global relationships in a graph [293]. Furthermore, node2vec cannot capture different semantics that associate with attributes of a given source node and has substantial runtime overhead with large networks [293].

Later, Liu et al. [154] proposed a novel algorithmic framework called *scalable attribute aware network embedding with locality* (SANE) to learn joint graph representations from both node attributes and graph topology. However, this method can lead to some misalignment between an attribute space and a topological space. Thus, it may give rise to an issue that a linear mapping between the attribute and topological spaces is not preserved. In this study, the authors used one of the classical methods called *locally linear embedding* (LLE) [212, 154] to embed attributes. The LLE defines neighbors of a source node using a collection of nodes with similar attributes, which is also known as *k-nearest neighbors* [241, 212]. The complexity of this framework is at least quadratic to the number of nodes in a graph, which is inefficient when a graph is large. Also, the LLE fails on noisy data and outliers. Thus, smoothness of a dataset is a critical factor for LLE-based analysis, and it is less accurate in preserving global pairwise similarity. The locality assumption of SANE may not be valid and comparable when considering a large number of neighbors.

Another line of shallow embedding methods is to learn embeddings from matrix factorization, using the traditional dimensional reduction technique, called *Laplacian eigenmaps*, [13]. This is a simple encoder-decoder method. Following this work, a number of methods have been proposed for learning graph embeddings using a pair-wise inner product decoder [3, 35, 188]. Meng et al. [202] introduced robust node representations for graphs with multiple views. Inspired by this method, several methods have been proposed for multi-view graph representation learning, such as multi-view matrix factorizing methods [92, 231] and multi-view clustering methods [150, 149, 238, 295]. However, those methods have some limitations such as lack of weight learning and insufficient collaboration of views to find a robust node representation [202]. Hence, a collaboration framework that can overcome those limitations was suggested, which is able to integrate different views to vote for robust node representations and also implicitly learn voting weights of each view through an attention-based approach. However, there is an inherent problem that how to fully

leverage the multi-view information on graphs. The search strategy is also rigid and lacks scalability for large networks.

In the early stages, most of graph representation models were based on shallow architectures. As information process of a graph is often non-linear and complicated [57], linear functions may not be sufficient to map a graph into a vector space. There has been a growing interest in developing deep neural models for graph representation learning since deep neural models have the ability to capture non-linear structures on graph-structured data and approximate an arbitrary objective. Compared with shallow embedding methods using linear functions, deep neural models can improve the robustness and also retrieve highly compressed powerful representations for graph embeddings through non-linear models [91].

Deep learning methods. Deep neural models on graphs were initially inspired by the work proposed by Sperduti et al. [235]. In this work, features are extracted from graphs by a recursive linear aggregation scheme with a non-linear activation function. Later, Baskin et al. [11] introduced a parameter sharing model that considers transformation invariants on node and edge features. Gori et al. [90] and Scarselli et al. [218] introduced new neural network models that can recursively aggregate the neighborhood feature information with RNNs. Following these two models, to improved the efficiency and convergence, Li et al. [144] introduced a gated graph sequence network that refines the information propagation of RNNs with a gating mechanism. These recursively neighborhood feature aggregation methods are also known as *message-passing neural networks* (MPNNs) or *spatial Graph Neural Networks* (spatial GNNs).

In the past several years, due to the rising trends in network analysis and prediction, generalizing MPNNs to graphs has attracted considerable interest [82, 247, 99, 216, 157, 63]. Xu et al. [275] proposed a message-passing neural network architecture, namely *graph isomorphism networks* (GIN) to analyze the expressive power of the MPNNs, which builds connections with the Weisfeiler-Lehman test (1-WL test). Following this method, one interesting challenge is to design an expressive, yet efficient spatial message-passing aggregation scheme to go beyond the 1-WL test. In general, there are three main directions of extending GNNs beyond the 1-WL test: (1) building higher-order GNNs based on higher-order WL algorithms (i.e. k-WL with $k \geq 3$) or their variants [167, 176, 175]; (2) counting on pre-defined substructures as additional node features [23]; (3) augmenting node identifiers or random features into GNNs [280, 250, 217]. However, these methods still have some limitations. We can obtain more powerful GNN models via higher order WL methods such as k-GNN [175], which nevertheless require high computational overheads and are not really useful in practice. If we have some prior knowledge about an application (e.g., triangles and cliques are useful in social networks), we can count topological features such as triangles and cliques, and add them into node features. This is useful for some applications; however, the problem is that this requires domain expertise for different applications and we have to know what kind of local structures could be useful for different applications in advance. Augmenting node identifiers or random features

for GNNs are either difficult or too costly to ensure permutation invariance.

Previously, Bruna et al. [28] introduced a notion of spectral convolution on graphs, which is indirectly defined as convolution operation through the spectral graph theory [53]. Following this study, a number of methods have been proposed to improve, extend and approximate these spectral convolutions [27, 60, 65, 71, 99, 122, 171, 299], which has led to state-of-the-art performance on several benchmark tasks such as semi-supervised node classification task on spectral GNNs. The key idea behind these spectral GNNs on designing spectral graph filters is to approximate the frequency responses of graph filters using a polynomial function (e.g. Chebyshev filters [65]) or a rational polynomial function (e.g. Cayley filters [137] and ARMA [20]). Polynomial filters are sensitive to changes in the underlying graph structure. They are also very smooth and can hardly model sharp changes. Rational polynomial filters are more powerful to model localization, but they often have to trade off with computational efficiency, resulting in higher learning and computational complexities as well as numerical instability.

Despite the success of spectral GNNs and spatial GNNs, there is an another perspective on GNNs, called *diffusion GNNs*, which is based on the graph diffusion process [128, 125, 291]. One of the main problems in GNNs is to develop powerful models that can capture rich and varying graph structures, thereby being able to obtain better representations for homophily and heterophily graphs. Although we can apply a neighborhood message aggregation scheme for homophlic graphs since locally connected vertices on a graph share the same class labels, it is difficult to generalize a neighborhood message aggregation scheme for heterophilic graphs because locally connected vertices on a graph have different class labels. In recent years, a number of works have addressed this issue by developing diffusion GNNs that can adapt to both homophilic or heterophilic graphs [49, 297, 72]. Due to the use of a standard PageRank setting, these methods have several drawbacks. Standard PageRanK restricts the landing probabilities to 1-hop neighbors; on the other hand, these landing probabilities are predefined and fixed. Thus, most of diffusion GNNs cannot capture long-range dependencies of a graph into the probability transition matrix of PageRank.

1.1.2 Graph Similarity Learning

The popularity of graph similarity learning has been degraded after the rise of graph representation learning, but graph similarity learning still remains an appealing paradigm for comparing pairs of graphs. Generally, similarity learning is also known as *metric learning*, which learns a function to measure the similarity or distance between pairs of objects. A number of methods for graph similarity learning have been proposed from different domains such as computer vision, information retrieval, neuroscience, bioinformatics, chemoinformatics, and natural language processing [169, 97, 115, 136, 148, 107, 21, 160, 234, 290]. Face recognition in computer vision, protein-protein networks analysis for disease diagnosis in bioinformatics, molecular graphs classification in chemoinformatics, brain networks neurological disorder diagnosis

in neuroscience, and text similarity learning in natural language processing are few examples of similarity learning applications.

Previously, a number of works have emerged which study different graph similarity metrics such as graph edit distance [30, 265], maximum common subgraph [31, 255, 207] and full-graph or sub-graph isomorphism [66, 16, 224, 276] in order to address the graph similarity search and graph matching problems. Generally, the computation cost of these similarity metrics is NP-complete [281]. Despite the fact that some heuristic and pruning methods have been introduced to speed up computation, these heuristic methods are inefficient for large graphs and their suboptimal solutions are unbounded [281]. To address these limitations of graph similarity learning methods, the most commonly considered solution is to first employ a mapping function that can map graphs into a low-dimensional vector space, and then apply kernelized machine learning algorithms such as support vector machines (SVMs) to classify pairs of graphs (i.e., graph kernels).

Inspired by the success of optimal transport theory, optimal transport has been applied in a variety of fields such as computer vision [78], image processing [77], and neural networks [5]. An optimal transport distance [252] compares the similarity between two probability distributions by incorporating a ground distance into the underlying geometric metric space, aiming to capture the geometric nature of these probability distributions. A graph can be viewed as a discrete probability distribution in some geometric metric space; therefore, optimal transport can be used to measure the similarity between two graphs.

According to whether optimal transport techniques have been used in computing graph kernels, we group graph kernel methods into two categories: (1) non-optimal-transport graph kernels (traditional graph kernels): non-optimal-transport graph kernels focus on comparing similarity of graphs based on their substructure patterns such as subtree, cycles, shortest paths, graphlets, and etc. [107, 22, 225, 226], and (2) optimal-transport-based graph kernels: optimal-transport techniques are used to explore the geometric nature of graphs by viewing graphs as discrete distributions in a geometric metric space when computing their graph similarity. There are several limitations in non-optimal-transport graph kernels. One of the main limitations is that, they require substructures to be pre-defined based on domain expertise, which is not always available in practice. Further, they often ignore topological structure and feature distributions on graphs. In recent years, optimal-transport graph kernels are emerging as a more popular research direction [182, 245, 164]. Nonetheless, existing optimal-transport graph kernels still have some limitations. Optimal-transport graph kernels primarily ignore the connection between topological structures and feature information in their transport plans, and also do not consider the local clustering structures of graphs. Furthermore, they still suffer from preserving global connectivity structures of graphs during the transportation.

1.2 Research Objectives

Graphs are irregular with complex relationships between nodes. This brings up new challenges on how to develop powerful machine learning models that can effectively handle intrinsic structural properties of graph structured data. Specifically, there are three key challenges that we will address in this thesis:

- How to improve the representation power of machine learning models on graphs?
- How to effectively design machine learning models to preserve intrinsic properties of graphs?
- How to ensure theoretical properties that can guarantee the convergence and numerical stability for machine learning techniques on graphs?

To address these challenges, in this thesis, we explore the solutions for the following two research objectives:

(1) We will design expressive yet simple and efficient GNNs that can learn powerful representations for distinguishing the graph structured data;

(2) We will develop a powerful graph kernel function that can preserve local intrinsic properties and reliably measure the similarity between pairs of graphs with theoretically guaranteed convergence.

1.3 Contributions

In this thesis, we aim to address the aforementioned research objectives. We study graph representation learning from three different aspects of graph neural network (GNN) architectures: (1) spectral GNNs, (2) spatial GNNs, and (3) diffusion GNNs. Then we propose a graph similarity learning method, which aims to build a kernel function that can incorporate intrinsic graph properties into a learning framework. Below, the main contributions of this thesis are summarized:

Contribution I: We propose expressive yet simple spatial GNNs that can go beyond the 1-WL test with a theoretically provable guarantee.

Contribution II: We design a new class of spectral graph filters and incorporate it into an effective, yet efficient spectral GNN architecture.

Contribution III: We propose two GNN architectures based on dynamic PageRank to capture rich and varying graph structures, i.e, homophily and heterophily.

Contribution IV: We develop a regularized optimal transport graph kernel that can preserve intricate structures on graphs with theoretically guaranteed convergence.

In the following, we elaborate on these contributions one by one.

1.3.1 Spatial Graph Neural Networks

The first problem that we study in this thesis is to design expressive yet simple GNNs that can go beyond the 1-WL test with a theoretically provable guarantee. We propose a new perspective on designing powerful GNNs without sacrificing computational simplicity and efficiency. In this work, we introduce a GNN framework namely, generalized message-passing (GMP) framework, which enables a general way of injecting structural information into a message-passing aggregation scheme. Our main contributions of this work lie in three folds.

- Firstly, we develop a new hierarchy of local isomorphism on neighborhood subgraphs. There exist a natural class of isomorphic graphs, which strictly lies in between neighborhood subgraph isomorphism and neighborhood subtree isomorphism, which is known as overlap subgraph isomorphism. The general idea is that if two graphs are isomorphic in terms of subgraph isomorphism, then these two graphs are isomorphic in terms of overlap isomorphism. On the other hand, if two graphs are isomorphic in terms of overlap isomorphism, then they must be isomorphic in terms of subtree isomorphism.

- Secondly, we develop a generalized message-passing scheme for GNNs by incorporating structural information. This generalized message-passing scheme can inject local structural information via structural coefficients into a message-passing aggregation scheme to learn the representations for vertices, compared with the standard message passing aggregation schemes [100].

- Thirdly, we propose a novel GNN model called *GraphSNN* for graph learning and prove that GraphSNN is more expressive than the 1-WL test in distinguishing graph structures. The message-passing aggregation scheme of GraphSNN is an instantiation of our generalized message-passing scheme. A specific kind of structural coefficients is designed and then carefully incorporated into a message-passing aggregation scheme to ensure the injectivity of a function for message-passing aggregation over neighbourhood subgraphs.

1.3.2 Spectral Graph Neural Networks

The main idea behind this work is to design a spectral graph filter by approximating the eigendecomposition using a rational polynomial function [118, 137].

Graph filters can generally be divided into three different groups. The first one is basis dependent filters, which corresponds to the traditional spectral filters [171]. There are several limitations in these filters. One is high computational complexity since they require to perform eigendecomposition explicitly and apply graph Fourier transform. The parameter complexity is linear with respect to the number of vertices and it is also hard to generalize across graphs. The second group of graph filters are polynomial filters [65], which can mitigate some issues of basis dependent filters. A polynomial filter can approximate the eigendcomposition as a polynomial function. These polynomial filters are efficient and guarantee the stability

under graph perturbation. This reduces the parameter complexity to constant time and improves the localization on graph filters. However, polynomial filters are very smooth and hard to model sharp changes. The last group is rational polynomial filters [137, 20], which are more powerful to model localization on graphs. However, they have higher learning and computational complexities as well as numerical instabilities. Existing rational polynomial filters can accept the narrow band of frequencies. Thus, these rational polynomial filters cannot capture the better characteristic properties of a graph. That is because, according to spectral graph theory [53], characteristic properties of a graph depend on graph frequencies.

Our work is built upon rational polynomial filters. We propose a new class of spectral graph filters, known as *feedback-looped filters*, which enable a better localization due to its rational polynomial form. The design of feedback-looped filters alleviates the matrix inversion of rational polynomial form through a recursive function. Therefore, our feedback-looped filters have linear convergence time and linear memory complexity w.r.t the number of edges of a graph. Specifically, feedback-looped filtering contains two filtering components: *feedforward* and *feedback* filters. The *feedforward* filtering employs k-hop localized polynomial filters, and *feedback* filtering refines the k-hop localized features that captured by the *feedforward* filtering to ameliorate the accuracy. We formulate a convex constrained optimization problem to learn optimal coefficients of feedback-looped filters and propose a new spectral convolutional layer with feedback-lopped filters.

To avoid numerical instabilities and gradient vanishing or exploding during the training, we propose two techniques: *scaled-normalization* and *cut-off frequency*. The *scaled-normalization* technique helps to reduce the spectral radius bound of Laplacian to alleviate numerical instability of a feedback-looped filter. The *cut-off frequency* technique helps to accept a wider range of frequencies in order to alleviate the narrow band frequency issue of rational polynomial filters. These techniques together enable feedback-looped filters to better capture characteristic properties of graphs.

1.3.3 Diffusion Graph Neural Networks

We introduce a novel GNN architecture in this work to capture the rich topological information of a graph using graph diffusion. According to the literature, we can categorize existing diffusion based GNNs into three different groups. The first one is, homogeneous isotropic diffusion, which is direction independent and is treated in the same way in everywhere. The second one is non-homogeneous isotropic diffusion, which is position dependent and direction independent, which can be treated as an attention mechanism [38]. The third one is non-homogeneous anisotropic diffusion, which is both position and direction dependent [12]. In this work, we propose a novel homogeneous isotropic diffusion GNN. We build the connection with dynamic PageRank and introduce two different solutions:

(1) *Forward Euler solution*: a simple and fast approach that reflects spatial dependencies between a current node with its long-range neighborhood dependencies to build the connection with a GNN message passing aggregation scheme;

(2) *Invariable feature solution*: a flexible approach that generalizes both personalised PageRank and heat kernel.

These two methods can effectively incorporate the dynamic PageRank to employ the different local structures on a graph. We incorporate a new learnable PageRank transition matrix, which helps to encode local topological information with long range dependencies. This alleviates several limitations in the standard PageRank model. Thus, we can improve the generalization capability of our diffusion GNN architecture on homophilic and heterophilic graphs.

1.3.4 Regularized Optimal Transport Graph Kernels

The last problem that we study in this thesis is to develop a powerful and theoretically guaranteed graph kernel framework with regularized optimal transport. In this work, we introduce a novel optimal transport based distance metric on graphs, namely *Regularized Wasserstein (RW) discrepancy*. This RW discrepancy regularizes the optimal transport learning problem by two *strongly convex* regularization terms to compute a distance between graphs.

- One regularization term is used to regularize a Wasserstein distance between graphs in the same ground space. This regularization term helps to preserve the local clustering structure on graphs by relaxing the optimal transport problem to be a cluster-to-cluster assignment between locally connected vertices.
- The other regularization term is used to regularize Gromov-Wasserstein distance between graphs across different ground spaces with a degree-entropy KL divergence term. This regularization term helps to improve the matching robustness of an optimal alignment to preserve the global connectivity structure of graphs.

Due to the strong convexity of these two regularization terms, we can find an optimal assignment between graphs by theoretically guaranteeing the convergence and numerical stability. Our optimal transport problem also considers the feature local variations on graphs, which measure how features change with respect to the underlying neighborhood structure of a graph. Thus, feature similarity matrices of our optimal transport problem can capture features and their local variations into a cost function. Therefore, this regularized optimal transport problem can preserve both local and global structure of graphs during the transport.

1.4 Thesis Outline

The rest of this thesis is organised as follows. In Chapter 2, we provide a comprehensive literature review of the related work studied in this thesis. Then, the main contributions of this thesis are arranged into Chapters 3-6 in two parts. In the first part of this thesis, we present how we develop powerful GNNs for graph representation learning from three different perspectives. In Chapter 3, we study the

problem of enhancing the expressive power of the spatial GNNs that can go beyond the 1-WL. In Chapter 4, we study the problem of designing an efficient spectral GNN with a new class of graph filters. In Chapter 5, we study the problem of designing diffusion GNNs with dynamic PageRank to capture rich and varying graph structure. In the second part of this thesis, we introduce a optimal transport based graph kernel method for graph similarity learning. In Chapter 6, we study the problem of designing a powerful learning framework for graph kernels with regularized optimal transport, which is theoretically robust with guaranteed convergence. For each of Chapters 3-6, we start with an overview of the specific problems studied in the chapter, and then propose the model architecture, theoretical analysis, experimental results and a summary of the chapter. We finally conclude the thesis and discuss future research opportunities in Chapter 7.

Chapter 2

Related Work

This chapter provides the background and related work. Specifically, this chapter elaborates different types of GNNs and kernel methods, i.e., spatial GNN methods, spectral GNN methods, diffusion GNN methods, and graph kernel methods, which have been developed in the literature for graph classification and node classifications.

2.1 Graph Representation Learning

GNNs belong to a class of Neural Network (NN) models, which are used to process the graph structured data. In this section, we review the recent research in graph representation learning from three different perspectives: (1) spatial GNNs, (2) spectral GNNs, and (3) diffusion GNNs.

2.1.1 Spatial Graph Neural Networks

A number of works have been undertaken in the literature to expand message-passing neural networks (MPNNs) for arbitrarily graph structured data. The early stages of message-passing GNNs were motivated by the work done by Sperduti et al. [235], who proposed a neural architecture with recursive neural networks for directed acyclic graphs. Initially, the notion of message-passing GNNs was outlined by Gori et al. [90] and further studied by Scarselli et al. [218], and Gallicchio et al. [82] for generalising the recursive neural networks to apply with more general class of graph structured data such as directed, undirected, and cyclic graphs. In particular, these message-passing GNNs learnt node representations by propagating neighborhood information with an iterative scheme using recurrent graph neural networks (RecGNNs). This iterative scheme terminates when a stable fixed point is reached. However, these neural message-passing schemes cannot be efficiently converged and, they are more expensive on large graphs. To mitigate limitations in RecGNNs, Li et al. [144] proposed a gated graph sequence network by refining the information propagation step with a gating mechanism. However, this approach can be worked well on small graphs and also remains computationally expensive.

Later, Kipf & Welling [122] introduced a simple and efficient aggregation scheme by averaging the neighborhood features with a normalized adjacency matrix. Consequently, Gilmer et al. [85] proposed a general view of GNN with message-passing

aggregation scheme. In this paradigm, each node recursively aggregates feature vectors of its neighboring nodes to compute a new feature vector. Following on these two works, a number of works have been proposed with message-passing GNNs that take graph structured data as input and build a function to compute embeddings for graph nodes by considering the topological structure and features of nodes and edges [247, 99, 216, 157, 63]. These MPNNs are also known as spatial GNNs since they follow a message-passing aggregation scheme to learn low dimensional vector space representations for nodes in a graph by exchanging and aggregating messages from spatially close neighbors. Thus, local structure of each node can be incorporated into the message-passing scheme.

Spatial GNN methods use different message-passing aggregation functions to preserve the spatial locality on a graph. These aggregation functions of spatial GNNs are parametric. For instance, GAT [247] introduced a weighted aggregation scheme with a multi-head self-attention mechanism over neighborhood feature vectors. Hamilton et al. [99] proposed an aggregation scheme by concatenating multiple layers with skip-connection and they used neighborhood sampling techniques to reduce the computation cost.

2.1.1.1 Message-Passing Neural Networks on Graphs

Let $G = (V, E, \mathbf{X})$ be an undirected and weighted graph, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, and $\mathbf{X} \in \mathbb{R}^{|V| \times f}$ be a matrix of input feature vectors where $x_v \in \mathbb{R}^f$ is an input feature vector associated with each $v \in V$. Let $\mathbf{A} \in \mathbb{R}^{|V| \times |V|}$ is an adjacency matrix of G, which encodes the weights of edges. Message-passing aggregation scheme is a basic building block of spatial GNNs, which has two main steps: (1) for each node $v \in V$, a message-passing aggregation scheme recursively aggregates the feature vectors of nodes in the neighborhood of v, and (2) then, the aggregated information combines with the feature vector of v itself to obtain a new representation.

Let $h_v^{(t)}$ be a hidden embedding of node $v \in V$. In each iteration t, a message-passing neural network (MPNN) update the hidden embeddings by aggregating neighborhood information (i.e., $\mathcal{N}(v)$) of node v. We can express the message-passing update rule as follows:

$$\begin{aligned} h_v^{(t+1)} &= \textsc{Combine}^t\Big(h_v^{(t)}, \textsc{Aggregate}^t(h_u^{(t)}, \forall u \in \mathcal{N}(v))\Big) \\ &= \textsc{Combine}\Big(h_v^{(t)}, m_{\mathcal{N}(v)}^{(t)}\Big), \end{aligned} \tag{2.1}$$

where $m_{\mathcal{N}(v))}^{(t)} = \textsc{Aggregate}^t(h_u^{(t)}, \forall u \in \mathcal{N}(v))$, Combine is a differentiable function, Aggregate is a differentiable and permutation invariant function, and $m_{\mathcal{N}(v))}$ is a message that is computed by aggregating the embeddings of neighbors $\mathcal{N}(v)$ of node v. Then Combine function combines the previous embedding $h_v^{(t)}$ of node v and the message $m_{\mathcal{N}(v))}^{(t)}$ to compute the new embedding $h_v^{(t+1)}$ for node v. When $t = 0$, we set

$h_v^0 = x_v$, where x_v refers to the input feature vector for node v as an initial embedding. After completing the K iterations, we set the last layer output as a final embedding for each node. Since there is no natural ordering on nodes, such message-passing aggregation schemes are usually required to be permutation-invariant [166, 119, 83]. Then the readout phase computes an embedding for the entire graph using some readout function over final node embeddings as follows:

$$h_G = \textsc{Readout}\Big(\{h_v^{(t)} | v \in G\}\Big), \tag{2.2}$$

where Readout is a permutation invariant function to the node states in order for achieving MPNN to be invariant to graph isomorphism.

There are two common message-passing techniques [34]: (1) isotropic technique: a neighborhood aggregation function that treats edge weights equally in every edge direction (i.e., GCN [122] and GraphSAGE [99]); (2) anisotropic technique: an anisotropic technique assigns different weights for every edge and performs weighted aggregation of neighborhood features (i.e., GAT [247] and GatedGCN [24]).

Despite advances of spatial GNNs, Xu et al. [275] shows that existing MPNNs are not powerful enough to distinguish some graph structures. The main intuition of this work is that if $f : \mathcal{G} \to \mathbb{R}^z$ is a message-passing GNN function that can map any two non-isomorphic graphs G_i and G_j into different embeddings, then the 1-WL test also identify G_i and G_j are not isomorphic. From the theoretical perspective, the WL hierarchy is used to analyze the expressive power of message-passing GNNs [216]. We will discuss more about the connection between WL hierarchy and message-passing GNNs in the next section.

2.1.1.2 Graph Isomorphism and WL Algorithm

The main goal of graph isomorphism is to determine whether there is an edge preserving bijection between the nodes of two graphs. Specifically, if we have two graphs, we need to check whether these two graphs are structurally equivalent or not. Graph isomorphism problem is neither P or NP-complete [240]. There are many heuristics for graph isomorphism testing. In this thesis, we use the classical graph isomorphism algorithm called 1-WL test.

WL hierarchy is a well-established framework for graph isomorphism tests [93]. Introduced by Weisfeiler and Lehman [262], the first-order WL algorithm (also called 1-WL or color refinement) is a computationally efficient heuristic for testing graph isomorphism [9]. The 1-WL test is an iterative color refinement procedure that starts with the same color for all nodes. Then, the color refinement of 1-WL test follows the following iteration:

$$h_v^{(t+1)} = hash\Big(h_v^{(t)}, \{\!\{h_u^{(t)} : \forall u \in \mathcal{N}(v)\}\!\}\Big), \tag{2.3}$$

where $h_v^{(t)}$ is the color of node v at iteration t, $\mathcal{N}(v)$ is the set of neighbors of node v and $\{\!\{.\}\!\}$ is a multiset of colors.

This color refinement process terminates after finite number of iterations. 1-WL test examines neighbors of each node and then refine their colors based on their neighborhood. In each iteration, the WL test applies recoloring *hash* function. This recoloring *hash* function is designed as an injective function since we need to map two nodes with different multisets into two distinct colors. We refine the colors until the colors of a graph is unchanged anymore between two adjacent iterations. Then, we analyze the colors between two graphs. If 1-WL test outputs two different colors for two different graphs, then these two graphs are non-isomorphic. However, if 1-WL test gives the same colors for two graphs, we cannot draw a conclusion that these two graphs are isomorphic since colors may be equal for two non-isomorphic graphs as well.

WL test can be extended from 1-WL to higher-order WL (k-WL) test, where k refers to the order of the tuple. The general idea is the same for both 1-WL and k-WL tests, but the color refinement procedure of k-WL is different from 1-WL, where 1-WL gives a color to each 1-tuple of node and k-WL gives a color to each k-tuples of nodes. Specifically, multisets of k-WL is defined as all k-tuples of nodes that differ in one element of the target k-tuple of nodes [110]. For each k-tuple, we assign a color, then the color of each k-tuple is refined iteratively based on the color of neighbors of each k-tuple. If we go higher in the hierarchy of k-WL test, it is known that k-WL is strictly more powerful than (k-1)-WL when k$\geq$3 [33, 93].

Message-passing GNNs are typically considered as a differentiable neural generalization of the WL algorithms on graphs. It has been reported [275] that some popular GNNs such as GraphSAGE [99] are at most powerful as 1-WL in distinguishing graph structures. Xu et al. [275] has shown that Graph Isomorphism Network (GIN) is as powerful as 1-WL. At its core, GIN provides an injective aggregation scheme that is defined as a function over multisets of feature vectors, and thus GIN has the representational power to map any two different multisets of feature vectors to different representations in an embedding space.

2.1.1.3 Spatial GNNs Beyond 1-WL

In this section, we discuss the spatial GNN models that can go beyond 1-WL and their limitations. A considerable amount of efforts have been devoted to improve the expressive power of GNNs beyond 1-WL.

Generally, there are three directions: (1) Several works proposed higher-order variants of GNNs that are as powerful as k-WL with $k \geq 3$ [8]. For example, Morris et al. [175] introduced k-order graph networks that are expressive as a set-based variant of k-WL, Maron et al. [167] proposed a reduced 2-order graph network that is as expressive as 3-WL, and Morris et al. [176] proposed a local version of k-WL which considers only a subset of vertices in a neighborhood. However, these more expressive GNNs are impractical to use due to their inherent high computational costs and sophisticated design. (2) Some works attempted to incorporate inductive biases based on isomorphism counting on pre-defined topological features such as triangles, cliques, and rings [23, 155, 172], similar to the traditional ideas of graph kernels [277].

However, pre-defining topological features requires domain-specific expertise, which is often not readily available. (3) Most recently, several works explored the ideas of augmenting GNNs using node identifiers or random features. For example, Vignac et al. [250] proposed a method that maintains a “local context" for each node based on manipulating node identifiers in a permutation equivariant way. You et al. [280] developed ID-GNNs by taking into account the identity information of vertices. Chen et al. [48] and Murphy et al. [177] assigned one-hot IDs to vertices based on the ideas of relational pooling. Sato et al. [217] added a random feature to each node to improve the representational capability of GNNs.

Our spatial GNN model in Chapter 3 is different from existing models by injecting properties of structural interactions among vertices based on a natural class of isomorphic graphs in the local neighborhood (i.e., overlap subgraph isomorphism) into a message-passing aggregation scheme of GNNs.

2.1.2 Spectral Graph Neural Networks

Spectral GNNs are based on the concepts of spectral graph theory [53]. The main idea of spectral GNNs is to define a convolution operation on graphs in the spectral domain. The main intuition for the spectral convolution comes from the graph signal processing domain [229], which relies on spectral graph filters [137]. Spectral graph filters define the convolution operation indirectly on graphs via eigenvalue decomposition of a graph Laplacian [28, 104]. However, eigenvalue decomposition on a graph Laplacian is computationally expensive. To avoid this issue, a number of works [20, 65, 101, 122, 137, 146] have studied the approximation of eigenvalue decomposition by a polynomial or rational polynomial function.

The first notable spectral GNNs was introduced by Bruna et al. [28], which considered a parameterized learnable diagonal matrix as a spectral filter, and extended a convolution operation on graphs. This spectral GNN is not computationally efficient nor localized over k-hop neighborhood. To address this issue, Henaff et al. [104] proposed a spectral filter with parameterized smooth coefficients to obtain the localization. Defferrard et al. [65] proposed ChebNet, based on the truncated Chebyshev polynomial approximation for eigenvalue decomposition. This Chebyshev polynomial filter is exactly localized on k-hop neighborhood. A number of works have been motivated by this work over the last several years and introduced various spectral GNNs. Kipf & Welling [122] proposed a simplified graph convolutional networks (GCNs) by employing first order approximation of the Chebyshev filters. Later, Bianchi et al. [20] introduced convolutional neural networks with auto-regressive moving average (ARMA) filters, which is more powerful for designing the k-hop neighborhood localization on graph structured data since ARMA filter is a rational polynomial function. However, this ARMA convolution is unstable and computationally expensive. We will discuss these spectral filters and convolution operations below.

2.1.2.1 Spectral Convolution on Graphs

We let $n = |V|$ and $m = |E|$. A *graph signal* is a function $x : V \to \mathbb{R}$ and can be represented as a vector $x \in \mathbb{R}^n$ whose i^{th} component x_i is the value of x at the i^{th} vertex in V. The graph Laplacian is defined as $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{D} \in \mathbb{R}^{n \times n}$ is a diagonal matrix with $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$ and $\mathbf{I}$ is an identity matrix. $\mathbf{L}$ has a set of orthogonal eigenvectors $\{u_i\}_{i=0}^{n-1} \in \mathbb{R}^n$, known as the *graph Fourier basis*, and non-negative eigenvalues $\{\lambda_i\}_{i=0}^{n-1}$, known as the *graph frequencies* [53]. $\mathbf{L}$ is diagonalizable by the eigendecomposition such that $\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^H$, where $\Lambda = diag([\lambda_0, \ldots, \lambda_{n-1}]) \in \mathbb{R}^{n \times n}$ and $\mathbf{U}^H$ is a hermitian transpose of $\mathbf{U}$. We use λ_{min} and λ_{max} to denote the smallest and largest eigenvalues of $\mathbf{L}$, respectively.

Given a graph signal x, the *graph Fourier transform* of x is $\hat{x} = \mathbf{U}^H x \in \mathbb{R}^n$ and its inverse is $x = \mathbf{U}\hat{x}$ [215, 229]. The graph Fourier transform enables us to apply graph filters in the vertex domain. A *graph filter* h can filter x by altering (amplifying or attenuating) the graph frequencies as

$$h(\mathbf{L})x = h(\mathbf{U}\Lambda\mathbf{U}^H)x = \mathbf{U}h(\Lambda)\mathbf{U}^H x = \mathbf{U}h(\Lambda)\hat{x}. \tag{2.4}$$

Here, $h(\Lambda) = diag([h(\lambda_0), \ldots, h(\lambda_{n-1})])$, which controls how the frequency of each component in a graph signal x is modified. However, applying graph filtering as in Eq. 2.4 requires the eigendecomposition of $\mathbf{L}$, which is computationally expensive. To address this issue, several works [20, 65, 101, 122, 137, 146] have studied the approximation of $h(\Lambda)$ by a polynomial or rational polynomial function, which we will discuss in the following.

2.1.2.2 Spectral Graph Filters for GNNs

Here, we present several spectral graph filters with polynomial approximation (i.e., Chebyshev and Lanczos filters) and rational polynomial approximation (i.e., Cayley and ARMA filters).

Chebyshev filters. Hammond et al. [101] first proposed to approximate $h(\lambda)$ by a polynomial function with k^{th}-order polynomials and Chebyshev coefficients. Later, Defferrard et al. [65] developed Chebyshev filters for spectral GNNs on graphs. A Chebyshev filter is defined as

$$h_\theta(\tilde{\lambda}) = \sum_{j=0}^{k-1} \theta_j T_j(\tilde{\lambda}), \tag{2.5}$$

where $\theta \in \mathbb{R}^k$ is a vector of learnable Chebyshev coefficients, $\tilde{\lambda} \in [-1, 1]$ is rescaled from λ, the Chebyshev polynomials $T_j(\lambda) = 2\lambda T_{j-1}(\lambda) - T_{j-2}(\lambda)$ are recursively defined with $T_0(\lambda) = 1$ and $T_1(\lambda) = \lambda$, and k controls the size of filters, i.e., localized in k-hop neighborhood of a vertex [101]. Kipf and Welling [122] simplified Chebyshev filters by restricting to 1-hop neighborhood.

Lanczos filters. Liao et al. [146] used the Lanczos algorithm to generate a low-rank matrix approximation $\mathbf{T}$ for the graph Laplacian. They used the affinity matrix

$\mathbf{S} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. Since $\mathbf{L} = \mathbf{I} - \mathbf{S}$ holds, $\mathbf{L}$ and $\mathbf{S}$ share the same eigenvectors but have different eigenvalues. As a result, $\mathbf{L}$ and $\mathbf{S}$ correspond to the same $\hat{x}$. To approximate the eigenvectors and eigenvalues of $\mathbf{S}$, they diagonalize the tri-diagonal matrix $\mathbf{T} \in \mathbb{R}^{m\times m}$ to compute Ritz-vectors $\mathbf{V} \in \mathbb{R}^{n\times m}$ and Ritz-values $\mathbf{R} \in \mathbb{R}^{m\times m}$, and thus $\mathbf{S} \approx \mathbf{V}\mathbf{R}\mathbf{V}^T$. Accordingly, a k-hop Lanczos filter operation is,

$$h_\theta(R) = \sum_{j=0}^{k-1} \theta_j \mathbf{R}^j, \tag{2.6}$$

where $\theta \in \mathbb{R}^k$ is a vector of learnable Lanczos filter coefficients. Thus, spectral convolutional operation is defined as $h_\theta(\mathbf{S})x \approx \mathbf{V}h_\theta(\mathbf{R})\mathbf{V}^T x$. Such Lanczos filter operations can significantly reduce computation overhead when approximating large powers of $\mathbf{S}$, i.e. $\mathbf{S}^k \approx \mathbf{V}\mathbf{R}^k\mathbf{V}^T$. Thus, they can efficiently compute the spectral graph convolution with a very large localization range to easily capture the multi-scale information of the graph.

Cayley filters. Observing that Chebyshev filters have difficulties in detecting narrow frequency bands due to $\tilde{\lambda} \in [-1, 1]$, Levie et al. [137] proposed Cayley filters, based on Cayley polynomials:

$$h_{\theta,s}(\lambda) = \theta_0 + 2Re(\sum_{j=1}^{k-1} \theta_j (s\lambda - i)^j (s\lambda + i)^{-j}), \tag{2.7}$$

where $\theta_0 \in \mathbb{R}$ is a real coefficient and $(\theta_1, \ldots, \theta_{k-1}) \in \mathbb{C}^{k-1}$ is a vector of complex coefficients. $Re(x)$ denotes the real part of a complex number x, and $s > 0$ is a parameter called *spectral zoom*, which controls the degree of "zooming" into eigenvalues in Λ. Both θ and s are learnable during training. To improve efficiency, the Jacobi method is used to approximately compute Cayley polynomials.

ARMA filters. Bianchi et al. [20] sought to address similar issues as identified in [137]. However, different from Cayley filters, they developed a first-order ARMA filter, which is approximated by a first-order recursion:

$$\bar{x}^{(t+1)} = a\tilde{\mathbf{L}}\bar{x}^{(t)} + bx, \tag{2.8}$$

where a and b are the filter coefficients, $\bar{x}^{(0)} = x$, and $\tilde{\mathbf{L}} = (\lambda_{max} - \lambda_{min})/2\mathbf{I} - \mathbf{L}$. Accordingly, the frequency response is defined as:

$$h(\tilde{\lambda}) = \frac{r}{\tilde{\lambda} - p}, \tag{2.9}$$

where $\tilde{\lambda} = (\lambda_{max} - \lambda_{min})/2\lambda$, $r = -b/a$, and $p = 1/a$ [113]. Multiple ARMA_1 filters can be applied in parallel to obtain a ARMA_k filter. However, the memory complexity of k parallel ARMA_1 filters is k times higher than ARMA_1 graph filters.

We make some remarks on how these existing spectral filters are related to each other. (i) As discussed in [20, 137, 146], polynomial filters (e.g. Chebyshev and

Lanczos filters) can be approximately treated as a special kind of rational polynomial filters. (ii) Chebyshev filters can be regarded as a special case of Lanczos filters. (iii) Although both Cayley and ARMA_k filters are rational polynomial filters, they differ in how they approximate the matrix inverse implied by the denominator of a rational function. Cayley filters use a fixed number of Jacobi iterations, while ARMA_k filters use a first-order recursion plus a parallel bank of k ARMA_1.

2.1.3 Diffusion Graph Neural Networks

According to the relationship between random walk and message-passing aggregation scheme [124], GCN [122] converges to a random walk limit distribution when increasing the number of layers. This issue is known as oversmoothing. Previously, Klicpera et al. proposed a diffusion GNN, namely graph diffusion convolution (GDC) [125], by introducing a connection between personalized PageRank (PPR) [189] and random walk limit distribution. PageRank has been used in a variety of graph analysis tasks such as gene and protein ranking in bioinformatics, object ranking in database queries, and analysis correlations of brain activities in Neuroscience [189, 86]. Essentially, PageRank encodes rich topological information via landing probabilities of random walks over graphs [140]. A question that is naturally provoked is whether one can leverage PageRank or more generally graph diffusion to strengthen the representational power of GNNs.

The main building block of diffusion is classical graph diffusion scheme [125, 6]. In recent years, several works have been studied diffusion with PPR and heat kernel since they are easy to derive from classical graph diffusion scheme [125, 291, 124, 49]. We can easily build the connection between spectral filters and graph diffusion by truncating classical graph diffusion, which has been explored in [125].

Recent work explored the possibility of developing new diffusion GNN models based on dynamic systems. Chamberlain et al. [38] proposed GRAND based on temporal spatial discretization of partial differential equation (PDE). They control dynamics by learning a diffusivity parameter, similar to attention based GNNs. Eliasof et al. [72] proposed PDE-GNN by examining the connection between time-dependent PDE and GNNs. Unlike these methods, in this thesis, we focus on homogeneous graph diffusion for GNNs.

2.1.3.1 Diffusion on Graphs with PDEs

The study on PDEs has been explored by many scientists in the past from eighteenth century. A diffusion process can be expressed as a PDE [127, 190]. In many domains, diffusion PDEs are widely considered as a dominant paradigm for modeling the data such as computer graphics [26, 151, 191, 237], image processing [29, 70, 246, 261, 233, 195], and computer vision [17, 39, 37]. In machine learning, PDEs are used to model physics informed learning [145, 228, 56, 214, 205] and neural networks [203, 269, 69, 46, 38].

Diffusion PDEs consist of various discretization schemes to build the connection

with message-passing aggregation scheme. The message-passing aggregation scheme on spatial GNNs is conceptually similar to the discrete diffusion process on graphs [38]. The diffusion process occurs along edges, where difference between spatially close node features is equivalent to the analogy of spatial derivatives. The spatial discretization in the following differentiable equation is intuitive on a graph in a continuous time space,

$$\frac{\partial X(u,t)}{\partial t} = div[\nabla X(u,t)], \tag{2.10}$$

where $\nabla X(u,t)$ is the gradient-flow along edges on node u at time t and div is the divergence aggregation of features along edges. The gradient-flow on a graph is a function based on edges, which allows to obtain difference between feature vectors of target node u and its spatially close neighbors, i.e., $(\nabla X)_{uv} = X_u - X_v$, where node v is an adjacent node of node v. The divergence is the sum of in-flows and out-flows of nodes (sum of edge gradient-flows), i.e., $(div(X))_u = \sum_{v \in \mathcal{N}(u)} W_{uv} X_v$, where W_{uv} is the weight between node u and node v. By using these two concepts together, we can generalize the diffusion process on graphs as follows [242],

$$\frac{\partial X(t)}{\partial t} = div[\mathbf{H}(X(t),t)\nabla X(t)], \tag{2.11}$$

where $\mathbf{H} = diag(a[X_u(t), X_v(t), t]) \in \mathbb{R}^{n \times n}$ is a diagonal matrix and a is a function that determines the similarity between nodes. The term $a[X_u(t), X_v(t), t]$ is time dependent; however in this thesis, we will consider as a time independent similarity function $a[X_u(t), X_v(t)]$ for simplicity. We can reformulate the Eq. 2.11 using the definitions of div and $\nabla X(t)$ as follows:

$$\frac{\partial X(t)}{\partial t} = (\mathbf{A}(X(t)) - \mathbf{I})X(t), \tag{2.12}$$

where $\mathbf{A}(X(t))$ refers to an adjacency matrix of a graph, $X(t)$ refers to a feature matrix, and $\mathbf{I}$ refers to an identity matrix.

There are various numerical methods for solving non-linear diffusion problems. We solve our non-linear diffusion equations using a finite difference method by discretizing the spatial temporal derivative. There are two popular schemes that can be used to discretize a spatial temporal derivative: (1) explicit scheme: this is the most simple way to perform the discretization using a forward euler method; (2) implicit scheme: this is based on the backward time difference to discretize the spatial temporal derivative. In this thesis, we use an explicit scheme to build the connection between diffusion equations and a message-passing aggregation scheme. Let $\mathbf{M}(X^{(t)}) = (\mathbf{A}(X(t)) - \mathbf{I})$ and $s > 0$ be a smaller time step. We have,

$$\frac{X^{(t+1)} - X^{(t)}}{s} = \mathbf{M}(X^{(t)})X^{(t)} \tag{2.13}$$

where t refers to a forward time-step, which is known as a discretization parameter. We can reformulate Eq. 2.13 as $X^{(t+1)} = \left(\mathbf{I} + s\mathbf{M}(X^{(t)})\right)X^{(t)}$. This is a simple iterative

scheme, equivalent to a message-passing aggregation scheme in spatial GNNs.

GNNs can be seen as a discretization of PDEs. Generally, there are three types of diffusion GNNs:

(1) Homogeneous isotropic diffusion (i.e., GCN [122]): diffusion is same in every direction - $\frac{\partial X}{\partial t} = \Delta X$, where $\Delta X = div(\nabla X)$ refers to the Laplacian operator.

(2) Non-homogeneous isotropic diffusion (i.e., GAT [247]): position dependent and direction independent diffusion - $\frac{\partial X}{\partial t} = -div(a\nabla X)$, where a is a scalar value that refers to a learnable position dependent diffusivity parameter.

(3) Non-homogeneous anisotropic diffusion (i.e., directional GNNs [12]): position and direction dependent diffusion - $\frac{\partial X}{\partial t} = -div(A\nabla X)$, where A is a matrix that refers to a learnable position and direction dependent diffusivity parameter.

2.2 Graph Similarity Learning

In the following, we review the recent research in the field of graph similarity learning from two different perspectives: (1) non-optimal transport (non-OT) graph kernels, and (2) optimal transport graph kernels.

Graph kernels play a vital role in bridging the gap between graph structured data and kernel-based methods in machine learning [221] such as support vector machines (SVM), kernel principal component analysis (PCA), or kernel regression [106]. Graph kernels offer an appealing paradigm for measuring the similarity between graphs. In the last several decades, kernel methods have been extended to build graph kernels for solving classification tasks on graphs, which can be categorized into two main classes: non-optimal transport-based graph kernels and optimal transport-based graph kernels. Non-optimal transport-based graph kernels was inspired by Haussler's framework for R-convolution kernels [102], which compare combinatorial real-valued feature vectors of all pairs of nodes. These kernel methods use non-linear kernel function to perform comparison between graphs in order to capture complex relational dependencies. Specifically, most of graph kernels in this category have focused on comparing graphs based on their substructures such as subtrees, cycles, shortest paths, and graphlets [107, 22, 225, 226]. They have been used in a wide range of fields such as chemoinformatics, bioinformatics, neuroscience, social networks, and computer vision [131, 253].

Despite the success of many years, R-convolution kernels often fail to leverage the useful information due to the intriguing combinatorial nature of graphs. They are primarily depend on substructure aggregation strategies. Thus, these methods have inherent limitations such as they exclude feature and structural distributions of graphs, and local clustering structure. These substructure aggregation methods are often simple and lacking the ability to consider complex characteristics of graph structured data. They also require substructures to be pre-defined based on domain-specific expertise which is not always available in practical applications. In recent years, there is considerable attention on optimal transport-based graph kernel methods to

avoid these limitations [182, 245, 243]. We will discuss the definition of graph kernel, optimal transport and graph kernel methods with optimal transport in the following.

2.2.1 Graph Kernels

Graph kernels have been extensively studied in the past years (see the survey by Kriege et al. [131]). Let $\mathcal{G}$ be a non-empty set of graphs. A kernel function $\kappa : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ is defined s.t. there exists a map $\phi : \mathcal{G} \to \mathcal{H}$ with $\kappa(G_i, G_j) = \langle \phi(G_i), \phi(G_j) \rangle_{\mathcal{H}}$, where $\mathcal{H}$ refers to a reproducing kernel Hilbert space (RKHS) [102]. Traditionally, κ must be symmetric and positive semidefinite (PSD) kernel because this enables kernel-based learning methods such as SVM to solve classification problems efficiently by convex quadratic programming [96]. However, many practical applications may produce indefinite kernels [199, 210, 183] and cannot be theoretically supported in the traditional kernel setting. For example, standard SVM learning with an indefinite kernel is a nonconvex optimization problem [96]. Therefore, several approaches have been proposed to address the issues of indefinite kernels, e.g., applying spectral transformations on indefinite kernels, reformulating a kernel learning problem into a convex optimization problem, etc. [193, 210, 184, 156, 47].

2.2.2 Non-Optimal Transport Graph Kernels

A number of non-OT graph kernel techniques have been proposed in the last two decades. Specifically, these graph kernel methods are motivated by empirical success of the classification tasks, theoretical properties, and adaptability to specific application domains. The main intuition on non-OT graph kernels is to compute embedding vectors based on characteristic properties of a graph such as subtree patterns, subgraphs, random-walks, and shortest paths. There are two main frameworks to design non-OT graph kernels: (1) R-convolution framework and (2) optimal assignment (OA) framework. These two techniques are decomposed a graph into its substructure patterns. Then, R-convolution and OA are defined a convolution operation and a bijective function on substructure patterns to measure the similarity between graphs, respectively [102, 130].

R-convolution kernels are widely studied domain compared to OA kernels. The R-convolution kernel measures the structural similarity between graph by comparing each substructure patterns such as subgraphs, graphlets, edges, and walks & paths. One of the main limitations of these R-convolution kernels is that they visit same substructure pattern multiple times, which is caused some redundancy in graph structural similarity comparison. For instance, in walk-based graph kernels [163] introduced some redundancy since there is a high probability to obtain high similarity value due to repeated visits of the same node. However, OA graph kernels can avoid this limitation since they consider a bijection between graph structures.

R-convolution kernels with subgraph patterns focus on comparing similarity between graphs by viewing a graph as bags of subgraph patterns. Shervashidze et al. [225] proposed a graph kernel by counting occurrences of graphlets (subgraph

patterns), where each graphlet is an instance of a non-isomorphic subgraph pattern. This graphlet kernel does not consider vertex attributes. Horváth et al. [107] introduced a cyclic pattern graph kernel that consider tree patterns and cycles on a graph. Da San Martino et al. [59] introduced a tree-based graph kernel method that decomposes a graph into multisets of ordered directed acyclic graphs. They extended a convolution tree kernel method to directed acyclic graphs to compute the similarity between graphs. Kriege et al. [129] proposed a subgraph matching kernel. This kernel considers the isomorphic subgraph patterns to find the similarity between two graphs.

Shervashidze et al. [226] proposed a new class of kernel method based on 1-WL algorithm for graphs with discrete attributes, which is known as WL-subtree kernel. The 1-WL test maps neighborhood feature vectors of every node into a discrete label, which is computed using a hash function. Then, WL-subtree kernel uses these labels to compare graph similarities. In this work, the authors have been proposed another two variants of this method namely, WL-edge and WL-shortest-path kernels, which follows the same procedure. Later, Morris et al. [174] proposed a graph kernel method by incorporating the higher dimensional WL algorithm such as k-WL algorithm with $k > 1$. Instead of iteratively refining the labels of vertices, this k-WL kernel method refines k-tuples of vertices to obtain the node embeddings. They also provided an efficient approximation to scale up the k-WL kernel to large graph datasets. Hido et al. [105] introduced a linear time graph kernel method, which is similar to the WL-subtree kernel. Later, Orsini et al. [186] introduced a graph invariant kernel, which can explore the high-dimensional and continuous node features. This method is first decomposed graphs into a set of subgraphs, and then compared graphs by R-convolution operation on node invariants. Later, Neumann et al. [179] introduced a propagation kernel, which is based on a randomized approach with locality-sensitive hashing technique to obtain embeddings for nodes in each iteration.

The walk & path-based graph kernels that compare sequences of edge or vertex attributes by considering traversal algorithms with shortest path or random walks. The shortest-path kernel [22], which compares the length of the shortest paths of all pairs of vertices between two graphs. This requires more computational cost for large graph datasets. Gärtner et al. [84] proposed a random walk-based graph kernels, which count the number of common walks between two graphs. Feragen et al. [75] proposed a GraphHopper kernel that is defined on attributed graphs (a graph with node and edges feature information). This graph kernel compares the shortest paths with the same length between graphs by incorporating node feature information.

However, there are several limitations on R-convolution graph kernel methods. They require substructure patterns to be pre-specified based on the domain expertise, which is not always available in practical applications. On the other hand, they simply ignore the topological structure and feature distributions.

OA-based kernel methods compare two graphs to find-out correspondences of structural elements such as cycles, triangles, and functional groups between two graphs. Fröhlich et al. [81] proposed an OA kernel for finding the similarity between attributed molecular graphs by comparing some structural elements of molecules such as rings, functional groups, etc. Specifically, this OA kernel captures an optimal

assignment by using the maximal weighted bipartite matching between embeddings of two molecules. This optimal assignment is based on the bijectivity mapping between substructures. However, these graph kernels are not PSD methods [249]. Several works have been done based on OA kernels with indefinite assignments [249, 253]. Moreover, indefinite graph kernel methods with SVM have been found to work well empirically [116]. Later, Kriege et al. [130] proposed a graph kernel method that employs a WL algorithm and uses optimal assignments of vertices. This WL-OA kernel provides a better classification accuracy on real world datasets than the WL-subtree kernel method. However, this WL-OA kernel does not perform well on continuous attributed graph structured data.

2.2.3 Optimal Transport

Typically, optimal transport compares two probability distributions by moving one distribution to the other distribution in an optimal way that minimizes a total cost of transporting probability masses [198, 251]. The distance introduced by this total transportation cost is known as the Wasserstein distance or Earth Mover's distance.

A Wasserstein distance can be defined as a distance function between two probability distributions on a metric space. Let μ and ν be two probability distributions on a metric space $\mathcal{M}$, which is equipped with a ground distance c (i.e., Euclidean distance). Thus, p-Wasserstein distance ($p \in [0, \infty)$) is defined as follows:

$$W_p(\mu, \nu) = \Big(\inf_{\gamma \in \pi(\mu,\nu)} \int_{\mathcal{M}\times\mathcal{M}} c(x,y)^p \, d\gamma(x,y) \Big)^{\frac{1}{p}}, \tag{2.14}$$

where $\pi(\mu, \nu)$ is a set of all probabilistic couplings and γ is a probabilistic coupling matrix over metric space $\mathcal{M} \times \mathcal{M}$ between marginals μ and ν.

This optimal transport problem was first introduced by Monge [170], and then after Kantorovich et al. [117] relaxed the deterministic nature of the original optimal transport problem to a more tractable one. In recent years, optimal transport has been widely revisited across machine learning applications such as image generative models [211], image segmentation [204], object matching & recognition [168, 198], and graph kernels [245].

Throughout this thesis, we consider 1-Wasserstein distance, where $p = 1$. In viewing graphs as discrete distributions in a geometric metric space $\mathcal{M}$, optimal transport techniques can be used to explore the geometric nature of graphs. Therefore, we can reformulate the continuous Wasserstein distance in Eq. 2.14 as a discrete Wasserstein distance.

$$W_1(\mu, \nu) = \min_{\gamma \in \pi(\mu,\nu)} \big\langle \gamma, \mathbf{C} \big\rangle_F, \tag{2.15}$$

where $\mathbf{C}$ is a cost function matrix which measures the cost of moving a probability mass from μ to ν (cost matrix C computes the distance $c(x, y)$ between each element x in μ and y in ν), $\big\langle ., . \big\rangle_F$ denotes the Frobenius dot product

A *feature embedding* function $\xi_f : V \to \mathbb{R}^m$ associates each vertex with a feature representation in a metric space $(\mathbb{R}^m, d_f)$. A *structure embedding* function $\xi_s : V \to \mathbb{R}^k$

associates each vertex with a structural representation in a metric space $(\mathbb{R}^k, d_s)$. We can define the notion of discrete probability distribution for graphs [243]. Let $\Sigma_n := \{\mu \in \mathbb{R}_+^n : \sum_i^n \mu_i = 1\}$ be a histogram which encodes the weight μ_i of each vertex $v_i \in V$ according to some prior information, e.g. uncertainty or relative importance. We set $\mu = (1/n)\mathbf{1}_n$ (i.e., uniform distribution) if no prior information is available, where $\mathbf{1}_n$ is a n-dimensional vector of ones. Then, a graph G can be represented as a discrete probability distribution in the product space of $(\mathbb{R}^m, d_f)$ and $(\mathbb{R}^k, d_s)$, where δ refers to a Dirac function that corresponds to the feature and structure embeddings of vertices:

$$p = \sum_{i=1}^{n} \mu_i \delta(\xi_f(v_i), \xi_s(v_i)). \tag{2.16}$$

Given two graphs G_1 and G_2 with n_1 and n_2 vertices, respectively, we denote their discrete probability distributions as $\mu \in \Sigma_{n_1}$ and $\nu \in \Sigma_{n_2}$. The set of probabilistic couplings between G_1 and G_2 is defined as:

$$\pi(\mu, \nu) = \Big\{\gamma \in \mathbb{R}_+^{n_1 \times n_2} \mid \gamma \mathbf{1}_{n_2} = \mu, \gamma^T \mathbf{1}_{n_1} = \nu\Big\}.$$

In this thesis, we aim to formalize a regularized optimal transport problem for graph kernel learning by finding an optimal coupling $\hat{\gamma}$ between two graphs:

$$\hat{\gamma} = \underset{\gamma \in \pi(\mu,\nu)}{\operatorname{argmin}} \big\langle \gamma, \mathbf{C} \big\rangle_F + \lambda \Theta(\gamma), \tag{2.17}$$

where $\lambda \in [0, 1]$ and $\Theta(\gamma)$ is a regularizer on γ. Then, given a set of graphs $\mathcal{G}$, we define a graph kernel: $\mathcal{G} \times \mathcal{G} \to \mathbb{R}$ where the kernel value for each pair of graphs in $\mathcal{G}$ is defined upon their optimal transport distance.

There are several benefits that we can obtain from regularized optimal transport instead of using classical optimal transport problems. For instance, we can smooth the Wasserstein distance estimation, encode the prior knowledge of data, and also build numerically a stable and robust optimization procedure that can guarantee convergence [198].

2.2.4 Optimal Transport Graph Kernels

Optimal transport has recently received revived interests from the machine learning community, due to its elegant way to measure the distance between two probability spaces. Following [168], Peyré et al. [197] introduced a Gromov-Wasserstein distance to compare pairwise similarity matrices from different metric spaces. Later, several studies have devoted to distance metrics for graphs. Titouan et al. [243] proposed a fused Gromov-Wasserstein distance to combine Wasserstein and Gromov-Wasserstein distances in order to jointly leverage feature and structural information of graphs. To capture global graph structure, Maretic et al. [164] proposed a Wasserstein distance between graph signal distributions by resorting to graph Laplacian matrices. This method was initially constrained to graphs of the same sizes, but recently extended to graphs of different sizes by formulating graph matching as a one-to-many assign-

ment problem [165]. Xu et al. [273] proposed to jointly align graphs and learn node embeddings using a Gromov-Wasserstein distance. To reduce computational complexity, Gromov-Wasserstein distances are often computed using a Sinkhorn algorithm [58, 232]. Recently, a scalable method was proposed in [272] to recursively partition and align large-scale graphs based on a Gromov-Wasserstein distance.

In recent years, various learning-based graph kernels have been proposed [277, 131]. Among them, several studies have attempted to cast the problem of measuring graph similarity as an instance of computing optimal transport distances for graphs in a kernel-based framework. Nikolentzos et al. [182] introduced a Wasserstein distance metric to compare graphs based on their node embeddings. Later, Togninalli et al. [245] proposed a method of computing a Wasserstein distance between the node feature distributions of two graphs in the Weisfeiler-Lehman framework [262]. Titouan et al. [243] combined Wasserstein and Gromov-Wasserstein distances in order to jointly leverage feature and structural information of graphs. These recent advances have achieved state-of-the-art results for graph classification tasks.

However, since optimal transport relies on cost functions to compare graphs but there is no ordering on vertices of a graph, a key challenge is, how to effectively define cost functions that can preserve intrinsic properties of graphs during the transport. Further, real-world graphs are often irregular and exhibit different geometric characteristics. This raises the challenge on how to develop a solid theoretical basis to ensure convergence and numerical stability for optimal transport learning on graphs.

Part I

Graph Representation Learning

Chapter 3

Structured Feature Aggregation for Spatial Graph Neural Networks

3.1 Overview

Despite advances of GNNs in various graph learning tasks such as node classification [122, 274], graph classification [275, 267] and link prediction [285], there is still a lack of theoretical understanding of how to design powerful and practically useful GNNs that can capture rich structural information of graphs. Recent studies [275, 175] have explored the connections between GNNs and the Weisfeiler-Lehman (1-WL) test [262]. By representing a neighborhood as a multiset of feature vectors and treating the neighborhood aggregation as an aggregation function over multisets, Xu et al. [275] showed that message-passing GNNs are at most as powerful as the 1-WL test in distinguishing graph structures. However, many simple graph structures still cannot be distinguished by the 1-WL test, e.g., G_1 and G_2 shown in Figure 3.1. A question is how to design expressive yet simple GNNs that can go beyond the WL test with a theoretically provable guarantee?

In this chapter, we study the problem of improving the expressive power of GNNs that can go beyond the 1-WL. This work is grounded in three observations: (i) Treating a neighborhood as a multiset of feature vectors ignores the rich structural information among vertices in the neighborhood, thereby limiting the representational capacity of the model. Thus, we represent a neighborhood as a neighborhood subgraph in which vertices are structurally related, and show that the 1-WL test is only as powerful as distinguishing neighborhood subgraphs in terms of their subtree structures in the neighborhood. (ii) There exists a natural class of isomorphic graphs, which strictly lies in between *subgraph isomorphism* and *subtree isomorphism*. We call it *overlap (subgraph) isomorphism*. The notion of overlap subgraph characterizes structural interactions of vertices and incorporate structural information into a GNNs message-passing aggregation scheme. (iii) By designing a proper function for quantifying structural interactions of vertices and preserving the injectiveness of a message-passing aggregation scheme, more expressive GNNs can be developed. We propose a new GNN model that is strictly more expressive than the 1-WL test to demonstrate an instance of this kind.

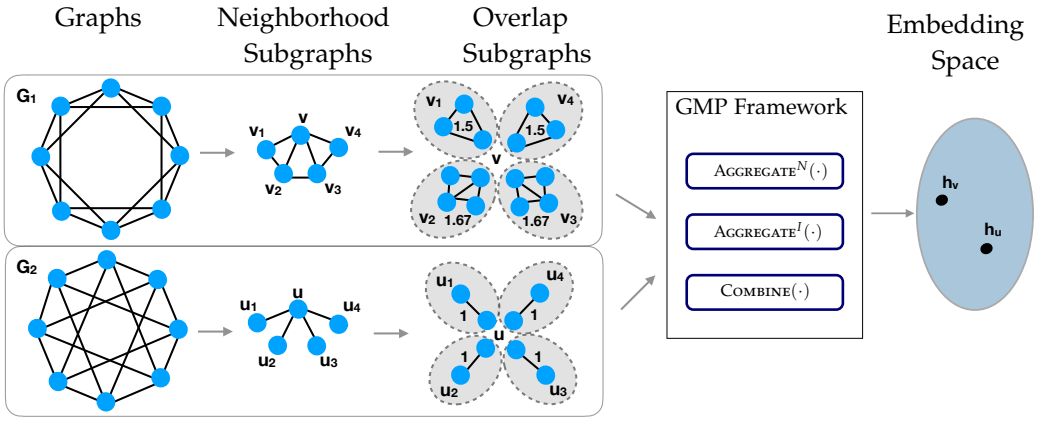


Figure 3.1: An overview of our proposed framework for GNNs that can go beyond the WL test in distinguishing non-isomorphic graphs G_1 and G_2. The overlap subgraphs of G_1 and G_2 are structurally different, which are captured by structural coefficients defined in Eq. 3.4.

The main contributions of this chapter are as follows:

- We introduce a new hierarchy of local isomorphism to characterize different classes of local structures in neighborhood subgraphs, and discuss its connections with the 1-WL test and GNNs.

- We develop a simple yet powerful framework to inject structural properties into a message-passing aggregation scheme, and theoretically characterize how GNNs can be designed to be more expressive beyond the 1-WL test.

- We propose a novel neural model for graph learning, called GraphSNN, and prove that GraphSNN is strictly more expressive than the the 1-WL test in distinguishing graph structures.

- We show that, due to the way of injecting structural properties into a structured-message-passing aggregation scheme, GraphSNN can overcome the oversmoothing issue [42, 292, 141].

- We have conducted experiments on benchmark tasks [108]. The experimental results show that our model is highly efficient and can significantly improve the state-of-the-art methods without sacrificing computational simplicity.

The rest of this chapter is organised as follows. In Section 3.2, we present a new hierarchy of local isomorphism, a GNN model beyond 1-WL and a generalized message passing GNN. In Section 3.3, we analyze the expressive power and complexity of the proposed GNN model. In Section 3.4, we discuss the experimental setup. In Section 3.5, we compare the performance of our proposed GNN method against the baseline methods. Section 3.6 summarises the chapter.

3.2 Graph Neural Networks Go Beyond Weisfeiler-Lehman

In this section, we characterize a new hierarchy of graph isomorphism based on local neighborhood subgraphs and explore its connections to 1-WL. We also present a novel generalized message-passing framework which enables to inject local structure into an aggregation scheme, in light of overlap subgraphs. We theoretically characterize how GNNs can be designed to be more expressive than 1-WL in this framework.

3.2.1 A New Hierarchy of Local Isomorphism

Let $G = (V, E, \mathbf{X})$ be a simple, undirected graph with a set V of vertices, a set E of edges and $\mathbf{X} \in \mathbb{R}^{|V| \times f}$ be a matrix of input feature vectors where $x_v \in \mathbb{R}^f$ is an input feature vector associated with each $v \in V$. The set of neighbors of a vertex v is denoted by $\mathcal{N}(v) = \{u \in V | (v, u) \in E\}$. The *neighborhood subgraph* of a vertex v, denoted by S_v, is the subgraph induced in G by $\tilde{\mathcal{N}}(v) = \mathcal{N}(v) \cup \{v\}$, which contains all edges in E that have both endpoints in $\tilde{\mathcal{N}}(v)$. For two adjacent vertices v and u, i.e., $(v, u) \in E$, the *overlap subgraph* S_{vu} between v and u is defined as $S_{vu} = S_v \cap S_u$.

Let S_i and S_j be the neighborhood subgraphs of two vertices i and j that are not necessarily adjacent, and h_v be the feature vector of a vertex $v \in V$. In the following, we define three notions of isomorphism, which correspond to different classes of local structures in neighborhood subgraphs.

Definition 1. *S_i and S_j are* subgraph-isomorphic, *denoted as $S_i \simeq_{subgraph} S_j$, if there exists a bijective mapping $g : \tilde{\mathcal{N}}(i) \to \tilde{\mathcal{N}}(j)$ such that $g(i) = j$ and for any two vertices $v_1, v_2 \in \tilde{\mathcal{N}}(i)$, v_1 and v_2 are adjacent in S_i iff $g(v_1)$ and $g(v_2)$ are adjacent in S_j, and $h_{v_1} = h_{g(v_1)}$ and $h_{v_2} = h_{g(v_2)}$.*

Definition 2. *S_i and S_j are* overlap-isomorphic, *denoted as $S_i \simeq_{overlap} S_j$, if there exists a bijective mapping $g : \tilde{\mathcal{N}}(i) \to \tilde{\mathcal{N}}(j)$ such that $g(i) = j$ and for any $v' \in \mathcal{N}(i)$ and $g(v') = u'$, $S_{iv'}$ and $S_{ju'}$ are subgraph-isomorphic.*

Definition 3. *S_i and S_j are* subtree-isomorphic, *denoted as $S_i \simeq_{subtree} S_j$, if there exists a bijective mapping $g : \tilde{\mathcal{N}}(i) \to \tilde{\mathcal{N}}(j)$ such that $g(i) = j$ and for any $v' \in \tilde{\mathcal{N}}(i)$ and $g(v') = u'$, $h_{v'} = h_{u'}$.*

The following theorem 1 states that there is a hierarchy among the notions of local isomorphism on neighborhood subgraphs, where subgraph-isomorphism is the strongest one, subtree-isomorphism is the weakest, and overlap-isomorphism lies in between. Figure 3.2 shows two groups of graphs: one is distinguishable w.r.t. subgraph-isomorphism but not overlap-isomorphism, while the other is distinguishable by overlap-isomorphism but not subtree-isomorphism.

Theorem 1. *The following statements are true: (a) If $S_i \simeq_{subgraph} S_j$, then $S_i \simeq_{overlap} S_j$; but not vice versa; (b) If $S_i \simeq_{overlap} S_j$, then $S_i \simeq_{subtree} S_j$; but not vice versa.*

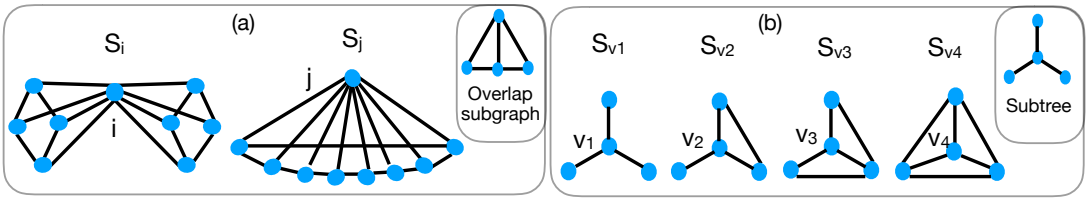


Figure 3.2: (a) S_i and S_j are overlap-isomorphic (i.e., having the same overlap subgraph) but not subgraph-isomorphic; (b) Four neighborhood subgraphs $\{S_{v_i}|i = 1,2,3,4\}$ are subtree-isomorphic (i.e., having the same subtree) but not overlap-isomorphic.

Proof. In the following, we prove the statements in this theorem one by one.

For Statement (a), by $S_i \simeq_{subgraph} S_j$ and Definition 1, we know that there exists a bijective mapping $g' : \tilde{\mathcal{N}}(i) \to \tilde{\mathcal{N}}(j)$ such that for the vertex i and any vertex $v' \in \mathcal{N}(i)$, i and v' are adjacent in S_i iff $j = g(i)$ and $u' = g(v')$ are adjacent in S_j, and $h_i = h_j$ and $h_{v'} = h_{u'}$, where g is a bijective mapping between S_i and S_j as defined by Definition 1. Then for each pair of overlap subgraphs $S_{iv'}$ and $S_{ju'}$, we can further extend g' along g on $S_{iv'}$ and $S_{ju'}$. That is, $g'(v) = u$ iff $g(v) = u$. If v in $S_{iv'}$, by the definition of overlap subgraph, v must either be i or a neighbor of i. Hence $u = g'(v)$ in this case must be either j or a neighbor of j. By the definition of g and the fact that $g'(v) = u$ iff $g(v) = u$, we know that for any two vertices v_1 and v_2 in $S_{iv'}$, they are adjacent in $S_{iv'}$ iff their corresponding vertices $g'(v_1)$ and $g'(v_2)$ are adjacent in $S_{ju'}$ and their corresponding feature vectors are indistinguishable, i.e, $S_{iv'} \simeq_{subgraph} S_{ju'}$ for any $v' \in \mathcal{N}(i)$ and $g(v') = u'$. Conversely, if $S_i \simeq_{overlap} S_j$, then it is possible that $S_i \not\simeq_{subgraph} S_j$ as shown by the two graphs in Figure 3.2(a).

For Statement (b), if $S_i \simeq_{overlap} S_j$, then to prove $S_i \simeq_{subtree} S_j$ we need to show that there exists a bijective mapping $g : \tilde{\mathcal{N}}(i) \to \tilde{\mathcal{N}}(j)$ such that $g(i) = j$ and, for any $v' \in \tilde{\mathcal{N}}(i)$ and $g(v') = u'$, the feature vectors of v' and u' are indistinguishable, i.e., $h_{v'} = h_{u'}$. By Def. 2, we can find a bijective mapping $g' : \tilde{\mathcal{N}}(i) \to \tilde{\mathcal{N}}(j)$ such that $g'(i) = j$ and, for any $v' \in \mathcal{N}(i)$ and $g'(v') = u'$, $S_{iv'}$ and $S_{ju'}$ are subgraph-isomorphic. This implies that g' cannot distinguish the feature vectors of v' and u' for any $v' \in \tilde{\mathcal{N}}(i)$ and $g(v') = u'$. Similarly, the converse does not necessarily hold and one counterexample is the set of graphs as shown in Figure 3.2(b) which are subtree-isomorphic but not overlap-isomorphic. □

Let $\mathcal{S} = \{S_v|v \in V\}$ be the set of neighborhood subgraphs in G and $\zeta : \mathcal{S} \to \mathbb{R}^d$ map each neighborhood subgraph in $\mathcal{S}$ into a node embedding in $\mathbb{R}^d$. The following theorem states that GNNs that are as powerful as 1-WL can distinguish two neighborhood subgraphs only w.r.t. subtree-isomorphism at each layer.

Theorem 2. *Let M be a GNN. M is as powerful as 1-WL in distinguishing non-isomorphic graphs if M has a sufficient number of layers and each layer can map any S_i and S_j in $\mathcal{S}$ into two different embeddings (i.e., $\zeta(S_i) \neq \zeta(S_j)$) if and only if $S_i \not\simeq_{subtree} S_j$.*

Proof. We first show that, for any two graphs G_1 and G_2, if they can be distinguished by 1-WL, then they must be distinguishable by such a GNN M as well. Suppose that

1-WL takes k iterations to distinguish G_1 and G_2, i.e., 1-WL yields the same multiset of node labels on G_1 and G_2 in the iterations from 0 to k-1, but two different multisets of node labels on G_1 and G_2 in the k-th iteration. To derive a contradiction, we assume that a GNN M that satisfies the above two conditions cannot distinguish G_1 and G_2 in the iterations from 0 to k. Since 1-WL can distinguish G_1 and G_2 in the k-th iteration, it means that there must exist two neighborhood subgraphs, say S_i and S_j, which correspond to two different multisets of node labels on G_1 and G_2 at the k-th iteration. These two different multisets of node labels correspond to two different multisets of feature vectors in their neighborhoods, i.e., $\{\!\{h_v|v \in \mathcal{N}(i)\}\!\} \neq \{\!\{h_u|u \in \mathcal{N}(j)\}\!\}$. By Def. 3, we know that $S_i \not\simeq_{subtree} S_j$. Then this means that $\zeta(S_i) \neq \zeta(S_j)$, which contradicts the assumption that M cannot distinguish G_1 and G_2 in the iteration k.

Now, we show the other direction that, for any two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, if they can be distinguished by such a GNN M, then they must be distinguishable by 1-WL. Similarly, suppose that at the k-th iteration, M maps the neighborhood subgraphs of these two graphs into two different multisets of node embeddings, i.e., $\{\!\{\zeta(S_v)|v \in V_1\}\!\} \neq \{\!\{\zeta(S_u)|v \in V_2\}\!\}$. This is means that we can find at least two different neighborhood subgraphs S_i and S_j such that $\zeta(S_i) \neq \zeta(S_j)$. For such neighborhood subgraphs S_i and S_j, we know that $S_i \not\simeq_{subtree} S_j$. Then this means that S_i and S_j correspond to either $h_i \neq h_j$ or $\{\!\{h_v|v \in \mathcal{N}(i)\}\!\} \neq \{\!\{h_u|u \in \mathcal{N}(j)\}\!\}$, which can be relabeled by 1-WL into two different new labels. Thus, 1-WL can also distinguish such neighborhood subgraphs, and accordingly distinguish G_1 and G_2.

□

3.2.2 A Generalized Message-Passing Framework

In this section, we present a generalized message-passing framework (GMP). The GMP injects the local structure into a message-passing scheme. Let $\mathcal{S}^* = \{S_{vu}|(v,u) \in E\}$ be the set of overlap subgraphs in G. We define *structural coefficients* for each vertex v and its neighbors, i.e., $\omega : \mathcal{S} \times \mathcal{S}^* \to \mathbb{R}$ such that $A_{vu} = \omega(S_v, S_{vu})$. A question arising is: what are the desirable properties of such a function ω? Ideally, it should quantify how a vertex v structurally interacts with its neighbor u in the local neighborhood. Let $u, u' \in \mathcal{N}(v)$, $S_{vu} = (V_{vu}, E_{vu})$ and $S_{vu'} = (V_{vu'}, E_{vu'})$, a carefully designed ω should exhibit the following properties:

(1) **Local closeness:** $\omega(S_v, S_{vu}) > \omega(S_v, S_{vu'})$ if S_{vu} and $S_{vu'}$ are complete graphs with $S_{vu} = K_i$, $S_{vu'} = K_j$, and $i > j$, where K_i refers to a complete graph on i vertices.

(2) **Local denseness:** $\omega(S_v, S_{vu}) > \omega(S_v, S_{vu'})$ if S_{vu} and $S_{vu'}$ have the same number of vertices but differ in the number of edges s.t. $|V_{vu}| = |V_{vu'}|$ and $|E_{vu}| > |E_{vu'}|$.

(3) **Isomorphic invariant:** $\omega(S_v, S_{vu}) = \omega(S_v, S_{vu'})$ if S_{vu} and $S_{vu'}$ are isomorphic.

Figure 3.3 illustrates the first two properties. Let $\{\!\{\cdot\}\!\}$ denote a multiset, $\tilde{A} = (\tilde{A}_{vu})_{v,u \in V}$ where $\tilde{A}_{vu}$ is a normalised value of A_{vu}. We denote the feature vector of v at the t-th layer by $h_v^{(t)}$ and $h_v^{(0)} = x_v$. Then, the (t+1)-th layer of an aggregation

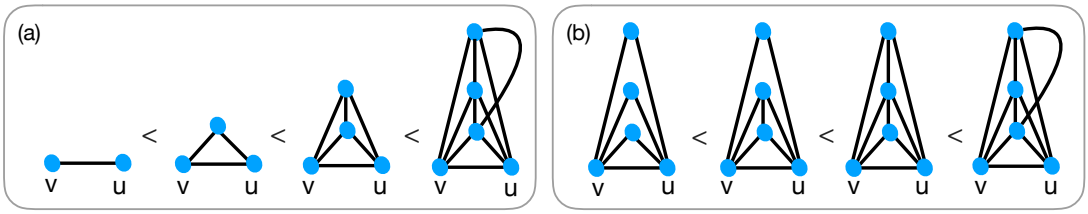


Figure 3.3: (a) *Local closeness*: for overlap subgraphs that are complete graphs, their structural coefficients increase with the number of vertices; (b) *Local denseness*: for overlap subgraphs that have the same number of vertices, their structural coefficients increase with the number of edges.

scheme can be defined as:

$$m_a^{(t)} = \textsc{Aggregate}^N\Big(\{\!\{(\tilde{A}_{vu}, h_u^{(t)}) | u \in \mathcal{N}(v)\}\!\}\Big), \tag{3.1}$$

$$m_v^{(t)} = \textsc{Aggregate}^I\Big(\{\!\{\tilde{A}_{vu} | u \in \mathcal{N}(v)\}\!\}\Big) h_v^{(t)}, \tag{3.2}$$

$$h_v^{(t+1)} = \textsc{Combine}\Big(m_v^{(t)}, m_a^{(t)}\Big). \tag{3.3}$$

$\textsc{Aggregate}^N(\cdot)$ and $\textsc{Aggregate}^I(\cdot)$ are two possibly different parameterized functions. Here, $m_a^{(t)}$ is a message aggregated from the neighbors of v and their structural coefficients, and $m_v^{(t)}$ is an "adjusted" message from v after performing an element-wise multiplication between $\textsc{Aggregate}^I(\cdot)$ and $h_v^{(t)}$ to account for structural effects from its neighbors. Then, $m_v^{(t)}$ and $m_a^{(t)}$ are combined by $\textsc{Combine}(\cdot)$ to obtain the feature vector $h_v^{(t+1)}$.

3.2.3 A Graph Neural Network Model Beyond 1-WL

Let M be a GNN whose aggregation scheme Φ is defined by Eq. 3.1-Eq. 3.3. Generally, there are many different ways of designing ω and Φ functions within the GMP framework, leading to GNNs with different expressive powers. To elaborate this, we propose a novel GNN model, named GraphSNN, whose aggregation scheme is an instantiation of our GMP framework. We prove that the expressive power of GraphSNN goes beyond 1-WL.

3.2.3.1 Model design

In the following, we provide a definition of ω that satisfies the properties of local closeness, local denseness, and isomorphic invariant. One key idea behind this definition is to make it capable of being generalized to support different graph learning tasks, controlled by $\lambda > 0$ (will be further discussed in Section 3.5.6):

$$\omega(S_v, S_{vu}) = \frac{|E_{vu}|}{|V_{vu}| \cdot |V_{vu} - 1|} |V_{vu}|^{\lambda}. \tag{3.4}$$

This definition allows us to formulate a weighted adjacency matrix $A = (A_{vu})_{v,u \in V}$ for GraphSNN. To compare structural coefficients across different neighboring nodes, we normalize A to $\tilde{A}$ by $\tilde{A}_{vu} = \frac{A_{vu}}{\sum_{u \in \mathcal{N}(v)} A_{vu}}$. Alternatively, A can be normalized using Softmax or other normalization techniques. For each vertex $v \in V$, the feature vector at the (t+1)-th layer is generated by

$$h_v^{(t+1)} = \textsc{Mlp}_\theta\Big(\gamma^{(t)}\Big(\sum_{u \in \mathcal{N}(v)} \tilde{A}_{vu} + 1\Big)h_v^{(t)} + \sum_{u \in \mathcal{N}(v)} \Big(\tilde{A}_{vu} + 1\Big)h_u^{(t)}\Big), \tag{3.5}$$

where $\gamma^{(t)}$ is a learnable scalar parameter. Since $\mathcal{N}(v)$ refers to one-hop neighbors of v, one can stack multiple layers to handle more than one-hop neighborhood. Note that, to ensure the injectivity in the feature aggregation in the presence of structural coefficients, we add 1 into the first and second terms in Eq. 3.5. This design is critical for guaranteeing the expressiveness of GraphSNN beyond 1-WL, as will be discussed in the proofs of the lemmas and Theorem 4 in Section 3.3.

3.2.3.2 Connections to Previous Work

In the following, we discuss how our framework generalizes the existing message-passing GNNs in the literature such as GCN [122], GraphSAGE [99], GAT [247] and GIN [275] as special cases. Table 3.1 presents the local aggregation schemes used by these existing GNN models. They differ from each other w.r.t. the way of aggregating feature vectors in a neighborhood and how they are combined with the current vertex's feature itself, i.e., summation or concatenation. Here, α_{vu} is an attention coefficient capturing the importance of a neighbor in GAT, ϵ is a learnable or fixed scalar parameter used in GIN, W is a learnable weight matrix, and σ is a non-linear activation function, such as ReLU.

Note that, as defined in Eq. 3.3, $m_a^{(t)}$ and $m_v^{(t)}$ refer to the messages aggregated by $\textsc{Aggregate}^N(\cdot)$ and $\textsc{Aggregate}^I(\cdot)$, respectively.

GNN Model	$\textsc{Aggregate}^N(\cdot)$	$\textsc{Aggregate}^I(\cdot)$	$\textsc{Combine}(\cdot)$
GCN	$\sum_{u \in \mathcal{N}(v)} \frac{W^{(t)} h_u^{(t)}}{\sqrt{\lvert\mathcal{N}(u)\rvert\lvert\mathcal{N}(v)\rvert}}$	$\frac{W^{(t)} h_v^{(t)}}{\sqrt{\lvert\mathcal{N}(v)\rvert\lvert\mathcal{N}(v)\rvert}}$	$\sigma(\textsc{Sum}(m_v^{(t)}, m_a^{(t)}))$
GraphSAGE	$\sum_{u \in \mathcal{N}(v)} \frac{h_u^{(t)}}{\lvert\mathcal{N}(v)\rvert}$	$h_v^{(t)}$	$\sigma(W^{(t)} \cdot \textsc{Concat}(m_v^{(t)}, m_a^{(t)}))$
GAT	$\sum_{u \in \mathcal{N}(v)} \alpha_{vu} W^{(t)} h_u^{(t)}$	$\alpha_{vv} W^{(t)} h_v^{(t)}$	$\sigma(\textsc{Sum}(m_v^{(t)}, m_a^{(t)}))$
GIN	$\sum_{u \in \mathcal{N}(v)} h_u^{(t)}$	$(1+\epsilon) h_v^{(t)}$	$\textsc{Mlp}_\theta(\textsc{Sum}(m_v^{(t)}, m_a^{(t)}))$

Table 3.1: Comparison of the aggregation schemes used in existing message-passing GNNs

3.2.3.3 Formulation of GraphGNN$_M$

The purpose of GraphGNN$_M$ formulation is to evaluate how effectively our aggregation scheme with structural coefficients can learn representations for vertices, compared with the corresponding standard message-passing aggregation scheme.

For each of these structural message-passing GNNs, denoted as M, we construct a variant GraphSNN$_M$ by replacing its standard aggregation scheme with our aggregation scheme as formulated in Eq. 3.5. These variants are used in our experiments for node classification benchmark tasks (see Section 3.5.1) in order to evaluate how our aggregation scheme with structural coefficients can improve performance, compared with their standard message-passing aggregation schemes. Below are the details of these variants.

GCN and GraphSNN$_{GCN}$: Graph Convolutional Network (GCN) [122] applies a normalized mean aggregation to combine the feature vector of a node v with the feature vectors in its neighborhood $\mathcal{N}(v)$:

$$h_v^{(t+1)} = \sigma\Big(\frac{W^{(t)}h_v^{(t)}}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(v)|}} + \sum_{u\in\{\mathcal{N}(v)\}} \frac{W^{(t)}h_u^{(t)}}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}}\Big). \tag{3.6}$$

$\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}$ is a normalization constant for the edge (v,u), which originates from the normalized adjacency matrix $D^{-1/2}AD^{-1/2}$. $W^{(t)}$ is a trainable weight matrix and σ is a non-linear activation function such as ReLU. We generalize GCN to a model under the GMP framework, namely **GraphSNN$_{GCN}$**, to improve the expressive power of GCN. We first construct a normalized structural coefficient matrix $\tilde{A}$. Then each neural layer of GraphSNN$_{GCN}$ is expressed as:

$$h_v^{(t+1)} = \sigma\left(\gamma^{(t)}\Big(\sum_{u\in\mathcal{N}(v)}\tilde{A}_{vu}+1\Big)\frac{W^{(t)}h_v^{(t)}}{\sqrt{|\tilde{\mathcal{N}}(v)||\tilde{\mathcal{N}}(v)|}} + \sum_{u\in\mathcal{N}(v)}\Big(\tilde{A}_{vu}+1\Big)\frac{W^{(t)}h_u^{(t)}}{\sqrt{|\tilde{\mathcal{N}}(u)||\tilde{\mathcal{N}}(v)|}}\right). \tag{3.7}$$

GraphSAGE and GraphSNN$_{GraphSAGE}$: GraphSAGE [99] learns aggregation functions to induce new node feature vectors by sampling and aggregating features from a node's local neighborhood. GraphSAGE has considered three different aggregation functions: mean aggregator, LSTM aggregator, and pooling aggregator. In our work, we mainly focus on the mean aggregator that, for each vertex v, takes the mean of the feature vectors of the nodes in its neighborhood and concatenates it with the feature vector of v as shown below:

$$h_v^{(t+1)} = \sigma\Big(W^{(t)}\cdot \textsc{Concat}\Big(\frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} h_u^{(t)}, h_v^{(t)}\Big)\Big), \tag{3.8}$$

where $W^{(t)}$ is a learnable weight matrix, . refers to a matrix multiplication, and σ represents a non-linear activation function. We also generalize GraphSNN to a model under the GMP framework, namely **GraphSNN$_{GraphSAGE}$**. This model first

takes a mean aggregation of the feature vectors in the neighborhood $\mathcal{N}(v)$ and then concatenates it with the feature vector of v itself in the following manner:

$$h_v^{(t+1)} = \sigma\Big(W^{(t)} \cdot \Big(\frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} \Big(\tilde{A}_{vu} + 1\Big) h_u^{(t)} \oplus \gamma^{(t)}\Big(\sum_{u \in \mathcal{N}(v)} \tilde{A}_{vu} + 1\Big) h_v^{(t)}\Big)\Big). \quad (3.9)$$

GAT and GraphSNN$_{GAT}$: Graph Attention Network (GAT) [247] linearly transforms the input feature vectors and performs a weighted sum of the feature vectors for vertices in a neighborhood after the transformation. GAT computes attention weights $\alpha_{vu}^{(t)}$ using an attention mechanism and aggregates the feature vectors in a neighborhood as follows:

$$h_v^{(t+1)} = \sigma\Big(\sum_{(v,u) \in E} \alpha_{vu}^{(t)} W^{(t)} h_u^{(t)}\Big), \quad (3.10)$$

where $W^{(t)}$ is a trainable weight matrix and σ represents a non-linear activation function. We generalize GAT to a model, called **GraphSNN$_{GAT}$**, in the GMP framework. Firstly, we aggregate the feature vectors based on structural coefficients in our aggregation scheme, i.e., we compute

$$\tilde{h}_u^{(t)} = \gamma^{(t)}\Big(\sum_{z \in \mathcal{N}(u)} \tilde{A}_{uz} + 1\Big) \frac{h_u^{(t)}}{\sqrt{|\tilde{\mathcal{N}}(u)||\tilde{\mathcal{N}}(u)|}} + \sum_{z \in \mathcal{N}(u)} \Big(\tilde{A}_{uz} + 1\Big) \frac{h_z^{(t)}}{\sqrt{|\tilde{\mathcal{N}}(z)||\tilde{\mathcal{N}}(u)|}} \quad (3.11)$$

and

$$\tilde{h}_v^{(t)} = \gamma^{(t)}\Big(\sum_{z' \in \mathcal{N}(v)} \tilde{A}_{vz'} + 1\Big) \frac{h_v^{(t)}}{\sqrt{|\tilde{\mathcal{N}}(v)||\tilde{\mathcal{N}}(v)|}} + \sum_{z' \in \mathcal{N}(v)} \Big(\tilde{A}_{vz'} + 1\Big) \frac{h_{z'}^{(t)}}{\sqrt{|\tilde{\mathcal{N}}(z')||\tilde{\mathcal{N}}(v)|}}. \quad (3.12)$$

We then construct attention coefficients $\alpha_{vu}^{(t)}$ on these aggregated feature vectors as follows:

$$\alpha_{vu}^{(t)} = \frac{\exp\Big(\text{LeakyReLU}\big(a^T [W^{(t)} \tilde{h}_v^{(t)} \oplus W^{(t)} \tilde{h}_u^{(t)}]\big)\Big)}{\sum_{z \in \mathcal{N}(v)} \exp\Big(\text{LeakyReLU}\big(a^T [W^{(t)} \tilde{h}_v^{(t)} \oplus W^{(t)} \tilde{h}_z^{(t)}]\big)\Big)}, \quad (3.13)$$

where $\oplus$ represents the concatenation, $W^{(t)}$ is a learnabe weight matrix and a is a learnable weight vector. After that, we aggregate the neighborhood features as follows using attention coefficients.

$$h_v^{(t+1)} = \sigma\Big(\sum_{(v,u) \in E} \alpha_{vu}^{(t)} W^{(t)} \tilde{h}_u^{(t)}\Big), \quad (3.14)$$

where $W^{(t)}$ is a learnable weight matrix, and σ represents a non-linear activation function. We use multi-head attention as stated in the original work [247].

GIN and GraphSNN$_{GIN}$: Graph Isomorphism Network (GIN) [275] takes the

sum aggregation over neighbors in a neighborhood, followed by a 2-layer MLP. $\epsilon^{(t+1)}$ is a learnable parameter or fixed scalar. Each neural layer is expressed as:

$$h_v^{(t+1)} = \textsc{Mlp}^{(t+1)}\Big((1+\epsilon^{(t+1)})h_v^{(t)} + \sum_{u\in\mathcal{N}(v)} h_u^{(t)}\Big). \tag{3.15}$$

Here, we consider one of GIN variants employed in the original paper, where the learnable parameter $\epsilon = 0$, and generalize it to **GraphSNN**$_{GIN}$ as defined bwlow:

$$h_v^{(t+1)} = \textsc{Mlp}^{(t+1)}\Big(\gamma^{(t)}\Big(\sum_{u\in\mathcal{N}(v)} \tilde{A}_{vu} + 1\Big)h_v^{(t)} + \sum_{u\in\mathcal{N}(v)} \Big(\tilde{A}_{vu} + 1\Big)h_u^{(t)}\Big). \tag{3.16}$$

3.3 Theoretical Analysis

In this section, we first prove that a GNN can be more expressive than 1-WL if ω is powerful enough to distinguish structures beyond neighborhood subtrees and the neighborhood aggregation function Φ is injective under a sufficient number of layers by Theorem 3. Then, we prove that GraphSNN message-passing aggregation scheme is an injective by Lemma 1 and Lemma 2. Then, we prove that GraphSNN is more expressive than 1-WL for distinguishing non-isomorphic graphs by Theorem 4.

3.3.1 Expressiveness Analysis

Theorem 3. *M is strictly more expressive than 1-WL in distinguishing non-isomorphic graphs if M has a sufficient number of layers and also satisfies the following conditions:*

(1) M can distinguish at least two neighborhood subgraphs S_i and S_j with $S_i \simeq_{subtree} S_j$, $S_i \not\simeq_{subgraph} S_j$ and $\{\!\{\tilde{A}_{iv'} | v' \in \mathcal{N}(i)\}\!\} \neq \{\!\{\tilde{A}_{ju'} | u' \in \mathcal{N}(j)\}\!\}$;

(2) $\Phi\Big(h_v^{(t)}, \{\!\{h_u^{(t)} | u \in \mathcal{N}(v)\}\!\}, \{\!\{(\tilde{A}_{vu}, h_u^{(t)}) | u \in \mathcal{N}(v)\}\!\}\Big)$ is injective.

Proof. We prove this theorem in two steps. First, we prove that a GNN M satisfying the above conditions can distinguish any two graphs that are distinguishable by 1-WL by contradiction. Assume that there exist two graphs G_1 and G_2 which can be distinguished by 1-WL but cannot be distinguished by M. Further, suppose that 1-WL cannot distinguish these two graphs in the iterations from 0 to k-1, but can distinguish them in the k-th iteration. Then, there must exist two neighborhood subgraphs S_i and S_j whose neighboring nodes correspond to two different multisets of node labels at the k-th iteration, i.e., $\{\!\{h_v^{(k)} | v \in \mathcal{N}(i)\}\!\} \neq \{\!\{h_u^{(k)} | u \in \mathcal{N}(j)\}\!\}$. By the above condition (2), we know that Φ is injective. Thus, for S_i and S_j, Φ would yield two different feature vectors at the k-th iteration. This means that M can also distinguish G_1 and G_2, which contradicts the assumption. Our proof in the first step is done. For the second step, we can prove that there exist at least two graphs that can be distinguished by M but cannot be distinguished by 1-WL. Figure 3.1 presents two of such graphs. □

We first generalize the result of universal functions over *multisets* [275] to universal functions over *pairs of multisets* since Eq. 3.5 involves not only node features but also structural coefficients. Assume that $\mathcal{H}$, $\mathcal{A}$ and $\mathcal{W}$ are countable sets where $\mathcal{H}$ is a node feature space, $\mathcal{A}$ is a structural coefficient space, and $\mathcal{W} = \{A_{ij}h_i | A_{ij} \in \mathcal{A}, h_i \in \mathcal{H}\}$. Let H and W be two multisets containing elements from $\mathcal{H}$ and $\mathcal{W}$, respectively, and $|H| = |W|$.

Lemma 1. *There exists a function f s.t. $\pi(H, W) = \sum_{h\in H, w\in W} f(h, w)$ is unique for any distinct pair of multisets (H, W).*

Proof. Since $\mathcal{H}$ and $\mathcal{W}$ are countable, there must exist two functions $\psi_1 : \mathcal{H} \to \mathbb{N}_{odd}$ mapping $h \in \mathcal{H}$ to odd natural numbers and $\psi_2 : \mathcal{W} \to \mathbb{N}_{even}$ mapping $w \in \mathcal{W}$ to even natural numbers. Further, for any pair of multisets (H, W), since the cardinality of H and W is bounded, there must exist a number $N \in \mathbb{N}$ such that $|H| < N$ and $|W| < N$. Thus, we can find a prime number $P > 2N$. Then we have a mapping f as $f(h, w) = P^{-\psi_1(h)} + P^{-\psi_2(w)}$ such that $\sum_{h\in H, w\in W} f(h, w)$ is unique for each distinct pair of (H, W). □

We can extend the injectiveness of $\pi(H, W)$ to $\pi'(h_v, H, W)$ as in the lemma below.

Lemma 2. *There exists a function f s.t. $\pi'(h_v, H, W) = \gamma f(h_v, |H|h_v) + \sum_{h\in H, w\in W} f(h, w)$ is unique for any distinct (h_v, H, W), where $h_v \in \mathcal{H}$, $|H|h_v \in \mathcal{W}$, and γ is an irrational number.*

Proof. As $h_v \in \mathcal{H}$ and $|H|h_v \in \mathcal{W}$, we may have $f(h_v, |H|h_v) = P^{-\psi_1(h_v)} + P^{-\psi_1(|H|h_v)}$ where $\psi_1 : \mathcal{H} \to \mathbb{N}_{odd}$ and $\psi_2 : \mathcal{W} \to \mathbb{N}_{even}$ as defined in the proof for Lemma 1. Let (h_{v1}, H_1, W_1) and (h_{v2}, H_2, W_2) be two different tuples. Then, there are two cases:

(1) When $h_{v1} = h_{v2}$ but $(H_1, W_1) \neq (H_2, W_2)$, by Lemma 1, we know that: $\sum_{h\in H_1, w\in W_1} f(h, w) \neq \sum_{h\in H_2, w\in W_2} f(h, w)$. Thus, $\pi'(h_{v1}, H_1, W_1) \neq \pi'(h_{v2}, H_2, W_2)$.

(2) When $h_{v1} \neq h_{v2}$, we prove $\pi'(h_{v1}, H_1, W_1) \neq \pi'(h_{v2}, H_2, W_2)$ by contradiction. Assume that $\pi'(h_{v1}, H_1, W_1) = \pi'(h_{v2}, H_2, W_2)$. Then, we have:

$$\gamma f(h_{v1}, |H_1|h_{v1}) + \sum_{h\in H_1, w\in W_1} f(h, w) = \gamma f(h_{v2}, |H_2|h_{v_2}) + \sum_{h\in H_2, w\in W_2} f(h, w).$$

This gives us the following equation:

$$\gamma\Big(f(h_{v1}, |H_1|h_{v1}) - f(h_{v2}, |H_2|h_{v2})\Big) = \Big(\sum_{h\in H_2, w\in W_2} f(h, w)\Big) - \Big(\sum_{h\in H_1, w\in W_1} f(h, w)\Big).$$

When γ is an irrational number, L.H.S. of the above equation is irrational but R.H.S. is rational. There is a contradiction. Thus, $\pi'(h_{v1}, H_1, W_1) \neq \pi'(h_{v2}, H_2, W_2)$.

□

Since any function over (h_v, H, W) can be decomposed as:

$$g(\gamma f(h_v, |H|h_v) + \sum_{h \in H, w \in W} f(h, w)),$$

similar to Xu et al. [275], we use a parameterized multi-layer perceptron (MLP) to learn f and g. The following theorem characterizes the expressive power of GraphSNN.

Theorem 4. *GraphSNN is more expressive than 1-WL in testing non-isomorphic graphs.*

Proof. We prove this theorem by showing that GraphSNN is a GNN satisfying the conditions stated in Theorem 3. There are two conditions:

(1) Consider the two graphs shown in Figure 3.1. GraphSNN can distinguish these two neighborhood subgraphs S_i and S_j with $\{\!\{\tilde{A}_{iv'} | v' \in \mathcal{N}(i)\}\!\} \neq \{\!\{\tilde{A}_{ju'} | u' \in \mathcal{N}(j)\}\!\}$.

(2) By Lemmas 1 and 2 as well as the fact that MLP as a universal approximator [275] can be used to model and learn the functions f and g, we know that GraphSNN also satisfies this condition.

□

Since GIN is as powerful as 1-WL [275], this theorem implies that GraphSNN is more expressive than GIN, i.e., GraphSNN can map at least two different neighborhood subgraphs that correspond to the same multiset of feature vectors to different representations, as shown in Figure 3.2.

3.3.2 Complexity Analysis

Table 3.2 summarizes the time and memory complexities of several popular message-passing GNNs and GraphSNN, where n and m are the numbers of vertices and edges in a graph, respectively, k refers to the number of layers, f and d are the dimensions of input and output feature vectors, respectively, a is the number of attention heads used in GAT, and s is the number of neighbors sampled for each node at each layer in GraphSAGE.

GNN Model	Time Complexity	Memory Complexity
GCN [122]	$O(kmfd)$	$O(m)$
GIN [275]	$O(kmfd)$	$O(m)$
GAT [247]	$O(k(anfd + amd))$	$O(n^2)$
GraphSAGE [99]	$O(snfd)$	$O(n)$
GraphSNN (ours)	$O(kmfd)$	$O(m)$

Table 3.2: Time and space complexities of message-passing GNNs and GraphSNN.

Similar to GCN and GIN, GraphSNN is computationally efficient. The time complexity and memory complexity are linear w.r.t. the number of edges in a

graph. Further, due to the locality of GraphSNN, the computation of aggregating feature vectors from neighborhood subgraphs at each layer can be parallelized across all vertices. Similarly, structural coefficients can be pre-computed with the time complexity $O(ml)$, where m is the number of edges and l is the maximum degree of vertices in a graph. Similarly, the computation of structural coefficients can be parallelized across all edges.

3.4 Experimental Setup

In this section, we present the datasets and baseline methods considered when evaluating our GNN model on semi-supervised node classification, small graph classification and large graph classification against the state-of-the-art baselines in order to answer the following questions. Our experiments are performed on a Linux server which has 12-core Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz, NVIDIA RTX A6000 with 96GB of main memory. We also discuss hyperparatemeter settings for each classification categories.

Q1. How well can our aggregation scheme with structural coefficients to improve representation learning for node classification tasks?

Q2. How well can GraphSNN empirically perform for small graph classification tasks and large graph classification tasks?

Q3. How effectively can our aggregation scheme alleviate the oversmoothing issue?

Q4. How efficiently can GraphSNN compute structural coefficients?

Q5. How well can the parameter λ, and augment cycle and clique counts to node features contribute to capture different classes of structure information on node classification and graph classification tasks?

3.4.1 Datasets

We use five datasets for node classification, eleven datasets for small graph classification, and five datasets for large graph classification in our experiments. The detailed description and summary about these datasets are provided in Section 3.4.1.1, Section 3.4.1.2 and Section 3.4.1.3.

3.4.1.1 Node Classification

We use five datasets: three citation network datasets Cora, Citeseer, and Pubmed [223] for semi-supervised document classification, one knowledge graph dataset NELL [36] for semi-supervised entity classification, and one OGB dataset ogbn-arxiv from [108]. Table 3.3 contains the statistics for the five datasets used in our experiments for node classification.

We also use three citation datasets: Cora, Citeseer, and Pubmed (see Table 3.3) for oversmoothing analysis on semi-supervised document classification.

Dataset	Type	#Nodes	#Edges	#Classes	#Features
Cora	Citation network	2,708	5,429	7	1,433
Citeseer	Citation network	3,327	4,732	6	3,703
Pubmed	Citation network	19,717	44,338	3	500
NELL	Knowledge graph	65,755	266,144	210	5,414
ogbn-arxiv	Citation network	169,343	1,166,243	40	128

Table 3.3: Statistics for node classification datasets.

3.4.1.2 Small Graph Classification

We use eleven datasets from two categories: (1) bioinformatics datasets: MUTAG, PTC-MR, COX2, BZR, NCI1, PROTEINS, ENZYMES, and D&D [64, 130, 254, 226, 239, 22]; (2) social network datasets: COLLAB, IMDB-B and RDT-M5K [277]. Table 3.4 below contains the statistics for the datasets used in our experiments on small graph classification.

Dataset	#Graphs	Avg #Nodes	Avg #Edges	#Classes
MUTAG	188	17.93	19.79	2
PTC-MR	344	14.29	14.69	2
BZR	405	35.75	38.36	2
COX2	467	41.22	43.45	2
ENZYMES	600	32.63	64.14	6
IMDB-B	1000	19.77	96.53	2
PROTEINS	1113	39.06	72.82	2
D & D	1178	284.32	715.66	2
NCI1	4110	29.87	32.30	2
RDT-M5K	5000	508.52	594.87	5
COLLAB	5000	74.49	2457.78	3

Table 3.4: Statistics for small graph classification datasets.

3.4.1.3 Large Graph Classification

We use five large graph datasets from Open Graph Benchmark (OGB) [108], including four molecular graph datasets (ogbg-molhiv, ogbg-moltox21, ogbg-moltoxcast and ogb-molpcba) and one protein-protein association network (ogbg-ppa). Table 3.5 contains the statistics for these five large graph datasets.

Dataset	#Graphs	Avg #Nodes	Avg #Edges	#Tasks	Task Type
ogbg-molmolhiv	41,127	25.5	27.5	1	Binary classification
ogbg-moltox21	7,831	18.6	19.3	12	Binary classification
ogbg-moltoxcast	8,576	18.8	19.3	617	Binary classification
ogbg-molpcba	437,929	26.0	28.1	128	Binary classification
ogbg-ppa	158,100	243.4	2,266.1	1	Multi-class classification

Table 3.5: Statistics for large graph classification dataset (OGB graph datasets).

3.4.2 Baseline Methods

We compare our GraphSNN model, as well as the variants of several existing GNN models under our GMP framework (GraphSNN$_{GCN}$, GraphSNN$_{GAT}$, GraphSNN$_{GIN}$ and GraphSNN$_{GraphSAGE}$), with the state-of-the-art methods in node classification (Section 3.4.2.1), small graph classification (Section 3.4.2.2), and large graph classification (Section 3.4.2.3).

3.4.2.1 Node Classification

We consider the popular message-passing GNNs: GCN [122], GAT [247], GIN [275], and GraphSAGE [99]. For each of these baselines, we construct a GraphSNN$_M$ model by replacing its aggregation scheme with our aggregation scheme, which is detailed in Section 3.2.2.

3.4.2.2 Small Graph Classification

We compare GraphSNN with standard and random data splittings against eleven baselines:

(1) *Graph kernel based methods:* Weisfeiler-Lehman subtree kernel (WL) [226], Return Probabilities of Random Walks Kernel (RetGK) [289], Graph Neural Tangent Kernel (GNTK) [68], P-WL [208], Weisfeiler-Lehman Pyramid Match Kernel (WL-PM) [182], Wasserstein Weisfeiler-Lehman Graph Kernel (WWL) [245] and Fused-Gromov Wasserstein (FGW) [243].

(2) *GNN based methods:* Deep Graph Convolutional Neural Network (DGCNN) [286], Capsule Neural Network (CapsGNN) [271], Graph Isomorphism Network (GIN) [275], and Inductive Representation Learning with Sampling and Aggregation (GraphSAGE) [99].

We also compare GraphSNN using cross-validation with inner hold-out method in Section 3.5.2.2, including: DGCNN [286] and GIN [275].

We also compare GraphSNN with the other GNNs that are more expressive than 1-WL in Section 3.5.2.3, including: Graph Substructure Networks (GSN) [23], Identity-Aware Graph Neural Networks (ID-GNNs) [280] and k-dimensional GNNs (k-WL GNNs) [175].

We also compare the oversmoothing analysis against two spatial GNNs baselines: Graph Convolutional Networks (GCN) [122] and GIN [275], and a spectral GNN baseline: Distributed Feedback-looped Networks (DFNets) [264].

3.4.2.3 Large Graph Classification

We compare against the following methods that have reported the results on the OGB datasets: GIN and Augmenting Graph Isomorphism Network with Virtual Nodes (GIN+VN) [108], GSN [23], Principal Neighbourhood Aggregation for Graph Nets (PNA) [54], ID-GNNs [280] and Deep Local Relation Pooling (Deep LRP) [48].

3.4.3 Hyperparameter Settings

In this section, we discuss the hyperparameters of GraphSNN and the GraphSNN$_M$ models, which are selected on the classification accuracy of the validation set by applying the randomized search strategy [15].

3.4.3.1 Node Classification

We consider two settings of data splits for all datasets except for ogbn-arxiv: (1) the standard splits in Kipf and Welling [122], i.e., 20 nodes from each class for training, 500 nodes for validation and 1000 nodes for testing, for which the results are presented in Table 3.6; (2) the random splits in Pei et al. [192], i.e., randomly splitting nodes into 60%, 20% and 20% for training, validation and testing, respectively. For ogbn-arxiv, we follow Hu et al. [108] to use a time-based data split based on publication dates.

We use the Adam optimizer [120] and set $\lambda = 1$. For ogbn-arxiv, our models are trained for 500 epochs with the learning rate 0.01, dropout 0.5, hidden units 256, and $\gamma = 0.1$. For the other datasets, we use 200 epochs with the learning rate 0.001, and choose the best values for weight decay from $\{0.001, 0.002, ..., 0.009\}$ and hidden units from $\{64, 128, 256, 512\}$. For γ and dropout at each layer, the best value for each model in each dataset is selected from $\{0.1, 0.2, ..., 0.6\}$.

GraphSNN$_{GAT}$ uses the attention dropout 0.6 and 8 multi-attention heads. GraphSNN$_{GraphSAGE}$ uses the neighborhood sample size 25 with the mean aggregation.

We also evaluate all the GraphSNN$_M$ models on Cora, Citeseer, and Pubmed datasets using the standard splits and same hyperparameter setup.

3.4.3.2 Small Graph Classification

Both the standard stratified splits [275] and the random splits are considered. We use 10-fold cross validation with 90% training and 10 % testing, and report the best mean accuracy. For both settings, we use the Adam optimizer [120], batch size 64, hidden dimension 64, weight decay of 0.009, a 2-layer MLP with batch normalization, 500 epochs and dropout of 0.6, and $\gamma = 0.1$ over all datasets. The readout function as in [275] is used which concatenates representations of all layers to obtain a final graph representation. For the standard stratified splits, we use the learning rate 0.009 over all datasets. For the random splits, we use the learning rate 0.008 for MUTAG and RDT-M5K, and 0.007 for the other datasets.

Previously, several experimental setups have been considered for evaluating graph classification on small graphs in TUD benchmark datasets (https://chrsmrrs.github.io/datasets/). All the baseline methods in our paper use the 10-fold cross validation technique. However, they differ in how they split training/validation/testing data and how they report the final results in terms of classification accuracy. Below, we discuss the details of their experimental setups.

- CapsGNN [271] splits the datasets into 80 % for training, 10 % for validation, and 10 % for testing. The training is stopped when the performance on the

validation set goes to the highest. Then they obtain the test set accuracy that corresponds to the epoch with the highest validation accuracy in each fold. The final results are reported by computing the mean accuracy and standard deviation over 10 folds.

- DGCNN [286] splits the datasets into 90 % for training and 10 % for testing. They obtain the test accuracy of the last epoch in each fold. They report the final results by computing the mean accuracy and standard deviation on the test accuracy over 10 folds.
- GIN and GraphSAGE [275] split the datasets into 90 % for training and 10 % for testing. They average the test accuracy on 10 folds and select the epoch with the highest averaged accuracy. Then they report the final results by computing the mean accuracy and standard deviation based on the selected epoch.
- FGW [243] splits the datasets into 90 % for training and 10 % for testing. Then, they use the nested cross validation technique on the same folds, and repeat the process 10 times. They report the final results by computing the mean accuracy and standard deviation.
- The other baseline methods split the datasets into 90 % for training and 10 % for testing, and repeat their experiment 10 times. Then they report the final results by computing the mean accuracy and standard deviation.

In our work, we split the datasets into 90 % for training and 10 % for testing. We obtain the best validation accuracy on each fold. Then we report the final results by computing the mean accuracy and standard deviation over 10 folds.

Following the data splits and experiment setup introduced in [73], we further evaluate our method using cross-validation with the inner hold-out method in Section 3.5.2.2. The experimental setup in [73] provides a fair performance comparison process on GNN methods. The evaluation process has two different phases: (1) model selection on the validation set, (2) model assessment on the test set. More specifically, they first split the datasets into 90 % for training and 10 % for testing. Then the entire training set is further split into 90% of training and 10% of validation. They apply the inner hold-out method to select the best model based on validation accuracy. After selecting the best model, they train the model three times on the entire training set with early stopping.

We further evaluate our GraphSNN method following the data splits and experimental setups introduced in [23, 280, 175]. In this setup, we evaluate GraphSNN with GNNs that are more expressive than 1-WL. GSN and ID-GNNs use the same experimental setup as GIN, while k-WL GNNs uses the same experimental setup as CapsGNN.

3.4.3.3 Large Graph Classification

In addition to the original model of GraphSNN, we also consider a variant, denoted as GraphSNN+VN, which performs the message passing over augmented graphs

with virtual nodes in GraphSNN [108, 111].

We follow the same experimental setup as in [108]. We use the Adam optimizer with learning rate 0.001, batch size 32, dropout 0.5 and 100 epochs for all datasets. GraphSNN uses a 8-layer MLP with embedding dimension 512 for ogbg-moltoxcast and ogbg-moltox21, while GraphSNN+VN has the embedding dimensions 300 and 256, and 8-layer and 5-layer MLPs for ogbg-moltoxcast and ogbg-moltox21, respectively. For ogbg-molhiv, ogbg-molpcba and ogbg-ppa, both GraphSNN and GraphSNN+VN use a 5-layer MLP and embedding dimension 200.

3.5 Results and Discussion

In this section, we discuss our experimental results to answer the aforementioned questions in Section 3.4.

3.5.1 Comparison with Node Classification

In this section, we evaluate the performance of GraphSNN$_M$ on node classification tasks to answer the question **Q1**.

Table 3.6 summarizes the results of semi-supervised node classification with standard splits. It shows that integrating our aggregation method with the existing GNNs, significantly improves performance on all benchmark datasets. Specifically, GraphSN$_{GCN}$ improves upon GCN by a margin of 1.60%, 2.00%, 0.80%, 2.30% and 0.46% on Cora, Citeseer, Pubmed, NELL and ogbn-arxiv, respectively. GraphSN$_{GAT}$ improves upon GAT by 0.80%, 0.90% and 1.10% on Cora, Citeseer and Pubmed, respectively. Similarly, GraphSN$_{GIN}$ improves upon GIN by 1.60%, 2.20%, 1.80% and 2.30% on Cora, Citeseer, Pubmed and NELL, respectively. GraphSN$_{GraphSAGE}$ improves upon GraphSAGE by 1.30%, 1.10%, 1.60%, 2.60% and 0.31% on Cora, Citeseer, Pubmed, NELL and ogbn-arxiv, respectively.

Table 3.7 shows the results of full-supervised node classification with random splits. We cal also see that our models consistently outperform all of the baseline methods on all benchmark datasets in this setting as well. Specifically, GraphSN$_{GCN}$ improves upon GCN by a margin of 1.50%, 1.70%, 1.60% and 2.40% on Cora, Citeseer, Pubmed and NELL, respectively. GraphSN$_{GAT}$ improves upon GAT by 1.30%, 1.60% and 2.00% on Cora, Citeseer and Pubmed, respectively. GraphSN$_{GIN}$ improves upon GIN by 3.80%, 1.70%, 1.80% and 1.60% on Cora, Citeseer, Pubmed and NELL, respectively. GraphSN$_{GraphSAGE}$ improves upon GraphSAGE by 1.30%, 1.70%, 1.10% and 2.30% on Cora, Citeseer, Pubmed and NELL, respectively.

3.5.2 Comparison with Small Graph Classification

In this section, we evaluate the performance of GraphSNN on small graph classification task to answer the question **Q2**.

Method	Cora	Citeseer	Pubmed	NELL	ogbn-arxiv
GCN	81.5 ± 0.4	70.3 ± 0.5	79.0 ± 0.5	66.0 ± 1.7	71.74 ± 0.29
GraphSNN$_{GCN}$	**83.1 ± 1.8**	**72.3 ± 1.5**	**79.8 ± 1.2**	**68.3 ± 1.6**	**72.20 ± 0.90**
GAT	83.0 ± 0.6	72.6 ± 0.6	78.5 ± 0.3	-	-
GraphSNN$_{GAT}$	**83.8 ± 1.2**	**73.5 ± 1.6**	**79.6 ± 1.4**	-	-
GIN	77.6 ± 1.1	66.1 ± 1.5	77.0 ± 1.2	61.5 ± 2.3	-
GraphSNN$_{GIN}$	**79.2 ± 1.7**	**68.3 ± 1.5**	**78.8 ± 1.3**	**63.8 ± 2.7**	-
GraphSAGE	79.2 ± 3.7	71.6 ± 1.9	77.4 ± 2.2	63.7 ± 5.2	71.49 ± 0.27
GraphSNN$_{GraphSAGE}$	**80.5 ± 2.5**	**72.7 ± 3.2**	**79.0 ± 3.5**	**66.3 ± 5.6**	**71.80 ± 0.70**

Table 3.6: Classification accuracy (%) averaged over 10 runs on node classification (standard splits).

Method	Cora	Citeseer	Pubmed	NELL
GCN	85.7 ± 1.6	73.6 ± 1.0	88.1 ± 1.2	72.2 ± 5.6
GraphSNN$_{GCN}$	**87.2 ± 1.5**	**75.3 ± 1.3**	**89.7 ± 1.7**	**74.6 ± 6.3**
GAT	86.3 ± 0.3	74.3 ± 0.3	87.6 ± 0.1	-
GraphSNN$_{GAT}$	**87.6 ± 0.9**	**75.9 ± 0.8**	**89.6 ± 0.6**	-
GIN	82.5 ± 0.8	70.8 ± 1.9	85.0 ± 1.5	66.7 ± 3.3
GraphSNN$_{GIN}$	**86.3 ± 0.7**	**72.5 ± 1.5**	**86.8 ± 1.2**	**68.3 ± 3.7**
GraphSAGE	86.8 ± 1.9	74.2 ± 1.8	88.3 ± 1.1	69.4 ± 4.3
GraphSNN$_{GraphSAGE}$	**88.1 ± 1.5**	**75.9 ± 1.3**	**89.4 ± 2.4**	**71.7 ± 4.5**

Table 3.7: Classification accuracy (%) averaged over 10 random splits on node classification.

Method	MUTAG	PTC-MR	PROTEINS	D&D	BZR	COX2	IMDB-B	RDT-M5K
WL	90.4 ± 5.7	59.9 ± 4.3	75.0 ± 3.1	79.4 ± 0.3	78.5 ± 0.6	81.7 ± 0.7	73.8 ± 3.9	52.5 ± 2.1
RetGK	90.3 ± 1.1	62.5 ± 1.6	75.8 ± 0.6	81.6 ± 0.3	-	-	71.9 ± 1.0	-
GNTK	90.0 ± 8.5	**67.9 ± 6.9**	75.6 ± 4.2	75.6 ± 3.9	83.6 ± 2.9	-	76.9 ± 3.6	-
P-WL	90.5 ± 1.3	64.0 ± 0.8	75.2 ± 0.3	78.6 ± 0.3	-	-	-	-
WL-PM	87.7 ± 0.8	61.4 ± 0.8	-	78.6 ± 0.2	-	-	-	-
WWL	87.2 ± 1.5	66.3 ± 1.2	74.2 ± 0.5	79.6 ± 0.5	84.4 ± 2.0	78.2 ± 0.4	74.3 ± 0.8	-
FGW	88.4 ± 5.6	65.3 ± 7.9	74.5 ± 2.7	-	85.1 ± 4.1	77.2 ± 4.8	63.8 ± 3.4	-
DGCNN	85.8 ± 1.7	58.6 ± 2.5	75.5 ± 0.9	79.3 ± 0.9	-	-	70.0 ± 0.9	48.7 ± 4.5
CapsGNN	86.6 ± 6.8	66.0 ± 1.8	76.2 ± 3.6	75.4 ± 4.1	-	-	73.1 ± 4.8	52.9 ± 1.5
†GraphSAGE	85.1 ± 7.6	63.9 ± 7.7	75.9 ± 3.2	72.9 ± 2.0	-	-	72.3 ± 5.3	50.0 ± 1.3
†GIN	89.4 ± 5.6	64.6 ± 7.0	75.9 ± 2.8	-	-	-	75.1 ± 5.1	57.5 ± 1.5
†GraphSNN (S)	**91.57 ± 2.8**	66.70 ± 3.7	**76.83 ± 2.5**	**81.97 ± 2.6**	**88.69 ± 3.2**	**82.86 ± 3.1**	**77.86 ± 3.6**	**58.43 ± 2.3**
†GraphSNN (R)	**91.24 ± 2.5**	66.96 ± 3.5	**76.51 ± 2.5**	**82.46 ± 2.7**	**88.97 ± 2.9**	**83.13 ± 3.5**	**76.93 ± 3.3**	**58.51 ± 2.7**
GraphSNN (S)	**94.70 ± 1.9**	**70.58 ± 3.1**	**78.42 ± 2.7**	**83.92 ± 2.3**	**91.12 ± 3.0**	**86.28 ± 3.3**	**78.51 ± 2.8**	**59.86 ± 2.6**
GraphSNN (R)	**94.14 ± 1.2**	**71.01 ± 3.6**	**78.21 ± 2.9**	**84.61 ± 1.5**	**91.88 ± 3.2**	**86.72 ± 2.9**	**77.87 ± 3.1**	**60.23 ± 2.2**

Table 3.8: Classification accuracy (%) averaged over 10 runs on graph classification. The results of WL and RetGK are taken from [68], GraphSAGE from [275], DGCNN from [167] and others from their original papers. † indicates the reporting setting used in GIN.

3.5.2.1 Small Graph Classification with Standard and Random Data Splittings

Table 3.8 summarizes the results. In this table, †GraphSNN (S) and †GraphSNN (R) indicate the results of cross-validation with standard stratified splits and random splits, respectively. We can see GraphSNN consistently improves the performance over all datasets except PTC-MR. First, we analyze the results of †GraphSNN (S). Compared with the graph kernels, GraphSNN improves upon their best results by a margin of 1.07%, 1.03%, 0.37%, 3.59%, 1.16%, 0.96% and 5.93% on MUTAG, PROTEINS, D&D, BZR, COX2, IMDB-B and RDT-M5k, respectively. Similarly, GraphSNN improves upon the best results of the GNNs, by a margin of 2.17%, 0.70%, 0.63%, 2.67%, 2.76% and 0.93% on datasets MUTAG, PTC-MR, PROTEINS, D&D, IMDB-B and RDT-M5K, respectively.

Then, we analyze the results of †GraphSNN (R). Compared with the graph kernels, GraphSNN improves upon their best results by a margin of 0.74%, 0.71%, 0.86%, 3.87%, 1.43%, 0.03% and 6.01% on MUTAG, PROTEINS, D&D, BZR, COX2, IMDB-B and RDT-M5k, respectively. Similarly, GraphSNN improves upon the best results of the GNNs, by a margin of 1.84%, 0.96%, 0.31%, 3.16%, 1.83% and 1.01% on datasets MUTAG, PTC-MR, PROTEINS, D&D, IMDB-B and RDT-M5K, respectively.

3.5.2.2 Small Graph Classification using Inner Hold-out Method

We have also conducted additional experiments on four bioinformatics datasets (NCI1, PROTEINS, ENZYMES and D&D) and three social network datasets (COLLAB, IMDB-B and REDDIT-5k) with the setting of cross-validation with the inner hold-out method [73]. The results of the baseline, DGCNN and GIN are taken from the paper [73]. Note that the final results of DGCNN and GIN from the paper [73] are reported by computing the mean accuracy and standard deviation on the test set in these three runs, which are different from the original papers of DGCNN and GIN.

Table 3.9 summarizes the results of small graph classification with inner hold-out method. We can see GraphSNN improves the performance over all datasets except PROTEINS, ENZYMES and D&D. Specifically, GraphSNN improves upon GIN by a margin of 1.60%, 1.20%, 2.1%, 1.8%, 1.40%, 1.10% and 1.00% on NCI1, PROTEINS, ENZYMES, D&D, COLLAB, IMDB-B and REDDIT-5k, respectively. Similarly, GraphSNN improves upon DGCNN by 5.20%, 1.60%, 22.8%, 0.5%, 5.8%, 3.1% and 7.9% on NCI1, PROTEINS, ENZYMES, D&D, COLLAB, IMDB-B and REDDIT-5k, respectively.

Method	NCI1	PROTEINS	ENZYMES	D&D	COLLAB	IMDB-B	REDDIT-5k
Baseline	69.8±2.2	**75.8 ± 3.7**	**65.2±6.4**	**78.4 ± 4.5**	70.2±1.5	70.8±5.0	52.2±1.5
DGCNN	76.4±1.7	72.9±3.5	38.9±5.7	76.6±4.3	71.2±1.9	69.2±3.0	49.2±1.2
GIN	80.0±1.4	73.3±4.0	59.6±4.5	75.3±2.9	75.6±2.3	71.2±3.9	56.1±1.7
GraphSNN	**81.6 ± 2.8**	74.5 ± 3.5	61.7 ± 3.4	77.1 ± 3.3	**77.0 ± 3.1**	**72.3 ± 3.6**	**57.1 ± 3.1**

Table 3.9: Classification accuracy (%) averaged over 10 runs on graph classification.

	Method	MUTAG	PTC-MR	PROTEINS	BZR	IMDB-B
GSN	GSN-e	90.6 ± 7.5	$\mathbf{68.2 \pm 7.2}$	76.6 ± 5.0	-	77.8 ± 3.3
	GSN-v	92.2 ± 7.5	67.4 ± 5.7	74.5 ± 5.0	-	76.8 ± 2.0
ID-GNNs	ID-GNN Fast	$\mathbf{96.5 \pm 3.2}$	61.9 ± 5.4	$\mathbf{78.0 \pm 3.5}$	86.4 ± 3.0	-
	ID-GNN Full	93.0 ± 5.6	62.5 ± 5.3	77.9 ± 2.4	88.1 ± 4.0	-
Ours	GraphSNN	91.57 ± 2.8	66.70 ± 3.7	76.83 ± 2.5	$\mathbf{88.69 \pm 3.2}$	$\mathbf{77.86 \pm 3.6}$
k-WL GNNs	1-GNN$_{NT}$	82.7 ± 0.0	51.2 ± 0.0	-	-	69.4 ± 0.0
	1-GNN	82.2 ± 0.0	59.0 ± 0.0	-	-	71.2 ± 0.0
	1-2-3-GNN$_{NT}$	84.4 ± 0.0	59.3 ± 0.0	-	-	70.3 ± 0.0
	1-2-3-GNN	86.1 ± 0.0	60.9 ± 0.0	-	-	74.2 ± 0.0
Ours	GraphSNN	$\mathbf{87.30 \pm 3.1}$	$\mathbf{61.63 \pm 2.8}$	$\mathbf{74.01 \pm 3.2}$	$\mathbf{82.72 \pm 3.9}$	$\mathbf{74.81 \pm 3.5}$

Table 3.10: Classification accuracy (%) averaged over 10 runs on graph classification, where $\lambda = 2$. The results of the baselines are taken from their original papers.

3.5.2.3 Comparison with GNNs Beyond 1-WL

Table 3.10 shows the results of GraphSNN against GNNs that go beyond 1-WL. The GSN and ID-GNNs follow the same experimental setup as in GIN [275], and k-WL GNNs follow the experimental setup as in CapsGNN [271]. Specifically, compared with GSN and ID-GNNs, GraphSNN improves upon their best results by a margin of 0.59% and 0.06% on BZR and IMDB-B, respectively. Compared with k-WL GNNs, GraphSNN improves upon their best results by a margin of 1.20%, 0.73%, and 0.61% on MUTAG, PTC-MR and IMDB-B, respectively.

3.5.3 Comparison with Large Graph Classification

In this section, we evaluate the performance of GraphSNN on large graph classification task to answer the question **Q2**.

Table 3.11 summarizes the results of large graph classification. We can see GraphSNN+VN consistently improves the performance over all datasets. Specifically, GraphSNN+VN improves upon GIN+VN by a margin of 4.52%, 0.57%, 1.50%, 1.65% and 1.47% on ogbg-molhiv, ogbg-moltox21, ogbg-moltoxcast, ogbg-ppa and ogbg-molpcba, respectively. Similarly, GraphSNN improves upon GIN by a margin of 2.93%, 0.54%, 1.99%, 1.74% and 2.30% on ogbg-molhiv, ogbg-moltox21, ogbg-moltoxcast, ogbg-ppa and ogbg-molpcba, respectively.

3.5.4 Oversmoothing Analysis

In this section, we analyze the effectiveness of our method in alleviating the oversmoothing issue to answer the question **Q3**. Specifically, we analyze the impact of model depth (number of layers) on node classification performance. In addition to GCN and GraphSNN$_{GCN}$, we also compare these models with a residual connection (i.e., GCN+residual and GraphSNN$_{GCN}$+residual). GraphSNN is similar to the GIN if we ignore the structural coefficients. Thus, it is worth to analyze how GraphSNN performs against GIN when increasing the model depth. Moreover, we analyze how

Method	ogbg-molhiv	ogbg-moltox21	ogbg-moltoxcast	ogbg-ppa	ogbg-molpcba
GIN	75.58±1.40	74.91±0.51	63.41±0.74	68.92±1.00	22.66±0.28
GIN+VN	75.20±1.30	76.21±0.82	66.18±0.68	70.37±1.07	27.03±0.23
GSN	77.99±1.00	-	-	-	-
PNA	79.05±1.30	-	-	-	28.38±0.35
ID-GNN	78.30±2.00	-	-	-	-
Deep LRP	77.19±1.40	-	-	-	-
GraphSNN	78.51±1.70	75.45±1.10	65.40±0.71	70.66±1.65	24.96±1.50
GraphSNN+VN	**79.72±1.83**	**76.78±1.27**	**67.68±0.92**	**72.02±1.48**	**28.50±1.68**

Table 3.11: Classification accuracy (%) averaged over 10 runs on graph classification, where $\lambda = 2$. The results of the baselines are taken from [108] and the leaderboard of the OGB website.

GraphSNN performs against spectral methods as well. Thus, we also compare GIN [275], DFNets [264], GraphSNN$_{GIN}$ and GraphSNN$_{GCN}$. For a fair comparison, we remove the dense-net architecture of DFNets and use the same hyperparameters as in the original paper.

Table 3.12 shows the results. When increasing the model depth, GraphSNN$_{GCN}$ performs consistently better than GCN at each layer. This is because structural coefficients capture structural connectivity between a target vertex and its neighbors. Thus, a neighbor whose structural connectivity is weak would pass little messages to the target vertex, whereas a neighbor whose structural connectivity is strong would pass a strong message to the target vertex. GraphSNN helps alleviate the oversmoothing issue even in the presence of residual connections.

#Layers	GCN	GCN+residual	GraphSNN$_{GCN}$	GraphSNN$_{GCN}$+residual
1	79.6±0.5	80.3±0.7	80.1±0.8	81.6±1.6
2	**81.5±0.4**	**82.8±1.2**	**83.1±1.8**	**84.1±1.7**
3	80.3±0.6	82.3±0.5	82.0±0.8	83.4±0.7
4	78.2±0.9	81.5±0.9	80.1±0.7	82.9±0.9
5	74.3±1.3	81.0±1.3	79.1±1.2	82.3±0.3
6	35.6±1.5	80.6±0.5	76.5±1.3	81.5±1.2
7	31.6±0.9	79.7±0.6	76.3±1.3	80.9±0.9
8	16.2±1.2	78.4±1.1	75.7±1.2	80.3±1.3

Table 3.12: Classification accuracy (%) averaged over 10 runs on Cora dataset.

Figure 3.4 shows the results of GCN and GraphSNN$_{GCN}$ on the datasets Cora, Citeseer and Pubmed, in terms of classification accuracy averaged over 10 runs in the setting of standard splits. GCN and GraphSNN$_{GCN}$ performance decreases over the number of layers when the number of layers > 2. Specifically, we can see performance of GCN suddenly drops on Cora when the number of layers > 5. We can also observe the similar pattern on Citeseer and Pubmed when the number of layers > 5 and number of layers > 6, respectively. Moreover, GraphSNN$_{GCN}$ and GCN have the similar trend when layers < 3, where both methods get their peak values with 2-layer GNNs on all three datasets. Compared to GCN, performance drop of GraphSNN$_{GCN}$ is very small when increasing the number of layers.

#Layers	GIN	GraphSNN$_{GIN}$	DFNet	GraphSNN$_{GCN}$
1	73.3±1.5	76.1±1.6	80.5±0.6	80.1±0.8
2	**77.6±1.3**	**79.2±1.7**	81.9±0.5	**83.1±1.8**
3	75.2±1.7	78.5±1.3	**82.6±0.3**	82.0±0.8
4	48.6±2.1	77.2±2.3	80.7±0.6	80.1±0.7
5	40.3±1.9	75.9±2.1	75.6±0.3	79.1±1.2
6	36.1±2.3	73.3±1.8	65.3±1.3	76.5±1.3
7	27.5±2.1	71.9±1.5	60.9±1.5	76.3±1.3
8	20.3±1.8	69.3±2.2	53.6±1.3	75.7±1.2

Table 3.13: Oversmoothing analysis of GIN and spectral GNN (DFNet) on cora dataset.

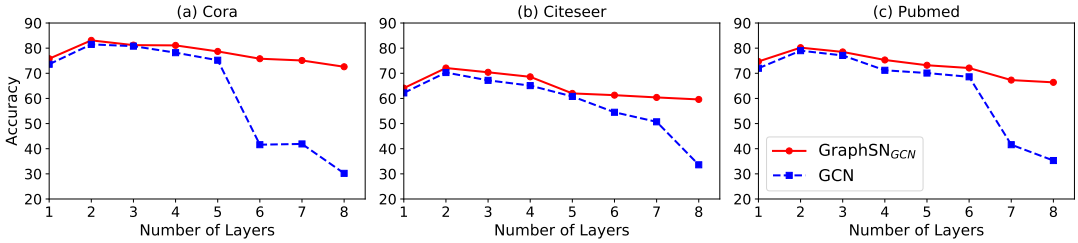


Figure 3.4: Oversmoothing analysis w.r.t. the model depth for node classification.

3.5.5 Runtime Analysis

In this section, we provide the runtime analysis for computing the structural coefficients of GraphSNN to answer the question **Q4**. The layer-wise massage-passing of GraphSNN is linear w.r.t number of edges on graphs as previous GNNs such as GIN. However, GraphSNN requires linear computational complexity w.r.t the number of edges on a graph for pre-computing the structural coefficients. Thus, in Table 3.14, we analyze the results for the running time of the prepocessing step in our method GraphSNN for large graph datasets (averaged over 5 runs). Note that the preprocessing step can be parallellized efficiently at the node level. The CPU time shows the total preprocessing time of a dataset in which each node is preprocessed sequentially, and the CPU time per node shows the average preprocessing time per node.

Dataset	CPU time (seconds)	CPU time per node (milliseconds)
ogbg-molhiv	66.97	0.06383
ogbg-moltox21	79.37	0.54565
ogbg-moltoxcast	380.84	2.36417
ogbg-ppa	820.12	4.71235

Table 3.14: Running time of the prepocessing step for large graph datasets averaged over 5 runs.

Dataset	Method	λ=1	λ=2	λ=3	λ=4	λ=5
Cora	GraphSNN$_{GCN}$	**83.1±1.8**	82.8±1.3	82.3±2.4	81.8±1.6	82.1±1.6
	GraphSNN$_{GIN}$	**79.2±1.7**	78.8±1.2	78.5±1.3	78.1±1.6	77.7±1.2
	GraphSNN$_{GraphSAGE}$	**80.5±2.5**	80.3±2.1	79.8±1.9	79.2±1.9	79.4±2.2
	GraphSNN$_{GAT}$	**83.8±1.2**	83.5±1.5	83.2±1.7	82.8±1.3	83.2±1.9
Citeseer	GraphSNN$_{GCN}$	**72.3±1.5**	71.7±1.3	71.1±1.6	70.6±1.2	70.9±1.1
	GraphSNN$_{GIN}$	**68.3±1.5**	68.3±1.9	67.7±1.4	67.1±1.3	67.3±1.4
	GraphSNN$_{GraphSAGE}$	**72.7±3.2**	72.0±2.5	71.6±2.9	71.9±2.1	71.3±2.3
	GraphSNN$_{GAT}$	**73.5±1.6**	72.9±1.7	72.5±1.1	72.6±1.6	72.0±1.3

Table 3.15: Classification accuracy (%) averaged over 10 runs on node classification with standard splits.

Dataset	Method	λ=1	λ=2	λ=3	λ=4	λ=5
MUTAG	GraphSNN	92.66±2.4	**94.14±1.2**	93.38±1.5	92.25±2.1	92.79±2.0
PTC-MR		70.76±5.1	**71.01±3.6**	70.67±2.8	69.59±2.1	69.97±3.1
PROTEINS		77.90±4.9	**78.21±2.9**	78.15±2.1	77.20±3.1	76.93±3.2
D&D		82.70±4.6	**84.61±1.5**	84.34±1.2	82.60±2.6	82.30±2.3
BZR		87.61±4.9	**91.88±3.2**	91.45±2.6	91.38±2.1	90.90±3.1
COX2		86.20±3.3	**86.72±2.9**	83.81±3.1	83.13±2.6	83.94±3.2
IMDB-B		77.07±5.2	**77.87±3.1**	77.60±3.6	77.32±3.2	77.10±3.3
RDT-M5K		59.53±2.6	**60.23±2.2**	60.10±2.3	60.00±2.1	59.90±2.6

Table 3.16: Classification accuracy (%) averaged over 10 runs on graph classification with random splits.

3.5.6 Ablation Analysis with λ

In this section, we perform an ablation study to analyze the effect of λ values on the performance of GraphSNN$_M$ models for node classification tasks and GraphSNN for graph classification tasks to answer the question **Q5**. Tables 3.15 and 3.16 show that $\lambda = 1$ yields the highest performance for node classification, while $\lambda = 2$ is the best for graph classification. This reflects a critical point - different types of structural information are needed by different graph learning tasks:

(1) $\lambda = 1$ captures local density, e.g., two overlap subgraphs may considerably vary in the number of vertices but their local density can be very close. Our experiments show that injecting such local density helps improve the performance of node classification.

(2) $\lambda = 2$ captures local similarity, i.e., how similar two overlap subgraphs are. Two overlap subgraphs that considerably differ in the number of vertices would have very different structural coefficients. Since graph classification requires to compare the similarity of two graphs, $\lambda = 2$ is thus the best.

3.5.7 Augmented Cycle and Clique Counts for Node Features

In this section, we evaluate the overall performance of GraphSNN by augmenting node features on small graph classification task to answer the question **Q5**. This

experimental setup allows us to analyze what types of local substructures our proposed model can distinguish. We consider an experimental setup called BL, which serves as the baseline for all experiments in this ablation study. In the setting of BL, the AGGREGATE^I in GraphSNN is set to 1. Then, different variants of BL consider different local substructure counts as additional node features.

There are five variants of BL being considered in the ablation study:

(1) BL_{SC}: Setting AGGREGATION^I of GraphSNN to 1 and keeping structural coefficients for neighbors.

(2) BL_{NF}^{clique}: Setting AGGREGATION^I of GraphSNN to 1, removing structural coefficients for neighbors, and adding additional node features (triangle and 4-clique counts) into the original feature vectors.

(3) $\text{BL}_{SC+NF}^{clique}$: Setting AGGREGATION^I of GraphSNN to 1, keeping structural coefficients for neighbors, and adding additional node features (triangle and 4-clique counts) into the original feature vectors.

(4) BL_{NF}^{cycle}: Setting AGGREGATION^I of GraphSNN to 1, removing structural coefficients for neighbors, and adding additional node features (cycle counts) into the original feature vectors.

(5) $\text{BL}_{SC+NF}^{cycle}$: Setting AGGREGATION^I of GraphSNN to 1, keeping structural coefficients for neighbors, and adding additional node features (cycle counts) into the original feature vectors.

We compare GraphSNN with GSN-v [23], BL_{NF}^{clique}, BL_{SC}, and $\text{BL}_{SC+NF}^{clique}$ to analyze how our proposed model relates to the models with triangle and 4 clique counts as additional node features. Similarly, we compare GraphSNN with ID-GNNs [280], BL_{NF}^{cycle}, BL_{SC}, and $\text{BL}_{SC+NF}^{cycle}$ to analyze how our proposed architecture relates to the models with cycle counts as additional node features. We concatenate the counts of cycles with length 1 to 4 starting and ending at the given source node with its original feature vector as in [280].

Table 3.17 and Table 3.18 show the experimental results. As AGGREGATE^I is set to 1 in the setting of BL, the performance gap between BL_{NF} and BL_{SC+NF} reflects the effectiveness of structural coefficients on enhancing relational inference between a target vertex and its neighbors. The performance gap between BL_{SC} and GraphSNN above shows the effectiveness of AGGREGATE^I in our proposed model GraphSNN. Furthermore, BL_{SC+NF} consistently performs best since we incorporate both extra node features and structural coefficients into the feature aggregation. There is a small performance gap between BL_{SC+NF} and GraphSNN due to augmented node features that can capture additional structural information that cannot be captured using structural coefficients.

Method	GSN-v	BL_{NF}^{clique}	BL_{SC}	BL_{SC+NF}^{clique}	GraphSNN
MUTAG	92.20±7.5	90.21±2.3	94.06±2.4	**95.16±2.5**	94.70±1.9
PTC-MR	67.40±5.7	67.13±2.9	70.18±3.1	**71.04±3.1**	70.58±3.1
PROTEINS	74.59±5.0	76.42±2.6	78.05±2.3	**78.66±2.1**	78.42±2.7
BZR	-	86.82±3.1	90.67±3.1	**91.98±3.2**	91.12±3.0
IMDB-B	76.80±2.0	77.00±3.1	77.23±2.8	**78.53±2.9**	78.01±2.8

Table 3.17: Analysis the effects of our structural coefficients with substructure counts, i.e, triangle and 4-clique counts. Classification accuracy (%) averaged over 10 runs on graph classification.

Method	ID-GNN	BL_{NF}^{cycle}	BL_{SC}	BL_{SC+NF}^{cycle}	GraphSNN
MUTAG	96.50±3.2	91.36±2.1	94.06±2.4	**96.61±2.3**	94.70±1.9
PTC-MR	61.90±5.4	67.57±3.3	70.18±3.1	**71.76±3.2**	70.58±3.1
PROTEINS	78.00±3.5	77.26±2.5	78.05±2.3	**78.95±2.5**	78.42±2.7
BZR	86.40±3.0	86.83±3.3	90.67±3.1	**91.75±3.4**	91.12±3.0
IMDB-B	-	76.36±2.6	77.23±2.8	**78.58±2.4**	78.01±2.8

Table 3.18: Analysis the effects of our structural coefficients with substructure counts, i.e, cycle counts. Classification accuracy (%) averaged over 10 runs on graph classification.

3.6 Summary

In this chapter we have proposed a new perspective on designing powerful Graph Neural Networks (GNNs)*. In a nutshell, this enables a general solution to inject structural properties of graphs into a message-passing aggregation scheme of GNNs. As a theoretical basis, we developed a new hierarchy of local isomorphism on neighborhood subgraphs. Then, we theoretically characterized how message-passing GNNs can be designed to be more expressive than the Weisfeiler Lehman test. To elaborate this characterization, we proposed a novel neural model, called *GraphSNN*, and proved that this model is strictly more expressive than the Weisfeiler Lehman test in distinguishing graph structures. We empirically verified the strength of our model on different graph learning tasks. It is shown that our model consistently improves the state-of-the-art methods on the benchmark tasks without sacrificing computational simplicity and efficiency.

*GraphSNN implementation can be found at: https://github.com/wokas36/GraphSNN

Chapter 4

Feedback-looped Filters for Spectral Graph Neural Networks

4.1 Overview

To effectively tackle the problem of designing effective, yet efficient spectral GNNs, a number of studies has been devoted to enhancing GNNs by developing filters over graphs. Compared to spatial GNNs in the previous chapter, spectral GNNs use spectral graph filters to indirectly define convolutions on graphs [28, 53, 65]. Bruna et al. [28] proposed convolution operations on graphs via a spectral decomposition of the graph Laplacian. To reduce learning complexity in the setting where the graph structure is not known a priori, Henaff et al. [104] developed a smooth parameteric spectral graph filter, where the number of filter parameters is independent from the size of an input graph. Then, Defferrard et al. [65] introduced Chebyshev filters to guarantee the stability of convolution operations under graph perturbation and these filters can be exactly localized in k-hop neighborhood. Later, Kipf et al. [122] proposed a simple layer-wise propagation model using Chebyshev filters on 1-hop neighborhood. Over the last few years, some works attempted to develop rational polynomial filters, such as Cayley filters [137] and ARMA [20]. From a different perspective, Petar et al. [247] proposed a self-attention based GNN architecture for graph filters, which extracts features by considering the importance of neighbors.

In this chapter, we study the problem of designing an effective and efficient GNN with a spectral graph filtering. The key idea behind existing works on designing spectral graph filters is to approximate the frequency responses of graph filters using a polynomial function (e.g. Chebyshev filters [65]) or a rational polynomial function (e.g. Cayley filters [137] and ARMA [20]). Polynomial filters are sensitive to changes w.r.t the underlying graph structure. They are also very smooth and can hardly model sharp changes. Rational polynomial filters are more powerful to model localization, but they often have to trade off computational efficiency, resulting in higher learning and computational complexities, as well as instability. In this chapter, we aim to overcome the above limitations.

The main contributions of this chapter are as follows:

- We propose a new class of spectral graph filters, called *feedback-looped filters*, to

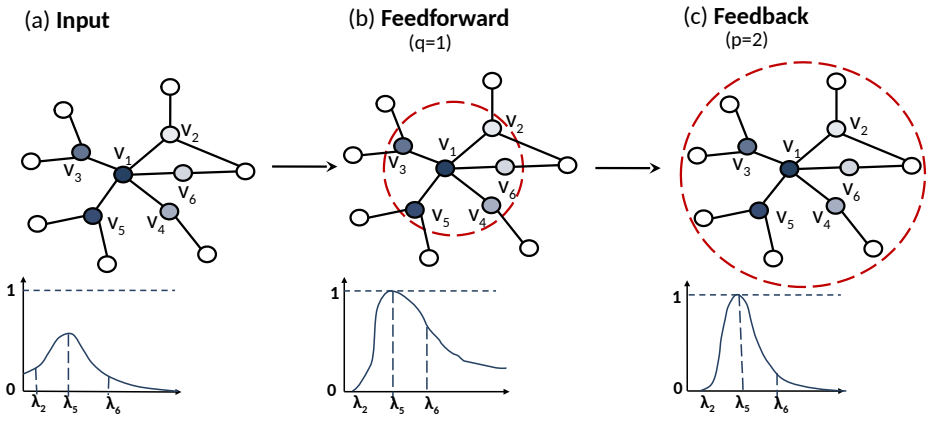


Figure 4.1: A simplified example of illustrating feedback-looped filters, where v_1 is the current vertex and the similarity of the colours indicates the correlation between vertices, e.g., v_1 and v_5 are highly correlated, but v_2 and v_6 are less correlated with v_1: (a) an input graph, where λ_i is the original frequency to vertex v_i; (b) the feedforward filtering, which attenuates some low order frequencies, e.g. λ_2, and amplify other frequencies, e.g. λ_5 and λ_6; (c) the feedback filtering, which reduces the error in the frequencies generated by (b), e.g. λ_6.

enable better localization, as illustrated in Figure 4.1. Basically, feedback-looped filters consist of two parts: *feedforward* and *feedback*. The feedforward filtering is k-localized as polynomial filters, while the feedback filtering is unique which refines k-localized features captured by the feedforward filtering to improve approximation accuracy. We also propose two techniques: *scaled-normalization* and *cut-off frequency* to avoid the issues of gradient vanishing/exploding and instabilities.

- For feedback-looped filters, we avoid the matrix inversion implied by the denominator through approximating the matrix inversion with a recursion. Thus, benefited from this approximation, feedback-looped filters attain linear convergence time and linear memory requirements w.r.t. the number of edges in a graph.

- Feedback-looped filters enjoy several nice theoretical properties. Unlike other rational polynomial filters for graphs, they have theoretically guaranteed convergence w.r.t. a specified error bound. On the other hand, they still have the universal property as other spectral graph filters [113], i.e., this graph filter allows to approximate any desired graph frequency response without realizing the underlying structure of a graph. The optimal coefficients of feedback-looped filters are learnable via an optimization condition for any given graph.

- We propose a layer-wise propagation rule for our spectral GNN model with feedback-looped filters, which densely connects layers as in DenseNet [109]. This design enables our model to diversify features from all preceding layers, leading to a strong gradient flow. We also introduce a layer-wise regularization

term to alleviate the overfitting issue. In doing so, we can prevent the generation of spurious features and thus improve accuracy of the prediction.

The rest of this chapter is organized as follows. In Section 4.2, we present the new class of spectral graph filters, formulates a coefficients optimization and a novel spectral GNN model. In Section 4.3, we analyze the convergence and complexity of the proposed spectral GNN model. In Section 4.4, we discuss the experimental setup. In Section 4.5, we compare the performance of our proposed spectral GNN method against the baseline methods. Section 4.6 summarises the chapter.

4.2 Spectral GNNs with Feedback-looped Filters

We introduce a new class of spectral graph filters, called *feedback-looped filters*, and propose a spectral GNN for graphs with feedback-looped filters, namely *Distributed Feedback-Looped Networks* (DFNets). We also discuss optimization techniques and analyze theoretical properties.

4.2.1 A New Class of Spectral Graph Filters

Feedback-looped filters belong to a class of Auto Regressive Moving Average (ARMA) filters [112, 113]. Formally, an ARMA filter is defined as:

$$h_{\psi,\phi}(\mathbf{L})x = \Big(\mathbf{I} + \sum_{j=1}^{p} \psi_j \mathbf{L}^j\Big)^{-1} \Big(\sum_{j=0}^{q} \phi_j \mathbf{L}^j\Big) x. \tag{4.1}$$

The parameters p and q refer to the *feedback* and *feedforward* degrees, respectively. $\psi \in \mathbb{C}^p$ and $\phi \in \mathbb{C}^{q+1}$ are two vectors of complex coefficients. Computing the denominator of Eq. 4.1 however requires a matrix inversion, which is computationally inefficient for large graphs. To circumvent this issue, *feedback-looped filters* use the following approximation:

$$\bar{x}^{(t)} = -\sum_{j=1}^{p} \psi_j \tilde{\mathbf{L}}^j \bar{x}^{(t-1)} + \sum_{j=0}^{q} \phi_j \tilde{\mathbf{L}}^j x, \tag{4.2}$$

where $\bar{x}^{(0)} = x$, $\tilde{\mathbf{L}} = \hat{\mathbf{L}} - (\frac{\hat{\lambda}_{max}}{2})\mathbf{I}$, $\hat{\mathbf{L}} = \mathbf{I} - \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$, $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, $\hat{\mathbf{D}}_{ii} = \sum_j \hat{\mathbf{A}}_{ij}$ and $\hat{\lambda}_{max}$ is the largest eigenvalue of $\hat{\mathbf{L}}$. Accordingly, the frequency response of feedback-looped filters is defined as:

$$h(\lambda_i) = \frac{\sum_{j=0}^{q} \phi_j \lambda_i^j}{1 + \sum_{j=1}^{p} \psi_j \lambda_i^j}. \tag{4.3}$$

To alleviate the issues of gradient vanishing/exploding and numerical instabilities, we further introduce two techniques in the design of feedback-looped filters: *scaled-normalization* and *cut-off frequency*.

4.2.1.1 Scaled-normalization Technique

To assure the stability of feedback-looped filters, we apply the scaled-normalization technique to increase the stability region, i.e., using the scaled-normalized Laplacian $\tilde{\mathbf{L}} = \hat{\mathbf{L}} - (\frac{\hat{\lambda}_{max}}{2})\mathbf{I}$, rather than just $\hat{\mathbf{L}}$. This accordingly helps centralize the eigenvalues of the Laplacian $\hat{\mathbf{L}}$ and reduce its spectral radius bound. The scaled-normalized Laplacian $\tilde{\mathbf{L}}$ consists of graph frequencies within $[0, 2]$, in which eigenvalues are ordered in an increasing order.

4.2.1.2 Cut-off Frequency Technique

To map graph frequencies within $[0, 2]$ to a uniform discrete distribution, we define a *cut-off frequency* $\lambda_{cut} = (\frac{\lambda_{max}}{2} - \eta)$, where $\eta \in [0, 1]$ and λ_{max} refers to the largest eigenvalue of $\tilde{\mathbf{L}}$. The cut-off frequency is used as a threshold to control the amount of attenuation on graph frequencies. The eigenvalues $\{\lambda_i\}_{i=0}^{n-1}$ are converted to binary values $\{\tilde{\lambda}_i\}_{i=0}^{n-1}$ such that $\tilde{\lambda}_i = 1$ if $\lambda_i \geq \lambda_{cut}$ and $\tilde{\lambda}_i = 0$ otherwise. This trick allows the generation of ideal high-pass filters so as to sharpen a signal by amplifying its graph Fourier coefficients. This technique also solves the issue of narrow frequency bands existing in previous spectral filters (i.e., Cayley filters), including both polynomial and rational polynomial filters [65, 137]. This is because these previous spectral filters only accept a small band of frequencies. In contrast, our proposed feedback-looped filters resolve this issue using a cut-off frequency technique, i.e., amplifying frequencies higher than a certain low cut-off value while attenuating frequencies lower than that cut-off value. Thus, our proposed filters can accept a wider range of frequencies and capture better characteristic properties of a graph.

4.2.2 Learnable Optimal Coefficients

Given a feedback-looped filter with a desired frequency response: $\hat{h} : \{\tilde{\lambda}_i\}_{i=0}^{n-1} \rightarrow \mathbb{R}$, we aim to find the optimal coefficients ψ and ϕ that make the frequency response as close as possible to the desired frequency response, i.e. to minimize the following error:

$$\acute{e}(\tilde{\lambda}_i) = \hat{h}(\tilde{\lambda}_i) - \frac{\sum_{j=0}^{q} \phi_j \tilde{\lambda}_i^j}{1 + \sum_{j=1}^{p} \psi_j \tilde{\lambda}_i^j} \tag{4.4}$$

However, the above equation is not linear w.r.t. the coefficients ψ and ϕ. Thus, we redefine the error as follows:

$$e(\tilde{\lambda}_i) = \hat{h}(\tilde{\lambda}_i) + \hat{h}(\tilde{\lambda}_i) \sum_{j=1}^{p} \psi_j \tilde{\lambda}_i^j - \sum_{j=0}^{q} \phi_j \tilde{\lambda}_i^j. \tag{4.5}$$

Let $e = [e(\tilde{\lambda}_0), \ldots, e(\tilde{\lambda}_{n-1})]^T$, $\hat{h} = [\hat{h}(\tilde{\lambda}_0), \ldots, \hat{h}(\tilde{\lambda}_{n-1})]^T$, and $\varrho \in \mathbb{R}^{n \times p}$ with $\varrho_{ij} = \tilde{\lambda}_i^j$ and $\kappa \in \mathbb{R}^{n \times (q+1)}$ with $\kappa_{ij} = \tilde{\lambda}_i^{j-1}$ be two Vandermonde-like matrices. Then, we have $e = \hat{h} + diag(\hat{h})\varrho\psi - \kappa\phi$. Thus, the stable coefficients ψ and ϕ can be learned by

minimizing e as a convex constrained least-squares optimization problem:

$$\textbf{minimize}_{\psi,\phi} \; ||\hat{h} + diag(\hat{h})\varrho\psi - \kappa\phi||_2 \tag{4.6}$$

$$\textbf{subject to} \; ||\varrho\psi||_\infty \leq \epsilon \text{ and } \epsilon < 1$$

Here, the parameter ϵ controls the trade-off between convergence efficiency and approximation accuracy. A higher value of ϵ can lead to slower convergence but better accuracy. It is not recommended to have very low ϵ values due to potentially unacceptable accuracy. $||\varrho\psi||_\infty \leq \epsilon < 1$ is the stability condition, which will be further discussed in detail in Section 4.3.

4.2.3 A New Spectral Convolutional Layer

We propose a GNN-based architecture, called DFNets, which can stack multiple spectral convolutional layers with feedback-looped filters to extract features of increasing abstraction. Let $\mathbf{Y} = -\sum_{j=1}^{p} \psi_j \tilde{\mathbf{L}}^j$ and $\mathbf{Q} = \sum_{j=0}^{q} \phi_j \tilde{\mathbf{L}}^j$. The propagation rule of a spectral convolutional layer is defined as:

$$\bar{X}^{(t+1)} = \sigma(\boldsymbol{Y}\bar{X}^{(t)}\theta_1^{(t)} + \boldsymbol{Q}X\theta_2^{(t)} + \eta(\theta_1^{(t)}; \theta_2^{(t)}) + b), \tag{4.7}$$

where σ refers to a non-linear activation function such as ReLU. $\bar{X}^{(0)} = \mathbf{X} \in \mathbb{R}^{n \times f}$ is a graph signal matrix where f refers to the number of features. $\bar{X}^{(t)}$ is a matrix of activations at the t^{th} layer. $\theta_1^{(t)} \in \mathbb{R}^{c \times h}$ and $\theta_2^{(t)} \in \mathbb{R}^{f \times h}$ are two trainable weight matrices at the t^{th} layer. To compute $\bar{X}^{(t+1)}$, a vertex needs access to its p-hop neighbors with the output signal of the previous layer $\bar{X}^{(t)}$, and its q-hop neighbors with the input signal from $\mathbf{X}$. To attenuate the overfitting issue, we add $\eta(\theta_1^{(t)}; \theta_2^{(t)})$, namely *kernel regularization* [55], and a bias term b. We use the xavier normal initialization method [87] to initialise the kernel and bias weights, the unit-norm constraint technique [67] to normalise the kernel and bias weights by restricting the parameters of all layers in a small range, and the kernel regularization technique to penalize the parameters in each layer during the training. In doing so, we can prevent the generation of spurious features and thus improve the accuracy of prediction.

In this model, each layer is directly connected to all subsequent layers in a feed-forward manner, as in DenseNet [109]. Consequently, the t^{th} layer receives all preceding feature maps $F_0, \ldots, F_{t-1}$ as input. We concatenate multiple preceding feature maps column-wise into a single tensor to obtain more diversified features for boosting the accuracy. This densely connected GNN architecture has several compelling benefits: (a) reduce the vanishing-gradient issue, (b) increase feature propagation and reuse, and (c) refine information flow between layers [109].

4.3 Theoretical Analysis

Feedback-looped filters have several nice properties, e.g., guaranteed convergence, linear convergence time, and universal design. We discuss these properties and

analyze computational complexities in this section.

4.3.1 Convergence analysis

Theoretically, a feedback-looped filter can achieve a desired frequency response only when $t \to \infty$ [113]. However, due to the property of linear convergence preserved by feedback-looped filters, stability can be guaranteed after a number of iterations w.r.t. a specified small error [112]. More specifically, since the pole of rational polynomial filters should be in the unit circle of the z-plane to guarantee the stability, we can derive the stability condition $|| - \sum_{j=1}^{p} \psi_j \mathbf{L}^j|| < 1$ by Eq. 4.1 in the vertex domain and correspondingly obtain the stability condition $||\varrho\psi||_\infty \leq \epsilon \in (0, 1)$ in the frequency domain as stipulated in Eq. 4.6 [112].

4.3.2 Universal Design

The universal design is beneficial when the underlying structure of a graph is unknown or the topology of a graph changes over time. The corresponding filter coefficients can be learned independently of the underlying graph and are universally applicable. When designing feedback-looped filters, we define the desired frequency response function $\hat{h}$ over graph frequencies $\tilde{\lambda}_i$ in a binary format in the uniform discrete distribution as discussed in Section 4.2.1.2. Then, we solve Eq. 4.6 in the least-squares sense for this finite set of graph frequencies to find optimal filter coefficients.

4.3.3 Complexity Analysis

Spectral Graph Filter	Type	Learning Complexity	Time Complexity	Memory Complexity
Chebyshev filters [65]	Polynomial	$O(k)$	$O(km)$	$O(m)$
Lanczos filters [146]		$O(k)$	$O(km^2)$	$O(m^2)$
Cayley filters [137]	Rational polynomial	$O((r+1)k)$	$O((r+1)km)$	$O(m)$
ARMA_1 filters [20]		$O(t)$	$O(tm)$	$O(m)$
d parallel ARMA_1 filters [20]		$O(t)$	$O(tm)$	$O(dm)$
Feedback-looped filters (ours)		$O(tp+q)$	$O((tp+q)m)$	$O(m)$

Table 4.1: Learning, time and memory complexities of spectral graph filters.

Table 4.1 summarizes the complexity results of existing spectral graph filters and feedback-looped filters, where r refers to the number of Jacobi iterations in [137]. Note that, when $t = 1$ (i.e., one spectral convolutional layer), feedback-looped filters have the same learning, time and memory complexities as Chebyshev filters, where $p + q = k$. When computing $\bar{x}^{(t)}$ as in Eq. 4.2, we need to calculate $\tilde{\mathbf{L}}^j \bar{x}^{(t-1)}$ for $j = 1, \ldots, p$ and $\tilde{\mathbf{L}}^j x$ for $j = 1, \ldots, q$. Nevertheless, $\tilde{\mathbf{L}}^j x$ is computed only once because $\tilde{\mathbf{L}}^j x = \tilde{\mathbf{L}}(\tilde{\mathbf{L}}^{j-1} x)$. Thus, we need p multiplications for each t in the first term in Eq. 4.2, and q multiplications for the second term in Eq. 4.2.

4.4 Experimental Setup

To empirically verify the effectiveness of our work, we have evaluated feedback-looped filters with three different spectral GNN models: (i) DFNet: a densely connected spectral GNN with feedback-looped filters, (ii) DFNet-ATT: a self-attention based densely connected spectral GNN with feedback-looped filters, and (iii) DF-ATT: a self-attention based spectral GNN model with feedback-looped filters. In this section, we present the datasets and baseline methods considered when evaluating our DFNet models on semi-supervised node classification task in order to answer the following questions. We also discuss hyperparatemeter settings for each model.

Q1. How well can DFNets empirically perform over standard and random data splittings on semi-supervised node classification task?

Q2. How well can different neighborhood orders influence the overall performance on semi-supervised node classification task?

Q3. How effectively DFNets can learn the node representations to reveal the clustering quality of classes?

Q4. How does each of the key components in DFNets (i.e., cut-off frequency technique, scaled-normalization technique,and feedback-looped filters) contribute to the performance of DFNets?

4.4.1 Datasets

We use three citation network datasets Cora, Citeseer, and Pubmed [223] for semi-supervised document classification, and one dataset NELL [36] for semi-supervised entity classification. NELL is a bipartite graph extracted from a knowledge graph [36]. Table 3.3 in Chapter 3 contains the statistics for these four datasets.

4.4.2 Baseline Methods

We compare against twelve baseline methods, including five methods using non-spectral graph learning, i.e., Semi-supervised Embedding (SemiEmb) [263], Label Propagation (LP) [298], skip-gram graph embedding model (DeepWalk) [196], Iterative Classification Algorithm (ICA) [158], and semi-supervised learning with graph embedding (Planetoid*) [278], and seven methods using spectral graph learning: Chebyshev [65], Graph Convolutional Networks (GCN) [122], Lanczos Networks (LNet) and Adaptive Lanczos Networks (AdaLNet) [146], CayleyNet [137], Graph Attention Networks (GAT) [247], and ARMA Convolutional Networks (ARMA_1) [20].

4.4.3 Hyperparameter Settings

In this section, we discuss the hyperparameters of DFNet, DFNet-ATT and DF-ATT models, which are selected on the classification accuracy of the validation set by applying the randomized search strategy [15].

4.4.3.1 Semi-supervised Node Classification with Standard Data Splitting

We use the same data splitting for each dataset as in Yang et al. [278]. The hyperparameters of our models are initially selected by applying the orthogonalization technique (a randomized search strategy). We also use a layerwise regularization (L2 regularization) and bias terms to attenuate the overfitting issue. All models are trained 200 epochs using the Adam optimizer [120] with a learning rate of 0.002. Table 4.2 summarizes the hyperparameter settings for citation network datasets. The same hyperparameters are applied to the NELL dataset except for L2 regularization (i.e., 9e-2 for DFNet and DFnet-ATT, and 9e-4 for DF-ATT). For ϵ, we choose the best setting for each model. For self-attention, we use 8 multi-attention heads and 0.5 attention dropout for DFNet-ATT, and 6 multi-attention heads and 0.3 attention dropout for DF-ATT. The parameters $p = 5$, $q = 3$ and $\lambda_{cut} = 0.5$ are applied to all three models over all datasets.

Model	L2 reg.	#Layers	#Units	Dropout	[p, q]	λ_{cut}
DFNet	9e-2	5	[8, 16, 32, 64, 128]	0.9	[5, 3]	0.5
DFNet-ATT	9e-4	4	[8, 16, 32, 64]	0.9	[5, 3]	0.5
DF-ATT	9e-3	2	[32, 64]	[0.1, 0.9]	[5, 3]	0.5

Table 4.2: Hyperparameter settings for citation network datasets.

4.4.3.2 Semi-supervised Node Classification with Random Data Splittings

We have benchmarked the performance of our DFNet model against state-of-the-art methods over three citation network datasets Cora, Citeseer and Pubmed. We use the same random data splittings as used in [146]. All the experiments are repeated 10 times. For our model DFNet, we use the same hyperparameter settings as discussed in Section 4.4.3.1.

4.5 Results and Discussion

In this section, we discuss our experimental results to answer the aforementioned questions in Section 4.4. We use four benchmark datasets to compare these three variants against the state-of-the-art methods. We further discuss the effectiveness of our model DFNet through the node embeddings in a 2-D space of vertices from two datasets.

4.5.1 Comparison with Standard Data Splitting

In this section, we evaluate the performance of DFNet variants on semi-supervised node classification tasks with standard data splits to answer the question **Q1**. Table 4.3 summarizes the results of classification in terms of accuracy. The results of the baseline methods are taken from the previous works [122, 146, 247, 278]. Our models DFNet

and DFNet-ATT outperform all the baseline methods over four datasets. Particularly, we can see that: (1) Compared with polynomial filters, DFNet improves upon GCN (which performs best among the models using polynomial filters) by a margin of 3.7%, 3.9%, 5.3% and 2.3% on the datasets Cora, Citeseer, Pubmed and NELL, respectively. (2) Compared with rational polynomial filters, DFNet improves upon CayleyNet and $ARMA_1$ by 3.3% and 1.8% on the Cora dataset, respectively. For the other datasets, CayleyNet does not have results available in [137]. (3) DFNet-ATT further improves the results of DFNet due to the addition of a self-attention layer. (4) Compared with GAT (Chebyshev filters with self-attention), DF-ATT also improves the results and achieves 0.4%, 0.6% and 3.3% higher accuracy on the datasets Cora, Citeseer and Pubmed, respectively.

Additionally, we compare DFNet (our feedback-looped filters + DenseBlock) with GCN + DenseBlock and GAT + DenseBlock. The results are also presented in Table 4.3. We can see that our feedback-looped filters perform best, no matter whether or not the dense architecture is used.

Model	Cora	Citeseer	Pubmed	NELL
SemiEmb [263]	59.0	59.6	71.1	26.7
LP [298]	68.0	45.3	63.0	26.5
DeepWalk [196]	67.2	43.2	65.3	58.1
ICA [158]	75.1	69.1	73.9	23.1
Planetoid* [278]	64.7	75.7	77.2	61.9
Chebyshev [65]	81.2	69.8	74.4	-
GCN [122]	81.5	70.3	79.0	66.0
LNet [146]	79.5	66.2	78.3	-
AdaLNet [146]	80.4	68.7	78.1	-
CayleyNet [137]	81.9*	-	-	-
$ARMA_1$ [20]	83.4	72.5	78.9	-
GAT [247]	83.0	72.5	79.0	-
GCN + DenseBlock	82.7 ± 0.5	71.3 ± 0.3	81.5 ± 0.5	66.4 ± 0.3
GAT + Dense Block	83.8 ± 0.3	73.1 ± 0.3	81.8 ± 0.3	-
DFNet (ours)	$\mathbf{85.2 \pm 0.5}$	$\mathbf{74.2 \pm 0.3}$	$\mathbf{84.3 \pm 0.4}$	$\mathbf{68.3 \pm 0.4}$
DFNet-ATT (ours)	$\mathbf{86.0 \pm 0.4}$	$\mathbf{74.7 \pm 0.4}$	$\mathbf{85.2 \pm 0.3}$	$\mathbf{68.8 \pm 0.3}$
DF-ATT (ours)	83.4 ± 0.5	73.1 ± 0.4	$\mathbf{82.3 \pm 0.3}$	$\mathbf{67.6 \pm 0.3}$

Table 4.3: Accuracy (%) averaged over 10 runs (* was obtained using a different data splitting in [137])

.

4.5.2 Comparison with Random Data Splittings

In this section, we evaluate the performance of DFNets on semi-supervised node classification tasks with random data splits to answer the question **Q1**. Tables 4.4-4.6 present the experimental results. Table 4.4 shows that DFNet performs significantly better than all the other models over the Cora dataset, including LNet and AdaLNet proposed in [146]. Similarly, Table 4.5 shows that DFNet performs significantly better than all the other models over the Citeseer dataset. For the Pubmed dataset, as shown

in Table 4.6, DFNet performs significantly better than almost all the other models, except for only one case in which DFNet performs slightly worse than AdaLNet using the splitting 0.03%. These results demonstrate the robustness of our model DFNet.

Training Split	Chebyshev	GCN	GAT	LNet	AdaLNet	DFNet
5.2% (standard)	78.0 ± 1.2	80.5 ± 0.8	82.6 ± 0.7	79.5 ± 1.8	80.4 ± 1.1	**85.2 ± 0.5**
3%	62.1 ± 6.7	74.0 ± 2.8	56.8 ± 7.9	76.3 ± 2.3	77.7 ± 2.4	**80.5 ± 0.4**
1%	44.2 ± 5.6	61.0 ± 7.2	48.6 ± 8.0	66.1 ± 8.2	67.5 ± 8.7	**69.5 ± 2.3**
0.5%	33.9 ± 5.0	52.9 ± 7.4	41.4 ± 6.9	58.1 ± 8.2	60.8 ± 9.0	**61.3 ± 4.3**

Table 4.4: Accuracy (%) averaged over 10 runs on the Cora dataset.

Training Split	Chebyshev	GCN	GAT	LNet	AdaLNet	DFNet
3.6% (standard)	70.1 ± 0.8	68.1 ± 1.3	72.2 ± 0.9	66.2 ± 1.9	68.7 ± 1.0	**74.2 ± 0.3**
1%	59.4 ± 5.4	58.3 ± 4.0	46.5 ± 9.3	61.3 ± 3.9	63.3 ± 1.8	**67.4 ± 2.3**
0.5%	45.3 ± 6.6	47.7 ± 4.4	38.2 ± 7.1	53.2 ± 4.0	53.8 ± 4.7	**55.1 ± 3.2**
0.3%	39.3 ± 4.9	39.2 ± 6.3	30.9 ± 6.9	44.4 ± 4.5	46.7 ± 5.6	**48.3 ± 3.5**

Table 4.5: Accuracy (%) averaged over 10 runs on the Citeseer dataset.

Training Split	Chebyshev	GCN	GAT	LNet	AdaLNet	DFNet
0.3% (standard)	69.8 ± 1.1	77.8 ± 0.7	76.7 ± 0.5	78.3 ± 0.3	78.1 ± 0.4	**84.3 ± 0.4**
0.1%	55.2 ± 6.8	73.0 ± 5.5	59.6 ± 9.5	73.4 ± 5.1	72.8 ± 4.6	**75.2 ± 3.6**
0.05%	48.2 ± 7.4	64.6 ± 7.5	50.4 ± 9.7	68.8 ± 5.6	66.0 ± 4.5	**67.2 ± 7.3**
0.03%	45.3 ± 4.5	57.9 ± 8.1	50.9 ± 8.8	60.4 ± 8.6	**61.0 ± 8.7**	59.3 ± 6.6

Table 4.6: Accuracy (%) averaged over 10 runs on the Pubmed dataset.

4.5.3 Comparison under Different Polynomial Orders

In this section, we test how the polynomial orders p and q influence the performance of our model DFNet to answer the question **Q2**. We conduct experiments to evaluate DFNet on three citation network datasets using different polynomial orders $p = [1, 3, 5, 7, 9]$ and $q = [1, 3, 5, 7, 9]$. Figure 4.2 presents the experimental results. In our experiments, $p = 5$ and $q = 3$ turn out to be the best parameters for DFNet over these datasets. In other words, this means that feedback-looped filters are more stable on $p = 5$ and $q = 3$ than other values of p and q. This is because, when $p = 5$ and $q = 3$, Eq. 4.6 can obtain better convergence for finding optimal coefficients than in the other cases. Furthermore, we observe that: (1) Setting p to be too low or too high can both lead to poor performance, as shown in Figure 4.2.(a), and (2) when q is larger than p, the accuracy decreases rapidly as shown in Figure 4.2.(b). Thus, when choosing p and q, we require that $p > q$ holds.

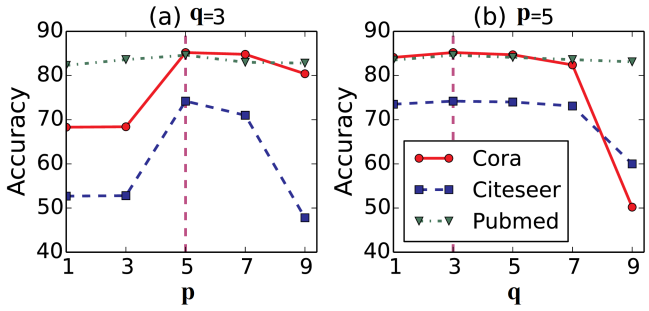
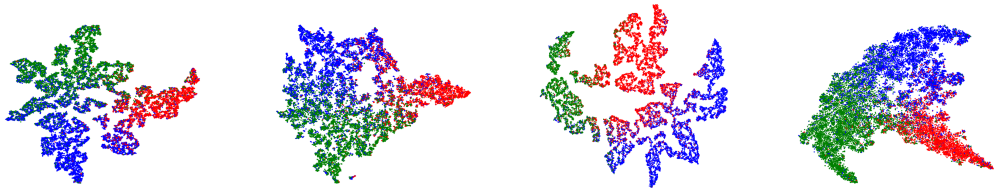


Figure 4.2: Accuracy (%) of DFNet under different polynomial orders p and q.



(a) GCN (b) GAT (c) Chebyshev (d) DFNet (ours)

Figure 4.3: The t-SNE visualization of the 2-D node embedding space for the Pubmed dataset.

(a) GCN (b) GAT (c) Chebyshev (d) DFNet (ours)

Figure 4.4: The t-SNE visualization of the 2-D node embedding space for the Cora dataset.

4.5.4 Node Embeddings Analysis

In this section, we evaluate the clustering quality of DFNets to answer the question **Q3**. We analyze the node embeddings by DFNets over two datasets: Cora and Pubmed in a 2-D space. Figures 4.3 and 4.4 display the visualization of the learned 2-D embeddings of GCN, GAT, Chebyshev, and DFNet (ours) on Pubmed and Cora citation networks by applying t-SNE [162] respectively. Colors denote different classes in these datasets. It reveals the clustering quality of theses models. These figures clearly show that our model DFNet has better separated 3 and 7 clusters respectively in the embedding spaces of Pubmed and Cora datasets. This is because features extracted by DFNet yield better node representations than GCN, GAT, and Chebyshev models.

4.5.5 Ablation Analysis of Scaled-Normalization and Cut-off Frequency

In this section, we evaluate the effectiveness of scaled-normalisation and cut-off frequency techniques to learn node representations to answer the question **Q4**. We

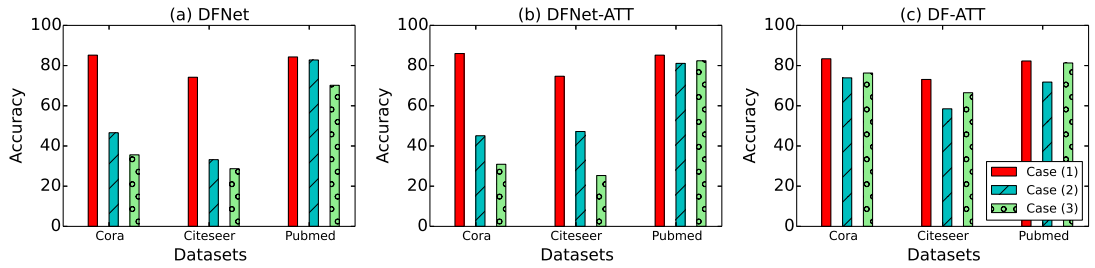


Figure 4.5: Accuracy (%) of our models in three cases: (1) using both scaled-normalization and cut-off frequency, (2) using only cut-off frequency, and (3) using only scaled-normalization.

compare our methods that implement these techniques with the variants of our methods that only implement one of these techniques. The results are presented in Figure 4.5. We can see that, the models using these two techniques outperform the models that only use one of these techniques over all citation network datasets. Particularly, the improvement is significant on the Cora and Citeseer datasets.

4.5.6 Ablation Analysis of the Impact of Feedback-looped Filters

In this section, we analyze the impact of feedback-looped filters to answer the question **Q4**. Specifically, we compare our feedback-looped filters, i.e., the newly proposed spectral filters in this paper, against other spectral filters such as Chebyshev filters and Cayley filters. To conduct this ablation study, we remove the dense connections from our model DFNet. The experimental results are presented in table 4.7. It shows that feedback-looped filters improve localization upon Chebyshev filters by a margin of 1.4%, 1.7% and 7.3% on the datasets Cora, Citeseer and Pubmed, respectively. It also improves upon Cayley filters by a margin of 0.7% on the Cora dataset.

Model	Cora	Citeseer	Pubmed
Chebyshev filters [65]	81.2	69.8	74.4
Cayley filters [137]	81.9	-	-
Feedback-looped filters (ours)	**82.6 $\pm$ 0.3**	**71.5 $\pm$ 0.4**	**81.7 $\pm$ 0.6**

Table 4.7: Accuracy (%) averaged over 10 runs.

4.6 Summary

In this chapter we have proposed a novel spectral graph neural network (GNN) model on graph structured data, namely *Distributed Feedback-Looped Networks* (DFNets)*. This model is incorporated with a robust class of spectral graph filters, called *feedback-looped filters*, to provide better localization on vertices, while still attaining fast convergence

*DFNets implementation can be found at: https://github.com/wokas36/DFNets

and linear memory requirements. Theoretically, feedback-looped filters can guarantee convergence w.r.t. a specified error bound, and be applied universally to any graph without knowing its structure. Furthermore, the propagation rule of this model can diversify features to produce strong gradient flows. We have evaluated our model using two benchmark tasks: semi-supervised document classification on citation networks and semi-supervised entity classification on a knowledge graph. The experimental results show that our model considerably outperforms the state-of-the-art methods in both benchmark tasks over all datasets.

Chapter 5

Dynamic PageRank for Diffusion Graph Neural Networks

5.1 Overview

In the past few years, several studies have explored the connection between graph diffusion and GNNs [125, 124, 49, 291, 38, 72]. Despite their successes, existing work still has several limitations: (1) *homophily vs heterophily:* Most GNNs [275] assumed that locally connected vertices on a graph share the same class labels (*homophily*), thereby relying on neighbourhood aggregation schemes. However, it is hard to generalize such neighbourhood aggregation schemes when locally connected vertices have different class labels (*heterophily*). Attempts have been made to extend nonnegativity diffusion weights for homophilic graphs [125] to general real numbers, assuming that the signs of diffusion weights can adapt to homophilic/heterophilic graph structure [49]. However, as discussed in [103], there is no clear evidence yet of negative weights in reflecting heterophily and this assumption may lead to generating ill-conditioned or oversimplified filters. (2) *depth vs width:* Some recent work argued that a shallow architecture of GNNs may hinder the model performance and thus proposed deep GNN architectures [44, 49]. Nonetheless, an increase of depth often leads to a linearly increasing number of model parameters since aggregation schemes used by most GNNs have learnable weight parameters in each layer which are not shared across layers [124]. This often makes training difficult, causing undesired issues such as overfitting and local optima.

By virtue of dynamic PageRank and its connection to dynamic systems [209], we propose two ways of developing new GNN architectures: (a) *Forward Euler solution*: a simple and fast approach that reflects spatial long-range dependencies between a current node with its neighbours (i.e., k-hop neighbourhood), and (b) *Invariable feature solution*: a flexible approach that generalizes both personalised PageRank and heat kernel. Both solutions are capable of lifting the dynamics of PageRank to adapt to different local structures underlying a graph. Further, we incorporate a learnable transition matrix to improve the discriminative power of PageRank diffusion. These designs together considerably enhance the generalization ability of our GNN architecture on homophilic and heterophilic graph structured data. Figure 5.1 provides

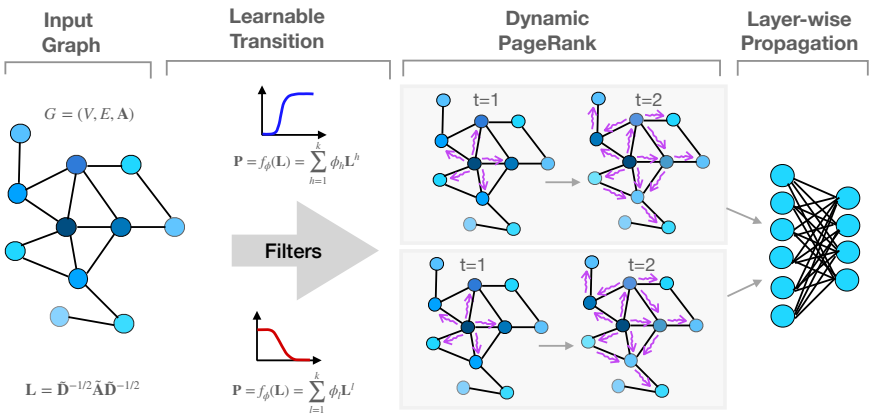


Figure 5.1: GNN Architecture of Dynamic PageRank Networks.

a high-level overview for the main components of this GNN architecture.

In this chapter, we present a GNN architecture, which is grounded on two observations: (1) Inspired by the modelling power of dynamic systems, can we dynamize PageRank for GNNs so as to improve its ability to capture the rich and varying graph structure, e.g., homophily/heterophily? (2) Instead of making a trade-off between the depth and width of a GNN architecture, can we go deeper in each layer in a shallow architecture to gain advantages from both sides?

The main contributions of this chapter are as follows:

- We shed light on how dynamic PageRank can be leveraged to design GNN models that are flexible and powerful for capturing rich and varying graph structures, including both homophilic and heterophilic structures. To this end, two solutions are proposed.
- We design a learning technique for PageRank transition, which is able to learn polynomial filter coefficients efficiently via a quadratic convex constrained optimization.
- Although dynamic PageRank generally does not converge, we theoretically prove that our GNN models are designed to achieve the guaranteed convergence due to the design of our dynamic PageRank diffusion schemes.
- We show that our shallow GNN architectures with *deeper single layers* represent a promising direction for alleviating some known GNN issues such as oversmoothing.

The rest of this chapter is organized as follows. In Section 5.2, we present a dynamic PageRank based GNN architecture with forward euler and invariable feature solutions. We also elaborate a new learnable PageRank transition matrix, a new GNN with deeper single layers and model training procedure. In Section 5.3, we analyze the convergence and complexity of the proposed diffusion GNN model. Section 5.4, we discuss the experimental setup. In Section 5.5, we compare the performance

of our proposed diffusion GNN method against the baseline methods. Section 5.6 summarises the chapter.

5.2 Diffusion GNNs with Dynamic PageRank

In this section, we introduce *Dynamic PageRank Network* (DPRN), a new class of graph neural networks based on dynamic PageRank and spectral filtering techniques on graphs.

5.2.1 Graph Diffusion

Standard PageRank [125] is known as the classical graph diffusion, defined as

$$\sum_{i=1}^{\infty} \theta_i \mathbf{P}^i \tag{5.1}$$

where θ are diffusion coefficients and $\mathbf{P} \in \mathbb{R}^{n \times n}$ is a column-stochastic transition matrix. As discussed in [124], a personalised PageRank vector is defined as $y_{ppr} = (1-\alpha)(\mathbf{I} - \alpha\mathbf{P})^{-1}x$ with $\theta_i^{ppr} = (1-\alpha)\alpha^i$ [189], where $\alpha \in (0,1)$ and $x \in \mathbb{R}^z$ refers to a teleportation vector. A heat kernel vector is defined as $y_{hk} = exp\{-t(\mathbf{I} - \mathbf{P})\}x$ with $\theta_i^{hk} = exp\{-t\}\frac{t^i}{i!}$, where $t > 0$ [128].

5.2.2 Dynamic PageRank on Graphs

Dynamic PageRank (DPR) was originally designed for graphs with time-dependent teleportation vectors [209]. Using the power method for the PageRank Markov chain [135], the standard PageRank can be represented as an iterative scheme where iterations correspond to time points such that

$$y(t+1) = (1-\alpha)x + \alpha\mathbf{P}y(t). \tag{5.2}$$

Here, $y(t)$ refers to a PageRank vector at a current time point t and $y(t+1)$ refers to a PageRank vector at next iteration. DPR incorporates a time-dependent teleportation vector $x(t)$ into PageRank by treating it as a function of time t. This leads to the continuous time differential equation:

$$\frac{\partial y(t)}{\partial t} = (1-\alpha)x(t) - (\mathbf{I} - \alpha\mathbf{P})y(t), \tag{5.3}$$

where $\mathbf{I} \in \mathbb{R}^{n \times n}$ is an identity matrix. A dynamic PageRank vector $y(t)$ is the solution of Eq. 5.3 [209]:

$$\begin{aligned} y(t) = {} & exp\big\{-(\mathbf{I} - \alpha\mathbf{P})t\big\}y(0) \\ & + (1-\alpha)\int_0^t exp\big\{-(\mathbf{I} - \alpha\mathbf{P})(t-\tau)\big\}x(\tau)d\tau. \end{aligned} \tag{5.4}$$

In this work, we consider the teleportation vector as a graph signal associated with each node, which may change depending on neighbours, and t refers to the range of spatial dependencies.

5.2.3 Dynamic PageRank with Forward Euler Solution

A simple and fast method for discretizing the general solution of dynamic PageRank in Eq. 5.4 is to use the forward Euler method based on the first order Taylor approximation [209]:

$$\frac{\partial y(t)}{\partial t} \approx \frac{y(t+s) - y(t)}{s}, \tag{5.5}$$

where $s > 0$ refers to a small time step. To incorporate node features into representations, we consider $y(t) = x(t)$ for all t and $x(0) = x$ is an initial graph signal. We have the following when $s = 1$:

$$y(t) = (1 - \alpha)y(t-1) + \alpha \mathbf{P} y(t-1). \tag{5.6}$$

Then, we treat Eq. 5.6 as a simple iterative scheme by mapping the time step t into a t-hop localization, which captures node features at different distances. This builds connections for dynamic PageRank with the message passing neighbourhood aggregation. Thus, we have the following representation matrix $\mathbf{Y}^{(t)}$ that captures the dynamics of graph signals in terms of the underlying graph structure, where $\mathbf{Y}^{(0)} = \mathbf{X}$ and $\alpha \in [0, 1]$ is fixed or learnable:

$$\mathbf{Y}^{(t)} = \Big((1 - \alpha)\mathbf{I} + \alpha \mathbf{P}\Big)\mathbf{Y}^{(t-1)}. \tag{5.7}$$

The scheme defined by Eq. 5.7 has a close connection with the neighbourhood feature aggregation in message-passing GNNs. Diffusion occurs from each node to its neighbours along edges, following the transition probabilities of $\mathbf{P}$. This means that each node receives diffusion from all its neighbours. Thus, $\mathbf{Y}^{(t)}$ combines the node feature of each node from the previous iteration with the aggregated node feature based on the topological structure (captured via $\mathbf{P}$) in the neighbourhood, where α controls the degree of structural bias injected from $\mathbf{P}$. Note that due to the dynamics of $x(t)$ and the condition on $y(t) = x(t)$, the scheme has departed from the standard PageRank and its variants.

5.2.4 Dynamic PageRank with Invariable Feature Solution

Dynamic PageRank generalizes both personalised PageRank and heat kernel [7]. This can be easily seen when we reduce the general solution of dynamic PageRank in Eq. 5.4 to the following equation where $x(t) = x$ is constant w.r.t. t,

$$y(t) = y_{ppr} + exp\Big\{ -t(\mathbf{I} - \alpha \mathbf{P})\Big\}(y(0) - y_{ppr}). \tag{5.8}$$

Here, $y(t)$ converges to y_{ppr} when $t \to \infty$, while $y(t)$ equals to y_{hk} when $\alpha = 1$. Accordingly, we have the following discrete version of Eq. 5.8 which defines a dynamic PageRank matrix $\mathbf{Y}^{(t)}$ with $\mathbf{Y}_{ppr} = (1-\alpha)(\mathbf{I} - \alpha\mathbf{P})^{-1}\mathbf{X}$ and $\mathbf{Y}^{(0)} = \mathbf{X}$:

$$\mathbf{Y}^{(t)} = \mathbf{Y}_{ppr} + exp\{-t(\mathbf{I} - \alpha\mathbf{P})\}(\mathbf{Y}^{(0)} - \mathbf{Y}_{ppr}). \tag{5.9}$$

To avoid the gradient vanishing or exploding during training, we apply the row-wise normalization to normalize $(1-\alpha)(\mathbf{I} - \alpha\mathbf{P})^{-1}$ and $exp\{-t(\mathbf{I} - \alpha\mathbf{P})\}$ in the implementation.

In the scheme defined by Eq. 5.9, $(\mathbf{Y}^{(0)} - \mathbf{Y}_{ppr})$ indicates how a node is different from its neighbours w.r.t. node features. When the difference is small (often occurring in homophilic graphs), dynamic PageRank behaves similarly to personalised PageRank which assigns random walks with shorter paths higher coefficients and decays at a fixed rate for long paths. This leverages the homophily in graphs. If the difference is large (often occurring in heterophilic graphs), non-linearity is added via the heat kernel $exp\{-t(\mathbf{I} - \alpha\mathbf{P})\}$ to capture longer range dependencies for heterophilic graphs.

5.2.5 Learnable PageRank Transition

A transition matrix in standard PageRank encodes landing probabilities of random walks within 1-hop neighbors of nodes. This has several limitations. First, restricting landing probabilities to 1-hop neighbors loses the flexibility of modelling varying landing probabilities of random walks within different ranges of neighbours. Second, landing probabilities are pre-determined and fixed, thereby limiting the ability to capture the structural properties of different graphs. To overcome these limitations, we define a transition matrix $\mathbf{P}$ as a learnable weighted linear combination of transition probabilities of different lengths. Formally, this is defined as a polynomial filter with learnable coefficients ϕ_i for $i = 1, \ldots, k$:

$$\mathbf{P} = f_\phi(\mathbf{L}) = \sum_{i=1}^{k} \phi_i \mathbf{L}^i, \tag{5.10}$$

where $\mathbf{L}$ is a Laplacian matrix.

For simplicity of notations, we use $\mathcal{Z}_{FE}$ or $\mathcal{Z}_{IF}$ to refer to the set of dynamic PageRank diffusion schemes defined by Eq. 5.7 and Eq. 5.9, respectively, in which each $\mathbf{P}$ is substituted by a learnable polynomial filter defined by Eq. 5.10.

5.2.6 A New GNN with Deeper Single Layers

Our DPRN model supports a flexible multi-layer architecture. DPRN can stack multiple layers of dynamic PageRank diffusion. Further, each layer of DPRN contains dynamic PageRank diffusion that dynamically adapts diffusion coefficients to the underlying structure of graphs.

Deeper single layers. Within each single layer in our GNN architecture, we incorporate multiple shallow layers, each corresponding to a dynamic PageRank diffusion

scheme as in Eq. 5.7 or Eq. 5.9. Our "deeper single layer" significantly reduces model parameters, in comparison with models with deep layers. Formally, we define the layer-wise propagation rule as:

$$\mathbf{Z}^{(l+1)} = \sigma\Big(\sum_{i=1}^{q}\Big(\mathbf{Z}_i^{(l)}W_i^{(l)} + b_i^{(l)}\Big)\Big), \tag{5.11}$$

where $l \geq 1$, σ refers to a non-linear activation function such as CELU [270], $W_i^{(t)} \in \mathbb{R}^{z\times r}$ is a trainable weight matrix, and $b_i^{(t)} \in \mathbb{R}^r$ is a trainable bias vector at the t^{th} layer. Each layer incorporates a number of dynamic PageRank diffusion schemes $\{\mathbf{Z}_1, \ldots, \mathbf{Z}_q\}$ with $q \geq 1$ and each $\mathbf{Z}_i$ can be flexibly chosen from $\mathcal{Z}_{FE}$ or $\mathcal{Z}_{IF}$. We initialize the weights and biases by drawing their values from a uniform distribution $\mathcal{U}(a, b)$ with a and b as the lower and upper bounds of the uniform distribution.

Discussion The design here brings in two important advantages over existing work: (1) *Weight parameter sharing* - By choosing $\{\mathbf{Z}_1, \ldots, \mathbf{Z}_q\}$, the model can support the sharing of weight parameters in dynamic PageRank diffusion schemes across different layers. This enables fast training. (2) *Varying graph structures* - different types of dynamic PageRank diffusion schemes correspond to different types of filters: high-pass, low-pass, and band-pass filters. They can be combined to learn varying graph structures. This improves model generalization across different graphs.

5.2.7 Model Training

We train the coefficients of polynomial filters through optimizing an objective function designed to achieve desired frequency responses. The other model parameters are trained via the label information. We use the Nystrom approximation to obtain the original frequency response since it is efficient than the eigenvalue decomposition. Our main goal is to learn optimal filter coefficients that minimize the error between original and desired frequency responses.

5.2.7.1 Nystrom Approximation

We apply the Nystrom approximation method [98] to compute the eigenvalues of a graph efficiently, which reduces the time complexity from $O(n^3)$ to $O(d^2n)$, where $d \ll n$. For a real symmetric matrix $\mathbf{L}$ and a basis matrix $\mathbf{Q} \in \mathbb{R}^{n\times l}$ with random initialization and orthonormal columns as input, we compute a nearly optimal rank-d approximation $\mathbf{U}_F\Lambda_F\mathbf{U}_F^T$ of $\mathbf{L}$ where $\mathbf{U}_F \in \mathbb{R}^{n\times d}$ and $\Lambda_F \in \mathbb{R}^{d\times d}$, as shown in Algorithm 1.

5.2.7.2 Training Coefficients

For a polynomial filter f_ϕ with coefficients $\phi \in \mathbb{R}^k$, the graph frequency response is defined as,

$$f_\phi(\lambda_j) = \sum_{i=1}^{k}\phi_i\lambda_j^i. \tag{5.12}$$

Algorithm 1: Nystrom approximation method

$\mathbf{R}_1 \leftarrow \mathbf{LQ}$ and $\mathbf{R}_2 \leftarrow \mathbf{Q}^T\mathbf{R}_1$
$[\mathbf{B}^T, \mathbf{B}] \leftarrow$ CholeskyFactorization($\mathbf{R}_2$)
$\mathbf{F} \leftarrow$ TriangularSolve($\mathbf{R}_1\mathbf{B}^{-1}$)
$[\mathbf{U}_F, \Sigma, \mathbf{V}_F] \leftarrow$ SVD($\mathbf{F}$) and $\Lambda_F \leftarrow \Sigma^2$

Given a desired frequency response: $\hat{h} : \{\lambda_i\}_{i=1}^d \rightarrow \mathbb{R}$, we compute $\phi = (\phi_1, \ldots, \phi_k)$ by solving a linear system:

$$\begin{bmatrix} \hat{h}(\lambda_1) \\ \hat{h}(\lambda_2) \\ \vdots \\ \hat{h}(\lambda_d) \end{bmatrix} = \begin{bmatrix} \lambda_1 & \lambda_1^2 & \ldots & \lambda_1^k \\ \lambda_2 & \lambda_2^2 & \ldots & \lambda_2^k \\ \vdots & \vdots & \vdots & \vdots \\ \lambda_d & \lambda_d^2 & \ldots & \lambda_d^k \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_k \end{bmatrix} \tag{5.13}$$

We consider $d > k$ in our work. Eq. (5.13) is overdetermined and has no exact solution. Thus, we learn ϕ by minimizing the error between the frequency response $f_\phi(\lambda_j)$ and the desired frequency response $\hat{h}(\lambda_j)$:

$$\hat{h}(\lambda_j) - \sum_{i=1}^{k} \phi_i \lambda_j^i \tag{5.14}$$

We recast the linear system in Eq. 5.13 as a convex constrained linear least-squares problem for learning the coefficients ϕ:

$$\textbf{minimize}_\phi \ ||\hat{h} - \mathbf{M}\phi||_2^2 \tag{5.15}$$

$$\textbf{subject to} \ \ ||\phi||_2 \leq \epsilon,$$

where $\epsilon \in [1, \infty)$ and $\mathbf{M} \in \mathbb{R}^{d \times k}$ is a full-rank Vandermonde matrix with entries $M_{j,i} = \lambda_j^i$. Empirically, we observe that a lower value of ϵ leads to efficient convergence and better accuracy. Thus, we set $\epsilon = 1$ in our experiments.

We apply a cut-off threshold $\lambda_{cut} = \lambda_{min} + \beta(\lambda_{max} - \lambda_{min})$ where $\beta \in [0, 1]$, and λ_{max} and λ_{min} refer to the largest and smallest eigenvalues of $\mathbf{L}$, respectively. For ideal high-pass filters, $\hat{h}(\lambda_i) = 1$ if $\lambda_i > \lambda_{cut}$; 0 otherwise. Conversely, for ideal low-pass filters, $\hat{h}(\lambda_i) = 1$ if $\lambda_i \leq \lambda_{cut}$; 0 otherwise.

5.2.8 Connection to Existing Methods

In the following, we discuss how our diffusion schemes generalize the existing GNNs in the literature such as GCN [122], GDC [125], APPNP [124], GPRGNN [49] and ADC [291] as the special cases.

5.2.8.1 Connection to Spectral Convolution

The connection between spectral filters and graph diffusion has been explored in [125]. When truncating Eq. 5.1 to the top-k items, there is a direct correspondence between the top-k truncation of graph diffusion and a polynomial filter of degree k.

The following theorem shows that our forward Euler solution has a close connection with spectral convolution.

Theorem 5. *The scheme of the forward Euler solution (Eq. 5.7) is equivalent to the following spectral convolution layer with $\phi_0 = -1$ and $\psi_i = \alpha\phi_i$:*

$$\mathbf{Y}^{(t)} = \Big(\mathbf{I} + \sum_{i=0}^{k} \psi_i \mathbf{L}^i\Big)\mathbf{Y}^{(t-1)}. \tag{5.16}$$

Proof. We can reformulate Eq. 5.7 by applying Eq. 5.10 as follows,

$$\begin{aligned}\mathbf{Y}^{(t)} =& \Big((1-\alpha)\mathbf{L}^0 + \alpha\sum_{i=1}^{k}\phi_i\mathbf{L}^i\Big)\mathbf{Y}^{(t-1)} \\ =& \Big(\mathbf{I} + (-\alpha)\mathbf{L}^0 + \alpha\sum_{i=1}^{k}\phi_i\mathbf{L}^i\Big)\mathbf{Y}^{(t-1)} \\ =& \Big(\mathbf{I} + \alpha\sum_{i=0}^{k}\phi_i\mathbf{L}^i\Big)\mathbf{Y}^{(t-1)},\end{aligned} \tag{5.17}$$

where $\phi_0 = -1$. Let $\psi_i = \alpha\phi_i$. Then we can reinterpret the above equation as follows,

$$\mathbf{Y}^{(t)} = \Big(\mathbf{I} + \sum_{i=0}^{k}\psi_i\mathbf{L}^i\Big)\mathbf{Y}^{(t-1)}. \tag{5.18}$$

The proof is done. □

GCN [122] is a first-order approximation of the spectral convolution, which is a special case of our method. When $k = 1$, $\alpha = 1$ and $\phi_1 = 1$, the above equation is equivalent to the GCN layer.

5.2.8.2 Connection to Graph Diffusion

We discuss how our invariable feature solution relates to existing diffusion-based GNN models. Let θ_i^{ppr} and θ_i^{hk} be diffusion coefficients for personalised PageRank and heat kernel (defined in Section 5.2.1) and f_ϕ be a learnable polynomial filter (defined in Section 5.2.5). A dynamic PageRank vector defined in Eq. 5.6 can be written as an infinite series [7] below, where $\vartheta_i = \theta_i^{ppr}(1 - \sum_{j=0}^{i}\theta_j^{hk}) + \alpha\theta_i^{hk}$,

$$\sum_{i=1}^{\infty}\vartheta_i f_\phi^i(\mathbf{L}). \tag{5.19}$$

GDC: Klicpera et al. [125] introduced Graph Diffusion Convolution (GDC) based on generalized graph diffusion $\sum_{i=1}^{\infty}\theta_i\mathbf{P}^i$. From Eq. 5.19, we see that our work can be considered as graph diffusion with coefficients $\vartheta_i = \theta_i$ and a transition matrix $f_\phi(\mathbf{L}) = \mathbf{P}$ that is learnable via a polynomial filter. Further, as discussed in Section 5.2.4, our work generalizes both personalised PageRank and heat kernel.

APPNP: Klicpera et al. [124] introduced a personalised PageRank GNN method, which uses a pre-defined kernel with fixed diffusion coefficients at its convolutional layers. We can see from Eq. 5.19 that our work generalizes APPNP when $\vartheta_i = \theta_i^{ppr}$ and $f_\phi(\mathbf{L}) = \mathbf{P}$.

GPRGNN: Chien et al. [49] proposed a generalized PageRank GNN method that adaptively learns diffusion coefficients. They truncate the infinite sum of generalized graph diffusion $\sum_{i=1}^{\infty}\theta_i\mathbf{P}^i$ into a finite sum and consider both positive and negative coefficients. When $i = 1$ and $\vartheta_i = 1$, Eq. 5.19 is reduced to their generalized PageRank.

ADC: Zhao et al. [291] proposed Adaptive Diffusion Convolution (ADC), which extends GDC to adaptively learn the neighborhood size for feature aggregation. They replace GNNs' discrete feature propagation function with a continuous heat kernel and use the neighborhood radius as a continuous substitute for the hop number in models.

5.3 Theoretical Analysis

In this section, we first prove that a diffusion GNN with the forward Euler method can guarantee the convergence in Section 5.3.1. Note that, we omit the convergence analysis of a diffusion GNN with invariable feature method since its teleportation vector is constant w.r.t time, and then, we cannot view it as a diffusion PDE. Then we provide the complexity analysis of our diffusion GNNs in Section 5.3.2.

5.3.1 Convergence Analysis

Generally, dynamic PageRank does not guarantee the convergence [209]. However, we theoretically show that our DPRN Forward Euler solution is guaranteed to converge due to our design choices on PageRank and teleportation vectors as well as the way of integrating polynomial filters.

Let $\mathbf{M} = \Big((1-\alpha)\mathbf{I} + \alpha\mathbf{P}\Big)$ and $d = \mathbf{D}\mathbf{1}_n$, where $\mathbf{1}_n$ is a n-dimensional vector of ones. Since $\mathbf{M}$ is a real symmetric matrix, it has real eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. The following theorems hold for dynamic PageRank diffusion schemes in $\mathcal{Z}_{FE}$.

Theorem 6. *When $t \to \infty$, the scheme defined by Eq. 5.7 converges to a stationary distribution $\tilde{\pi} : V \to \mathbb{R}^n$ with $\tilde{\pi}(u) = \frac{d(u)}{\sum_{v\in V} d(v)}$. The convergence rate for the teleportation distribution of a node v to jump to another node u is bounded by*

$$||\mathbf{Y}^{(t)}(u) - \tilde{\pi}(u)|| \leq \sqrt{\frac{d(u)}{d(v)}}\lambda_2^t||x||, \tag{5.20}$$

where $\mathbf{Y}^{(t)}(u)$ *refers to the entry of* $\mathbf{Y}^{(t)}$ *for node u and x is the graph signal associated with node v.*

Proof. Let $\mathbf{P}$ be a transition matrix of a connected graph, where all eigenvalues of $\mathbf{P}$ lie between 1 and -1, and $\mathbf{M} = \Big((1-\alpha)\mathbf{I} + \alpha\mathbf{P}\Big)$. Since $\mathbf{M}$ is a real symmetric matrix, $\mathbf{M}$ has a set of real eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ and the corresponding set of orthonormal eigenvectors $\mu_1, \ldots, \mu_n$. Then, we transform the scheme defined by Eq. 5.7 to the form $\mathbf{Y}^{(t)} = \mathbf{M}^t\mathbf{Y}^{(0)}$. Due to $\mathbf{Y}^{(0)} = \mathbf{X}$ where $\mathbf{X}$ is a graph signal matrix, $\mathbf{Y}^{(t)} = \mathbf{M}^t\mathbf{Y}^{(0)}$ is guaranteed to converge when $t \to \infty$.

Below we show how to obtain the bound for the convergence rate. Let $\mathbf{M}^t = \sum_{i=0}^{n} \lambda_i^t \mu_i \mu_i^T$, where λ_i and μ_i refer to the eigenvalues and orthonormal eigenvectors of $\mathbf{M}$, respectively. Let $\omega_i = \Big\langle \mathbf{D}^{-1/2}x, \mu_i \Big\rangle$, $\mathbf{D}^{-1/2}x = \sum_{i=1}^{n} \omega_i \mu_i$, and $x = \sum_{i=1}^{n} \omega_i \mathbf{D}^{1/2}\mu_i$. We can obtain ω_1 as follows,

$$\omega_1 = \Big\langle \mathbf{D}^{-1/2}x, \mu_1 \Big\rangle = \Big(\mathbf{D}^{-1/2}x\Big)^T \frac{\sqrt{d}}{||\sqrt{d}||} = \frac{x^T\mathbf{D}^{-1/2}\sqrt{d}}{||\sqrt{d}||} = \frac{1}{||\sqrt{d}||} \tag{5.21}$$

Let $\mathbf{Y}^{(t)}(u) = e_u^T\mathbf{Y}^{(t)}$, where e_u^T is an elementary unit vector in the direction u. We can compute the u-th entry of $\mathbf{Y}^{(t)}(u)$ as follows,

$$e_u^T\mathbf{Y}^{(t)} = e_u^T\mathbf{M}^t x = e_u^T\mathbf{M}^t \sum_{i=1}^{n} \omega_i \mathbf{D}^{1/2}\mu_i \tag{5.22}$$

Now we apply the eigenvalue decomposition for M. Since μ_is are an orthonormal basis, we can reinterpret the above equation as follows,

$$\begin{aligned}
e_u^T\mathbf{Y}^{(t)} =& e_u^T \sum_{i=1}^{n} \lambda_i^t \omega_i \mathbf{D}^{1/2}\mu_i \\
=& e_u^T\Big(\omega_1\mathbf{D}^{1/2}\mu_1 + \sum_{i=2}^{n} \lambda_i^t \omega_i \mathbf{D}^{1/2}\mu_i\Big) \\
=& e_u^T\Big(\frac{\mathbf{D}^{1/2}\mu_1}{||\sqrt{d}||} + \sum_{i=2}^{n} \lambda_i^t \omega_i \mathbf{D}^{1/2}\mu_i\Big) \\
=& e_u^T\Big(\frac{1}{||\sqrt{d}||}\mathbf{D}^{1/2}\frac{\sqrt{d}}{||\sqrt{d}||} + \sum_{i=2}^{n} \lambda_i^t \omega_i \mathbf{D}^{1/2}\mu_i\Big) \\
=& e_u^T\Big(\frac{d}{||\sqrt{d}||_1} + \sum_{i=2}^{n} \lambda_i^t \omega_i \mathbf{D}^{1/2}\mu_i\Big) \\
=& e_u^T\tilde{\pi} + e_u^T \sum_{i=2}^{n} \lambda_i^t \omega_i \mathbf{D}^{1/2}\mu_i.
\end{aligned} \tag{5.23}$$

Since $\tilde{\pi}(u) = e_u^T\tilde{\pi}$, we can obtain following equation,

$$\mathbf{Y}^{(t)}(u) - \tilde{\pi}(u) = e_u^T \sum_{i=2}^{n} \lambda_i^t \omega_i \mathbf{D}^{1/2}\mu_i \tag{5.24}$$

If we begin the walk at node v, then we have $||\mathbf{D}^{-1/2}x|| = \frac{||x||}{\sqrt{d(v)}}$. Since μ_is are orthornormal and $\lambda_i \leq \lambda_2$ for $i \geq 2$, we can derive the following upper bound,

$$\begin{aligned} ||\mathbf{Y}^{(t)}(u) - \tilde{\pi}(u)|| =& ||e_u^T \mathbf{D}^{1/2} \sum_{i=2}^{n} \lambda_i^t \omega_i \mu_i|| \\ =& ||\sqrt{d(u)} \sum_{i=2}^{n} \lambda_i^t \omega_i \mu_i|| \\ \leq& \sqrt{d(u)} \lambda_2^t || \sum_{i=1}^{n} \omega_i \mu_i|| \\ \leq& \sqrt{\frac{d(u)}{d(v)}} \lambda_2^t ||x|| \end{aligned} \tag{5.25}$$

□

In our work, t corresponds to the number of hops in each single layer. Thus, when $t \to \infty$, the power of these eigenvalues will diminish and close to 0 because the absolute value of every eigenvalue is strictly less than 1. Accordingly, when $t \to \infty$, the scheme defined by Eq. 5.7 converges to a stable distribution $\tilde{\pi}$. This means that node representations become stable when $t \to \infty$. As the stationary distribution $\tilde{\pi}(u)$ is proportional to node degrees, by Eq. 5.20, nodes with higher degrees converge faster than nodes with lower degrees. The convergence rate depends on λ_2. That is, the convergence is slower when λ_2 is larger.

For dynamic PageRank diffusion schemes in $\mathcal{Z}_{FE}$ in which $\mathbf{P}$ is a learnable polynomial filter, the following theorem holds. This is because polynomial filters in our work are Finite Impulse Response (FIR) filters, which are known to be inherently stable [152].

Theorem 7. *When $t \to \infty$, dynamic PageRank diffusion schemes in $\mathcal{Z}_{FE}$ are guaranteed to converge.*

Proof. In our work, we replace a random walk transition matrix with a learnable polynomial filter as defined in Eq. 5.10, which is based on the normalized adjacency matrix $\mathbf{L}$. Such a polynomial filter is known to be inherently stable in the frequency domain [152]. After replacing with such a polynomial filter, the matrix $\mathbf{M}$ is still a real symmetric and has a set of real eigenvalues $\tilde{\lambda}_1 \geq \tilde{\lambda}_2 \geq \cdots \geq \tilde{\lambda}_n$ and orthonormal eigenvectors $\tilde{\mu}_1, \ldots, \tilde{\mu}_n$. Thus, we can prove that the dynamic PageRank diffusion schemes in $\mathcal{Z}_{FE}$ guarantee the convergence to a stationary distribution based on Theorem 6. □

5.3.2 Complexity Analysis

Both of our solutions are efficient because their time complexity is linear w.r.t. the number of edges in a graph. Concretely, the time complexity of our solutions is $O(z(nr + tm) + tmk)$ and its memory complexity is $O(tm)$, where n and m are the

numbers of vertices and edges in a graph, respectively, t refers to the number of layers, z and r are the dimensions of input and output feature vectors of deeper single layer, respectively, and k is the dimension of coefficient vector. In our work, t and k are very small. The Nystrom approximation has the time complexity $O(d^2n)$. As it is performed only once to obtain the eigenvalues of the $\mathbf{L}$, it does not affect the time complexity of layer-wise propagation in Eq. 5.11. In our experiments, $t = 2$ for the forward Euler, $t = 1$ for the invariable feature, and $k \leq 10$.

The model parameters (weights and biases) are shared over graphs in a single layer and the parameters of polynomial filters are shared across different layers. The total number of parameters is:

$$\Big(\sum_{i=1}^{q}(\#\text{model params} + \#\text{filter params})\Big) \times \text{depth}.$$

Note that #filter params $\ll$ #model params and the number of polynomial filter parameters is very small (less than 20). The depth of our model is set to 2 in our experiments.

5.4 Experimental Setup

To empirically verify the effectiveness of our work, we have evaluated DPRN models with different benchmark datasets to compare against the state-of-the-art methods. We use DPRN-FE and DPRN-IF to refer to our GNN models, each of which has two layers with diffusion schemes from $\mathcal{Z}_{FE}$ and $\mathcal{Z}_{IF}$, respectively. We conduct experiments to answer the following questions:

Q1. How well our models perform against the state-of-the-art baselines on semi-supervised node classification task?

Q2. How well our models perform on graphs with varying degrees of homophily on fully-supervised node classification task?

Q3. How robust our models are w.r.t. the model depth on semi-supervised node classification task?

Q4. How well can teleportation parameter α impact on the mode performance?

Q5. How well can learnable polynomial filtering and dynamic PageRank techniques affect the performance of our models?

5.4.1 Datasets

For semi-supervised node classification, we consider three benchmark datasets, which include three citation network datasets Cora, Citeseer, and Pubmed [223]. We also use two large graph datasets (ogbn-arxiv and ogbn-proteins) from Open Graph Benchmark (OGB) [108] to evaluate the node classification task.

For fully-supervised node classfication, we consider eleven benchmark datasets, including three citation graphs Cora, Citeseer, Pubmed [122, 49], two Amazon co-purchase graphs Photo and Computers [49], three webpage graphs from the WebKB

dataset Texas, Cornell and Wisconsin [192], an Actor co-occurrence graph [192], and two Wikipedia graphs Squirrel and Chameleon [192].

Table 5.1 presents some statistics for all the datasets used in our experiments.

Dataset	Homophily Ratio	#Nodes	#Edges	#Classes	#Features
Cora	0.83	2,708	5,429	7	1,433
Citeseer	0.71	3,327	4,732	6	3,703
Pubmed	0.79	19,717	44,338	3	500
Photo	0.45	7,650	1,19,081	8	745
Computers	0.27	13,752	2,45,861	10	767
Chameleon	0.25	2,277	36,101	5	2,325
Actor	0.24	7,600	33,544	5	931
Squirrel	0.22	5,201	217,073	5	2,089
Wisconsin	0.16	251	499	5	1,703
Cornell	0.11	183	295	5	1,703
Texas	0.06	183	309	5	1,703
ogbn-arxiv	-	169,343	1,166,243	40	128
ogbn-proteins	-	132,534	39,561,252	2	8

Table 5.1: Dataset statistics.

The homophily ratio $\eta \in [0, 1]$ measures the homophily level of a graph, which is computed following [192]:

$$\eta = \frac{1}{n} \sum_{u \in V} \frac{|\{v \in \mathcal{N}(u) | l(v) = l(u)\}|}{|\mathcal{N}(u)|}, \tag{5.26}$$

where $l(u)$ and $\mathcal{N}(u)$ refer to the label and the neighbors of a node u, respectively. A graph is strongly homophilic when $\eta \to 1$, and conversely, a graph is strongly heterophilic when $\eta \to 0$.

5.4.2 Baseline Methods

We compare our DPRN-IF and DPRN-FE models with the state-of-the-art methods in semi-supervised node classification (Section 5.4.2.1), fully-supervised node classification (Section 5.4.2.2), and node classification on OGB datasets (Section 5.4.2.3).

5.4.2.1 Semi-supervised Node Classification with Standard Data Splittings

In the experiments for semi-supervised node classification (Section 5.5.1), we compare performance against 13 baseline methods: graph convolutional networks (GCN) [122], graph attention networks (GAT) [247], fast learning with graph convolutional networks (FastGCN) [43], simple graph convolution networks (SGCN) [266], ARMA convolutional networks (ARMA) [20], higher-order graph convolutional with sparsified neighborhood networks (MixHop) [2], deep graph infomax (DGI) [248], simple spectral graph convolution networks (SSGC) [296], graph diffusion convolution (GDC) [125], approximate personalised propagation of neural predictions (APPNP) [124],

adaptive universal generalized pagerank graph neural network (GPRGNN) [49], explicit pairwise factorized graph neural network (EPFGNN) [259], graph neural diffusion (GRAND) [38], and node classification with pairwise Markov random fields (LCM) [256].

5.4.2.2 Fully-supervised Node Classification with Random Data Splittings

We consider two different experimental setups to evaluate DPRN for fully-supervised node classification:

(1) We use the experimental setup as used in [49, 192]. We compare performance against 12 baseline methods: graph convolutional networks (GCN) [122], graph attention networks (GAT) [247], approximate personalised propagation of neural predictions (APPNP) [124], jumping knowledge networks (JKNet) [274], geometric graph convolutional networks (Geom-GCN) [192], adaptive spectral filters with GAT (ASGAT-Cheb and ASGAT-ARMA) [143], non-local graph neural networks (NLMLP and NLGCN) [153], adaptive universal generalized pagerank graph neural network (GPRGNN) [49], graph spectral filters via bernstein approximation (BernNet) [103], and PDE-GCN [72].

(2) We use the experimental setup as used in [297]. We also compare the performance of our models against six baseline methods: GraphSAGE [99], MixHop [2], H_2GCN [297], GPRGNN [49], MLP+GCN [161], and GRAND [38]. The results are presented in Tables 5.8 and 5.9.

5.4.2.3 Node Classification with OGB Datasets

In the experiments for OGB node classification, we compare the performance of DPRN-IF and DPRN-FE against the baseline methods: GCN, GAT, GraphSAGE [108] and GRAND [38].

5.4.3 Hyperparameter Settings

The hyperparameters of DPRN-IF and DPRN-FE are selected on the classification accuracy of the validation set by applying the randomized search strategy.

Our experiments for semi-supervised node classification (Section 5.5.1) and model depth (Section 5.5.4) are conducted on the standard splits following [122, 108]. For all experiments on the standard splits, our models are trained 200 epochs using the Adam optimizer [120]. Tables 5.2 and 5.3 provide the detailed information for the hyperparameter settings on the standard splits for our models DPRN-IF and DPRN-FE, respectively.

Our experiments for fully-supervised node classification (Section 5.5.3) and ablation studies (Section 5.5.6) are conducted on the random splits following [49, 192]. Specifically, each dataset is randomly split into 60%, 20% and 20% for training, validation and testing, respectively. We evaluate our model over 10 random splits with 200 training epochs for Cora, Citeseer, and Pubmed datasets and 1000 training epochs for

Model	Weight Decay	Learning Rate	#Hidden Dim	Dropout	$[k_1, k_2]$	β_l	β_h	α	#Epochs
Cora	9e-3	0.009	16	0.9	[2, 5]	9e-1	9e-2	0.75	200
Citeseer	9e-3	0.05	16	0.9	[2, 5]	9e-1	1e-2	0.65	200
Pubmed	9e-3	0.009	16	0.9	[2, 6]	9e-1	9e-2	0.75	200
ogbn-arxiv	9e-3	0.009	64	0.9	[2, 5]	9e-1	9e-2	0.75	500
ogbn-proteins	9e-3	0.009	128	0.7	[2, 5]	9e-1	9e-2	0.65	500

Table 5.2: Hyperparameter settings for semi-supervised node classification with the standard splits for DPRN-IF.

Model	Weight Decay	Learning Rate	#Hidden Dim	Dropout	$[k_1, k_2]$	β_l	β_h	α	#Epochs
Cora	9e-3	0.009	16	0.9	[2, 5]	5e-1	6e-1	0.68	200
Citeseer	9e-3	0.009	16	0.7	[2, 5]	4e-1	1e-1	0.68	200
Pubmed	9e-3	0.009	64	0.9	[2, 5]	5e-1	6e-1	0.70	200
ogbn-arxiv	9e-3	0.009	64	0.9	[2, 5]	5e-1	6e-1	0.70	500
ogbn-proteins	9e-3	0.009	128	0.7	[2, 5]	5e-1	6e-1	0.75	500

Table 5.3: Hyperparameter settings for semi-supervised node classification with the standard splits for DPRN-FE.

the other datasets, and the Adam optimizer [120]. We take the average of the results and provide the mean accuracy with 95 % confidence interval as the evaluation metric in the same way as used in [49]. We also evaluate the random data splits as suggested in [297], where each dataset is split into 48%, 32%, and 20% training, validation, and testing, respectively. For both settings, we use the same hyperparameter settings as in Table 5.4 and Table 5.5 for our models DPRN-IF and DPRN-FE, respectively.

For our GNN models, DPRN-FE and DPRN-IF, each of them has two layers with diffusion schemes from $\mathcal{Z}_{EF}$ and $\mathcal{Z}_{IF}$, respectively. Further, we consider two diffusion schemes for each layer of these models. In Tables 5.2-5.3 and Tables 5.4-5.5, the parameters β_l and β_h refer to the cut-off thresholds for two diffusion schemes used in each layer in these models. More specifically, β_l corresponds to a low-pass diffusion scheme that has a polynomial filter of degree k_1. Similarly, β_h corresponds to a high-pass diffusion scheme that has a polynomial filter of degree k_2. It is also possible to have other types of diffusion schemes such as band-pass diffusion schemes.

5.5 Results and Discussion

In this section, we present our experimental results to answer the aforementioned questions in Section 5.4.

5.5.1 Comparison with Semi-supervised Node Classification

In this section, we evaluate the performance of DPRN-IF and DPRN-FE on semi-supervised node classification task to answer the question **Q1**. Table 5.6 summarizes the results for Cora, Citeseer, and Pubmed datasets.

DPRN-IF outperforms all the baselines on Cora and Citeseer and performs comparably to GRAND on Pubmed. Generally, DPRN-FE has worse performance than DPRN-IF, although it still outperforms the baselines on Cora. Particularly, we can see that DPRN-IF improves upon best result by a margin of 1.1% and 0.4% on the

Model	Weight Decay	Learning Rate	#Hidden Dim	Dropout	$[k_1, k_2]$	β_l	β_h	α	#Epochs
Cora	9e-3	0.009	64	0.9	[2, 5]	9e-1	9e-2	0.60	200
Citeseer	9e-3	0.09	64	0.9	[5, 6]	9e-1	9e-2	0.75	200
Pubmed	9e-3	0.009	64	0.9	[2, 5]	9e-1	9e-2	0.60	200
Photo	5e-4	0.009	16	0.9	[2, 6]	9e-1	9e-2	0.60	1000
Computers	5e-4	0.009	16	0.9	[2, 6]	9e-1	9e-2	0.60	1000
Actor	5e-4	0.09	16	0.9	[2, 5]	9e-1	9e-2	0.30	1000
Wisconsin	9e-3	0.009	16	0.9	[2, 6]	9e-1	9e-2	0.15	1000
Cornell	5e-5	0.009	512	0.9	[2, 5]	4e-1	9e-2	0.05	1000
Texas	5e-4	0.009	2048	0.9	[2, 5]	9e-1	9e-2	0.19	1000
Chameleon	9e-4	0.01	64	0.9	[9, 7]	9e-1	1e-2	0.90	1000
Squirrel	9e-4	0.01	16	0.9	[2, 5]	9e-1	1e-2	0.90	1000

Table 5.4: Hyperparameter settings for semi-supervised node classification with the random splits for DPRN-IF.

Model	Weight Decay	Learning Rate	#Hidden Dim	Dropout	$[k_1, k_2]$	β_l	β_h	α	#Epochs
Cora	9e-3	0.009	16	0.9	[2, 5]	5e-1	6e-1	0.60	200
Citeseer	9e-3	0.009	256	0.9	[2, 5]	5e-1	6e-1	0.65	200
Pubmed	9e-3	0.009	64	0.9	[2, 5]	5e-1	6e-1	0.70	200
Photo	1e-4	0.009	16	0.9	[2, 5]	5e-1	6e-1	0.30	1000
Computers	5e-4	0.009	16	0.9	[2, 5]	5e-1	6e-1	0.30	1000
Actor	5e-4	0.009	16	0.9	[2, 5]	9e-1	9e-2	0.2	1000
Wisconsin	1e-3	0.009	512	0.9	[2, 5]	9e-1	9e-2	0.070	1000
Cornell	1e-3	0.009	512	0.9	[2, 5]	9e-1	9e-2	0.072	1000
Texas	2e-3	0.009	1024	0.9	[2, 5]	9e-1	4e-2	0.064	1000
Chameleon	1e-7	0.001	1024	0.6	[2, 5]	9e-1	2e-1	0.99	1000
Squirrel	1e-7	0.001	1024	0.6	[2, 5]	9e-1	2e-1	0.99	1000

Table 5.5: Hyperparameter settings for semi-supervised node classification with the random splits for DPRN-FE.

datasets Cora and Citeseer, respectively. Similarly, we can see that DPRN-FE improves upon best result by a margin of 0.3% on Cora.

5.5.2 Comparison with Node Classification on OGB Datasets

In this section, we evaluate the performance of DPRN-IF and DPRN-FE on OGB node classification task to answer the question **Q1**. Table 5.7 shows the results for these large graph datasets. Specifically, DPRN-IF improves upon GAT by a margin of 0.06% on ogbn-arxiv. Moreover, we can see that DPRN-IF performs comparably to GAT on ogbn-proteins. For both datasets, DPRN-FE has worse performance than DPRN-IF.

5.5.3 Comparison with Fully-supervised Node Classification

In this section, we evaluate the performance of DPRN-IF and DPRN-FE on fully-supervised node classification task to answer the question **Q2**. Table 5.11 and Table 5.10 show the results for the first experimental setup of this setting (i.e., randomly splits datasets into 60%, 20% and 20% for training, validation and testing).

From Table 5.11 and 5.10, we can also see that DPRN-FE and DPRN-IF outperform the state-of-the-art baselines on most of homophilic and heterophilic graphs. DPRN-FE outperforms the baseline methods over almost all datasets except Citeseer, Actor, Chameleon and Squirrel. DPRN-IF has generally achieved comparable performance as

Model	Cora	Citeseer	Pubmed
GCN	81.5	70.3	79.0
GAT	83.0 ± 0.70	72.5 ± 0.70	79.0 ± 0.30
FastGCN	79.8 ± 0.30	68.8 ± 0.60	77.4 ± 0.30
ARMA	83.4 ± 0.60	72.5 ± 0.40	78.9 ± 0.30
MixHop	81.8 ± 0.62	71.4 ± 0.81	80.0 ± 1.10
DGI	82.3 ± 0.60	71.8 ± 0.70	76.8 ± 0.60
SSGC	83.5 ± 0.02	73.6 ± 0.09	80.2 ± 0.02
GDC	83.3 ± 0.20	72.2 ± 0.30	78.6 ± 0.40
APPNP	83.3 ± 0.56	71.4 ± 0.60	80.1 ± 0.24
GPRGNN	83.6 ± 0.47	71.5 ± 0.29	79.7 ± 0.27
EPFGNN	83.5	73.1	80.1
GRAND	82.9 ± 0.70	73.6 ± 0.30	**81.0 ± 0.40**
LCM	83.3 ± 0.70	72.2 ± 0.50	77.0 ± 1.90
DPRN-IF (ours)	**84.7 ± 0.70**	**74.0 ± 1.90**	80.8 ± 1.80
DPRN-FE (ours)	**83.9 ± 1.00**	73.5 ± 1.50	80.3 ± 1.98

Table 5.6: Classification accuracy (%) averaged over 10 runs for semi-supervised node classification.

Model	ogbn-arxiv	ogbn-proteins
GCN	71.74 ± 0.29	72.51 ± 0.35
GAT	73.65 ± 0.11	**78.63 ± 1.62**
GraphSAGE	71.49 ± 0.27	77.68 ± 0.20
GRAND	72.23 ± 0.20	-
DPRN-IF (ours)	**73.71 ± 0.95**	78.03 ± 1.96
DPRN-FE (ours)	72.37 ± 0.93	77.89 ± 1.87

Table 5.7: Classification accuracy (%) averaged over 10 runs for OGB node classification.

DPRN-FE. Particularly, for homophilc graphs in Table 5.10, we can see that DPRN-IF improves upon best result by a margin of 1.53% and 0.87% on the datasets Cora and Pubmed, respectively. Similarly, we can see that DPRN-FE improves upon best result by a margin of 1.59%, 0.04%, 0.50% and 0.12% on Cora, Pubmed, Photo and Computers, respectively. Specifically, for heterophilic graphs in Table 5.11, we can see that DPRN-IF improves upon the best result by a margin of 0.03%, 0.65%, 1.35% and 0.27% on Actor, Cornell, Texas and Chameleon, respectively. Similarly, we can see that DPRN-FE improves upon the best result by a margin of 1.99%, 1.31% and 0.53% on Wisconsin, Cornell and Texas, respectively.

Table 5.8 and Table 5.9 show the results for the second experimental setup (i.e., randomly splits datasets into 48%, 32% and 20% for training, validation and testing). For homophilic graphs in Table 5.8, DPRN-FE outperforms all the baselines on Cora and Citeseer and performs comparably to MLP+GCN on Pubmed. Most of the datasets, DPRN-IF has worse performance than DPRN-FE except Pubmed, although it still outperforms the baselines on Citeseer. Particularly, we can see that DPRN-IF improves upon best result by a margin of 0.66% on Citeseer. We can also see that DPRN-FE improves upon the best result by a margin of 0.05% and 1.92% on Cora and Citeseer, respectively. For heterophilic graphs in Table 5.9, DPRN-IF and DPRN-FE

Model	Cora	Citeseer	Pubmed
GraphSAGE	86.90 ± 1.04	76.04 ± 1.30	88.45 ± 0.50
MixHop	87.61 ± 0.85	76.26 ± 1.33	85.31 ± 0.61
H_2GCN	86.92 ± 1.37	77.07 ± 1.64	89.40 ± 0.34
GPRGNN	87.95 ± 1.18	77.13 ± 1.67	87.54 ± 0.38
MLP+GCN	87.01 ± 1.35	76.35 ± 1.85	**89.77 ± 0.39**
GRAND	87.36 ± 0.96	76.46 ± 1.77	89.02 ± 0.51
DPRN-IF (ours)	87.38 ± 0.56	**77.79 ± 0.86**	89.70 ± 1.27
DPRN-FE (ours)	**88.00 ± 0.37**	**79.05 ± 0.59**	88.71 ± 1.81

Table 5.8: Classification accuracy (%) averaged over 10 runs for fully-supervised node classification on homophilic datasets.

Model	Actor	Wisconsin	Cornell	Texas	Chameleon	Squirrel
GraphSAGE	34.23 ± 0.99	81.18 ± 5.56	75.95 ± 5.01	82.43 ± 6.14	58.73 ± 1.68	41.61 ± 0.74
MixHop	32.22 ± 2.34	75.88 ± 4.90	73.51 ± 6.34	77.84 ± 7.73	60.50 ± 2.53	43.80 ± 1.48
H_2GCN	35.86 ± 1.03	86.67 ± 4.69	82.16 ± 4.80	84.86 ± 6.77	57.11 ± 1.58	36.42 ± 1.89
GPRGNN	34.63 ± 1.22	82.94 ± 4.21	80.27 ± 8.11	78.38 ± 4.36	46.58 ± 1.71	31.61 ± 1.24
MLP+GCN	36.24 ± 1.09	86.43 ± 4.00	84.82 ± 4.87	83.60 ± 6.04	**68.04 ± 1.86**	**54.48 ± 1.11**
GRAND	35.62 ± 1.01	79.41 ± 3.64	82.16 ± 7.09	75.68 ± 7.25	54.67 ± 2.54	40.05 ± 1.50
DPRN-IF (ours)	**40.64 ± 0.54**	**86.88 ± 1.53**	**91.52 ± 0.89**	**90.98 ± 1.79**	65.51 ± 1.68	45.93 ± 1.91
DPRN-FE (ours)	**38.65 ± 0.43**	**86.91 ± 0.64**	**91.76 ± 1.70**	**90.39 ± 1.05**	63.31 ± 2.51	43.54 ± 2.21

Table 5.9: Classification accuracy (%) averaged over 10 runs for fully-supervised node classification on heterophilic datasets.

outperforms the baseline methods over on most of datasets except Chameleon and Squirrel. Specifically, DPRN-IF outperforms the best result by a margin of 4.40%, 0.21%, 6.70% and 6.12% on Actor, Wisconsin, Cornell and Texas, respectively. Similarly, DPRN-FE outperforms the best result by a margin of 2.41%, 0.24%, 6.94% and 5.53% on Actor, Wisconsin, Cornell and Texas, respectively.

	Model	Cora	Citeseer	Pubmed	Photo	Computers
Homophily	GCN	86.87 ± 0.26	79.28 ± 0.25	86.97 ± 0.12	88.26 ± 0.73	83.32 ± 0.33
	GAT	87.52 ± 0.24	**80.56 ± 0.31**	86.64 ± 0.11	90.94 ± 0.68	83.32 ± 0.39
	APPNP	88.10 ± 0.23	80.50 ± 0.26	89.15 ± 0.13	88.51 ± 0.31	85.32 ± 0.37
	JKNet	86.97 ± 0.27	77.69 ± 0.35	87.38 ± 0.13	87.70 ± 0.70	77.80 ± 0.97
	Geom-GCN	85.40 ± 0.26	76.42 ± 0.37	88.51 ± 0.08	-	-
	ASGAT-Cheb	87.50 ± 0.50	79.30 ± 0.60	89.90 ± 0.90	-	-
	ASGAT-ARMA	87.40 ± 1.10	79.20 ± 1.40	88.30 ± 1.00	-	-
	NLMLP	76.90 ± 1.80	73.40 ± 1.90	88.20 ± 0.50	-	-
	NLGCN	88.10 ± 1.00	75.20 ± 1.40	89.00 ± 0.50	-	-
	GPRGNN	88.65 ± 0.28	80.01 ± 0.28	89.18 ± 0.15	93.85 ± 0.28	86.85 ± 0.25
	BernNet	88.52 ± 0.95	80.09 ± 0.79	88.48 ± 0.41	93.63 ± 0.35	87.64 ± 0.44
	PDE-GCN	88.60	78.48	89.93	-	-
	DPRN-IF (ours)	**90.18 ± 0.36**	80.39 ± 0.73	**90.80 ± 1.96**	93.40 ± 0.41	86.11 ± 0.21
	DPRN-FE (ours)	**90.24 ± 0.42**	79.84 ± 0.48	**89.97 ± 1.53**	**94.35 ± 0.11**	**87.76 ± 1.77**

Table 5.10: Classification accuracy (%) averaged over 10 runs for fully-supervised node classification on homophilic datasets. The results of GCN, GAT, APPNP, JKNet, Geom-GCN and GPRGNN are taken from [49] and the others are from their original papers.

	Model	Actor	Wisconsin	Cornell	Texas	Chameleon	Squirrel
Heterophily	GCN	30.59 ± 0.23	-	66.72 ± 1.37	75.16 ± 0.96	60.96 ± 0.78	45.66 ± 0.39
	GAT	35.98 ± 0.23	-	76.00 ± 1.01	78.87 ± 0.86	63.90 ± 0.46	42.72 ± 0.33
	APPNP	38.86 ± 0.24	-	91.80 ± 0.63	91.18 ± 0.70	51.91 ± 0.56	34.77 ± 0.34
	JKNet	33.41 ± 0.25	-	66.73 ± 1.73	75.53 ± 1.16	62.92 ± 0.49	44.72 ± 0.48
	Geom-GCN	31.81 ± 0.24	-	55.59 ± 1.59	58.56 ± 1.77	61.06 ± 0.49	38.28 ± 0.27
	ASGAT-Cheb	-	86.30 ± 3.70	82.70 ± 8.30	85.10 ± 5.70	66.50 ± 2.80	**55.80 ± 3.20**
	ASGAT-ARMA	-	84.70 ± 4.40	83.20 ± 5.50	79.50 ± 7.70	65.80 ± 2.20	51.40 ± 3.20
	NLMLP	37.90 ± 1.30	87.30 ± 4.30	84.90 ± 5.70	85.40 ± 3.80	50.70 ± 2.20	33.70 ± 1.50
	GPRGNN	39.30 ± 0.27	-	91.36 ± 0.70	92.92 ± 0.61	67.48 ± 0.40	49.93 ± 0.53
	BernNet	41.79 ± 1.01	-	92.13 ± 1.64	93.12 ± 0.65	68.29 ± 1.58	51.35 ± 0.73
	PDE-GCN	-	91.76	89.73	93.24	66.01	-
	DPRN-IF (ours)	**41.82 ± 0.78**	88.37 ± 2.09	**92.78 ± 1.67**	**94.59 ± 0.85**	**68.56 ± 1.92**	50.69 ± 1.75
	DPRN-FE (ours)	40.03 ± 0.54	**93.75 ± 0.79**	**93.44 ± 1.26**	**93.77 ± 1.42**	65.00 ± 2.31	47.31 ± 2.01

Table 5.11: Classification accuracy (%) averaged over 10 runs for fully-supervised node classification on heterophilic datasets. The results of GCN, GAT, APPNP, JKNet, Geom-GCN and GPRGNN are taken from [49] and the others are from their original papers.

5.5.4 Comparison with Model Depth

In this section, we benchmark the performance of our models by increasing the number of convolutional layers to answer the question **Q3**. Table 5.12 summarizes the results of semi-supervised node classification with the standard splits on the datasets Cora, Citeseer and Pubmed, and the same hyperparameter setting as in Section 5.5.1. When increasing the model depth, our models achieve the best performance with a two-layer architecture, while the other models either suffer from the oversmoothing issues (e.g., GCN and SGCN) or achieve their best performance with a much deeper architecture (e.g., JKNet, SSGC, and PDE-GCN). This is because our models incorporate dynamics of PageRank and learnable polynomial filters, which associate only a small number of

parameters in comparison with weights and biases in each layer and such parameters are also shared across layers.

Dataset	#Layers	GCN	JKNet	SGCN	SSGC	PDE-GCN	DPRN-IF (ours)	DPRN-FE (ours)
Cora	2	81.1	-	80.8	76.2	82.0	**84.7**	83.9
	4	80.4	80.2	81.5	79.8	83.6	83.5	83.3
	8	69.5	80.7	80.7	82.2	84.0	83.0	82.5
	16	64.9	80.2	79.0	83.5	84.2	82.9	82.1
	32	60.3	81.1	75.9	82.6	84.3	82.5	81.5
	64	28.7	71.5	66.8	82.0	84.3	82.0	81.1
Citeseer	2	70.8	-	71.9	70.7	74.6	**74.0**	73.5
	4	67.6	68.7	72.6	72.6	75.0	73.5	72.8
	8	30.2	67.7	73.1	72.7	75.2	73.1	72.4
	16	18.3	69.8	72.2	73.6	75.5	72.8	72.0
	32	25.0	68.2	70.6	74.0	**75.6**	72.3	71.5
	64	20.0	63.4	69.2	73.4	75.5	72.0	71.1
Pubmed	2	79.0	-	79.2	78.5	79.3	**80.8**	80.3
	4	76.5	78.0	79.7	79.2	80.6	80.2	79.3
	8	61.2	78.1	78.4	79.7	80.1	79.7	78.9
	16	40.9	72.6	76.4	80.2	80.4	79.1	78.2
	32	22.4	72.4	71.6	79.1	80.2	78.5	77.8
	64	35.3	74.5	68.6	78.1	80.3	77.7	77.1

Table 5.12: Classification accuracy (%) averaged over 10 runs for semi-supervised node classification w.r.t model depth.

5.5.5 Comparison with Varying Teleportation Parameter

In this section, we analyze the impact of teleportation parameter α on the model performance by varying α values to answer the question **Q4**. The results are summarized in Tables 5.13 and 5.14.

For highly homophilic graphs such as Cora, Citeseer and Pubmed, higher α values such as 0.6-0.7 provide better performance; whereas for highly heterophilic graphs such as Texas, Cornell and Wisconsin, lower α values such as 0.1-0.2 yield better performance. For the graphs in the middle, there is no clear trend for choosing α values in relating to their homophily ratios. However, we notice that, generally, heterophilic graph datasets can be divided into two categories:

(1) datasets in which nodes within a local neighborhood provide more noise than useful information (Wisconsin, Cornell, Texas and Actor) - these datasets often achieve good performance with a lower α value.

(2) datasets in which informative nodes may locate in both a local neighborhood and distant locations (Chameleon and Squirrel) - these datasets often achieve good performance with a very high α value such as 0.9 in our experiments.

Dataset	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Cora	79.65 ± 0.29	83.46 ± 0.44	86.84 ± 0.41	88.21 ± 0.37	89.65 ± 0.39	**90.29 ± 0.37**	89.88 ± 0.45	89.16 ± 0.57	88.40 ± 0.33
Citeseer	74.37 ± 0.58	75.60 ± 0.76	77.05 ± 0.39	78.13 ± 0.48	79.15 ± 0.47	**79.61 ± 0.48**	79.39 ± 0.23	79.24 ± 0.32	79.08 ± 0.40
Pubmed	82.66 ± 1.58	84.37 ± 0.98	86.98 ± 1.26	88.01 ± 0.71	88.61 ± 0.98	89.32 ± 1.63	**89.97 ± 1.53**	89.00 ± 1.66	88.31 ± 1.57
Photo	91.98 ± 0.21	93.25 ± 0.25	93.82 ± 0.23	**93.84 ± 0.23**	93.81 ± 0.12	93.46 ± 0.36	92.68 ± 0.49	92.14 ± 0.34	91.43 ± 0.19
Computers	84.56 ± 0.44	84.86 ± 0.59	85.32 ± 0.68	**85.76 ± 0.17**	85.61 ± 0.38	85.28 ± 0.61	84.96 ± 0.32	84.75 ± 0.76	84.31 ± 0.93
Chameleon	48.32 ± 1.34	50.67 ± 0.81	52.39 ± 1.66	54.63 ± 0.92	58.35 ± 1.50	60.23 ± 1.67	62.32 ± 0.98	63.67 ± 0.88	**64.91 ± 0.91**
Actor	39.91 ± 0.63	**40.03 ± 0.54**	39.63 ± 1.50	39.26 ± 1.63	39.01 ± 0.59	38.67 ± 1.58	38.32 ± 1.69	37.32 ± 1.50	36.86 ± 1.81
Squirrel	41.71 ± 0.96	42.15 ± 0.56	42.96 ± 0.96	43.41 ± 0.81	44.11 ± 1.29	45.12 ± 1.60	46.05 ± 1.50	46.32 ± 1.23	**47.01 ± 1.96**
Wisconsin	**93.62 ± 1.30**	92.00 ± 1.78	90.25 ± 2.48	87.62 ± 2.33	83.00 ± 2.36	78.37 ± 2.44	72.87 ± 2.66	70.56 ± 2.65	68.37 ± 2.53
Cornell	**93.77 ± 1.22**	93.77 ± 1.42	93.60 ± 1.36	92.73 ± 1.53	91.14 ± 0.80	89.36 ± 1.30	88.43 ± 1.26	86.26 ± 1.58	84.09 ± 1.47
Texas	**94.26 ± 0.81**	93.27 ± 1.71	92.45 ± 1.31	90.67 ± 1.36	89.50 ± 2.34	87.04 ± 1.36	84.23 ± 1.53	81.47 ± 1.47	79.67 ± 1.08

Table 5.13: Classification accuracy (%) averaged over 10 runs with DPRN-FE, where the teleportation parameter $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$.

Dataset	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Cora	78.50 ± 0.88	83.76 ± 0.79	86.64 ± 0.86	88.34 ± 0.69	89.87 ± 0.53	**90.18 ± 0.36**	89.35 ± 0.80	89.01 ± 0.48	88.54 ± 0.42
Citeseer	75.34 ± 0.72	76.53 ± 0.92	76.69 ± 1.01	76.99 ± 1.09	78.31 ± 0.96	**79.84 ± 0.86**	77.36 ± 0.67	76.93 ± 0.88	76.57 ± 0.60
Pubmed	83.28 ± 0.83	84.69 ± 0.91	86.92 ± 0.68	88.32 ± 1.50	89.68 ± 1.26	**90.80 ± 1.96**	90.26 ± 1.30	89.68 ± 1.60	89.02 ± 1.63
Photo	91.96 ± 0.23	92.36 ± 1.23	92.86 ± 1.50	93.01 ± 1.30	92.23 ± 0.88	**93.40 ± 0.41**	92.21 ± 0.56	92.00 ± 1.32	91.67 ± 1.56
Computers	84.66 ± 0.96	84.97 ± 1.28	85.01 ± 1.63	85.68 ± 1.32	86.01 ± 1.67	**86.11 ± 0.21**	85.32 ± 0.12	85.03 ± 0.68	84.26 ± 0.96
Chameleon	50.67 ± 1.50	52.36 ± 1.67	52.67 ± 1.23	55.93 ± 0.96	59.63 ± 0.87	61.49 ± 1.26	64.12 ± 0.67	66.36 ± 1.50	**68.56 ± 1.92**
Actor	37.81 ± 1.15	39.31 ± 1.67	**41.82 ± 0.78**	40.83 ± 1.50	40.23 ± 1.50	39.58 ± 1.30	39.11 ± 1.60	38.51 ± 1.26	37.51 ± 1.26
Squirrel	43.62 ± 1.56	43.93 ± 1.11	44.02 ± 0.91	44.56 ± 0.76	45.32 ± 0.61	46.11 ± 0.82	46.88 ± 0.93	48.56 ± 1.66	**50.69 ± 1.75**
Wisconsin	**88.01 ± 1.67**	87.93 ± 1.23	87.82 ± 1.66	87.51 ± 0.93	86.91 ± 0.63	86.51 ± 0.58	86.45 ± 0.65	86.21 ± 0.63	85.02 ± 1.60
Cornell	**92.41 ± 1.67**	92.01 ± 1.50	91.68 ± 0.96	91.36 ± 0.62	90.66 ± 0.59	89.21 ± 1.23	88.32 ± 0.96	86.02 ± 1.32	85.31 ± 1.50
Texas	94.16 ± 0.96	**94.45 ± 0.88**	93.59 ± 1.53	92.61 ± 1.60	91.32 ± 0.75	89.23 ± 0.66	87.32 ± 0.45	83.21 ± 0.98	80.48 ± 0.65

Table 5.14: Classification accuracy (%) averaged over 10 runs with DPRN-IF, where the teleportation parameter $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$.

5.5.6 Ablation Analysis

In this section, we evaluate the impact of learnable polynomial filtering and dynamic PageRank techniques for the performance of DPRN-IF and DPRN-FE on fully-supervised node classification task to answer the question **Q5**.

Let $\mathcal{M}(\mathbf{P})$, where $\mathcal{M} \in \{\text{PPR, HK, DPRN-FE, DPRN-IF}\}$ denote a GNN model with the following variants for a diffusion scheme and $\mathbf{P} \in \{\mathbf{P}_{RW}, \mathbf{P}_{Filter}\}$ for a transition matrix. Here, PPR and HK refer to the personalised PageRank and heat kernel, respectively; $\mathbf{P}_{RW}$ and $\mathbf{P}_{Filter}$ refer to a random walk transition matrix $\mathbf{AD}^{-1}$ and learnable polynomial filter defined in Eq. 5.10, respectively. Note that,

(1) PPR ($\mathbf{P}_{RW}$) and HK ($\mathbf{P}_{RW}$) are the same as in the standard PageRank with a random walk transition matrix.

(2) DPRN-FE ($\mathbf{P}_{Filter}$) and DPRN-IF ($\mathbf{P}_{Filter}$) are the same as DPRN-FE and DPRN-IF in our previous experiments with learnable polynomial filters.

The results on eleven datasets can be found in Tables 5.15 and 5.16. For all diffusion schemes, using $\mathbf{P}_{Filter}$ consistently improves the performance on all datasets in comparison with using $\mathbf{P}_{RW}$. This shows that learnable polynomial filters can enhance the model learning ability effectively. The diffusion schemes built upon dynamic PageRank (i.e. DPRN-FE and DPRN-IF) further improve the performance in comparison with using the standard PageRank.

Variants	Cora	Citeseer	Pubmed	Photo	Computers
PPR ($\mathbf{P}_{RW}$)	87.93 ± 0.66	78.96 ± 1.25	88.56 ± 1.30	90.23 ± 0.96	81.63 ± 0.68
PPR ($\mathbf{P}_{Filter}$)	**89.12 ± 1.63**	**80.00 ± 0.96**	**89.73 ± 1.83**	**91.90 ± 0.91**	**83.57 ± 0.88**
HK ($\mathbf{P}_{RW}$)	86.13 ± 0.66	76.93 ± 0.86	86.38 ± 1.56	87.89 ± 1.20	80.61 ± 1.65
HK ($\mathbf{P}_{Filter}$)	**87.31 ± 1.32**	**77.42 ± 0.96**	**87.21 ± 0.83**	**88.93 ± 1.34**	**81.31 ± 1.50**
DPRN-FE ($\mathbf{P}_{RW}$)	88.76 ± 0.93	78.27 ± 0.63	88.68 ± 1.67	90.26 ± 0.86	84.21 ± 0.73
DPRN-FE ($\mathbf{P}_{Filter}$)	**90.24 ± 0.42**	**79.84 ± 0.48**	**89.97 ± 1.53**	**93.82 ± 0.23**	**85.76 ± 0.17**
DPRN-IF ($\mathbf{P}_{RW}$)	88.32 ± 0.68	79.36 ± 0.78	88.51 ± 1.23	90.81 ± 0.71	84.49 ± 0.57
DPRN-IF ($\mathbf{P}_{Filter}$)	**90.18 ± 0.36**	**80.39 ± 0.73**	**90.80 ± 1.96**	**93.40 ± 0.41**	**86.11 ± 0.21**

Table 5.15: Classification accuracy (%) averaged over 10 runs on homophilic datasets.

Variants	Actor	Wisconsin	Cornell	Texas	Chameleon	Squirrel
PPR ($\mathbf{P}_{RW}$)	37.33 ± 0.32	85.67 ± 1.36	90.96 ± 1.60	90.01 ± 1.45	55.36 ± 0.93	34.66 ± 1.25
PPR ($\mathbf{P}_{Filter}$)	**39.07 ± 1.66**	**86.88 ± 1.87**	**92.21 ± 0.86**	**91.30 ± 1.51**	**57.21 ± 0.76**	**36.23 ± 0.93**
HK ($\mathbf{P}_{RW}$)	35.96 ± 1.30	84.93 ± 1.72	88.62 ± 0.81	86.15 ± 1.36	51.23 ± 1.68	33.69 ± 1.60
HK ($\mathbf{P}_{Filter}$)	**36.71 ± 1.50**	**86.01 ± 1.50**	**89.68 ± 0.96**	**87.16 ± 1.60**	**53.66 ± 1.98**	**34.61 ± 1.96**
DPRN-FE ($\mathbf{P}_{RW}$)	38.67 ± 1.36	90.66 ± 0.91	91.31 ± 1.60	92.67 ± 1.56	63.16 ± 1.60	44.01 ± 1.93
DPRN-FE ($\mathbf{P}_{Filter}$)	**40.03 ± 0.54**	**93.75 ± 0.79**	**93.44 ± 1.26**	**94.75 ± 0.65**	**65.00 ± 2.31**	**47.31 ± 2.01**
DPRN-IF ($\mathbf{P}_{RW}$)	39.51 ± 0.98	86.11 ± 1.66	90.27 ± 1.56	91.41 ± 0.83	66.21 ± 1.46	47.21 ± 1.87
DPRN-IF ($\mathbf{P}_{Filter}$)	**41.82 ± 0.78**	**88.37 ± 2.09**	**92.78 ± 1.67**	**94.59 ± 0.85**	**68.56 ± 1.92**	**50.69 ± 1.75**

Table 5.16: Classification accuracy (%) averaged over 10 runs on heterophilic datasets.

5.6 Summary

In this section, we have proposed a novel Graph Neural Network (GNN) architecture, namely *Dynamic PageRank Networks* (DPRNs). Our architecture leverages dynamics of PageRank to capture rich and varying graph structures. To enhance the discriminative power of PageRank diffusion on graphs, we encoded local topological information into a learnable PageRank transition matrix via learning polynomial filter coefficients efficiently by formulating a quadratic convex constrained optimization problem as a convex function. Although dynamic PageRank generally does not converge, we theoretically proved how our GNN architecture is designed to achieve the guaranteed convergence. We showed that shallow GNN architectures with deeper single layers represent a promising direction for adapting the different graph structured data (i.e., homophilic and heterophilic). We evaluated our models on benchmark tasks.

Part II

Graph Similarity Learning

Chapter 6

A Regularized Optimal Transport Framework for Graph Kernels

6.1 Overview

Recently, several studies have considered optimal transport learning on graphs [182, 245, 164], which amounts to two kinds of graph aligning problems: (1) aligning graphs in the same ground space and (2) aligning graphs across different ground spaces. A ground space is an Euclidean space with some fixed dimension. Recent work has considered methods to jointly deal with these aligning problems based on the similarity of node features and pairwise distances [243]. However, this is still inadequate due to several reasons. Firstly, these methods did not *explicitly* capture the connection between node features and structures into transport costs, which limits the learning ability. Secondly, these methods ignored local clustering structures. Lastly, these methods did not exploit degree distributions when learning on pairwise distances of vertices.

In this chapter, we propose to capture feature local variations which quantify how features change upon the underlying structures of a graph. We explicitly incorporate feature local variations into feature similarity matrices and accordingly into a cost function to enhance optimal transport learning. Further, we propose a new optimal transport distance metric on graphs, called *Regularized Wasserstein (RW) discrepancy*. This RW discrepancy regularizes optimal transport learning to compute a distance between graphs via two *strongly convex* regularization terms. One is to regularize a Wasserstein distance between graphs in the same ground space. This regularization relaxes an optimal alignment between graphs to be a cluster-to-cluster mapping between their locally connected vertices, thereby preserving the local clustering structures of vertices across graphs. The other is to regularize a Gromov-Wasserstein distance between graphs across different ground spaces using a degree-entropy KL divergence term. This regularization considers node degree distributions in order to increase the matching robustness of an optimal alignment, allowing to distribute probability masses smoothly in overlapping regions of the geometric spaces of graphs. Together with feature similarity matrices that capture features and their local variations in cost functions, our regularized optimal transport learning can preserve both local

and global structures of graphs during the transport, in addition to features.

Although our framework provides a powerful optimal transport learning for graph kernels, the corresponding optimization problem is NP-hard and thus computationally difficult in the general case [197, 4], due to its non-convexity and combinatorial nature. To circumvent this problem, we design an efficient algorithm, namely *Sinkhorn Conditional Gradient (SCG)*, which reaps the computational benefits of the proposed strongly convex regularization terms and extends the conditional gradient with *Sinkhorn-knopp* matrix scaling [126] to enable a fast approximation for solving the optimization problem. We theoretically analyze the convergence properties of SCG and prove the upper bound of its minimal suboptimality gap.

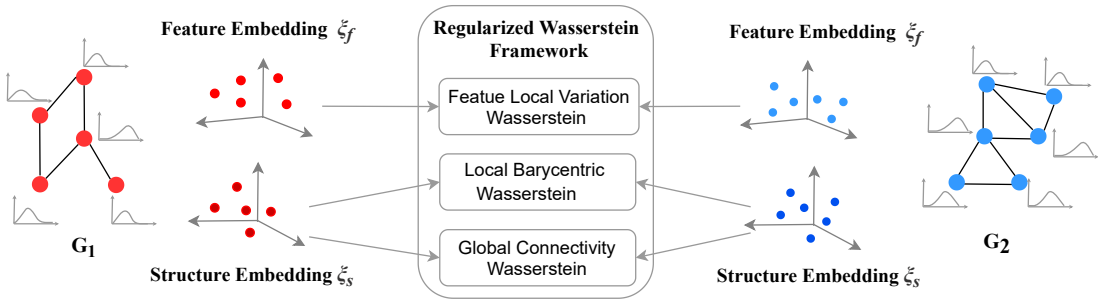


Figure 6.1: An overview of the proposed framework for regularized Wasserstein kernels (RWKs), which unifies feature local variation, local barycentric and global connectivity Wasserstein distances based on feature and structure embeddings.

The main contributions of this chapter are as follows:

- We propose a theoretically robust class of graph kernels (i.e., RWKs) based on a new optimal transport distance metric which optimizes graph aligning problems in the same or across different ground spaces by exploiting strongly convex regularisation.
- We improve the geometric representation of graphs by incorporating feature local variations into similarity matrices, which can explicitly preserve the connection between features and structures of a graph.
- We devise a fast and numerically stable algorithm to solve the optimization problem and theoretically prove the suboptimal gap of our algorithm converges at the rate of $O(\frac{1}{\sqrt{k}})$ where k is the number of iterations.

The rest of this chapter is organized as follows. In Section 6.2, we present the graph similarity matrices. In Section 6.3, we discuss the regularized Wasserstein framework. In Section **??**, we analyze the convergence and complexity of the proposed regularized Wasserstein kernel method. In Section 6.5, we discuss the experimental setup. In Section 6.6, we compare the performance of our proposed regularized Wasserstein kernel method against the baseline methods. Section 6.7 summarises the chapter.

6.2 Graph Similarity Matrices

In this section we discuss the feature and structural representations of graphs and several cost functions for optimal transport learning on graphs.

6.2.1 Feature Similarity

Following the previous work [164], we consider features residing on vertices as graph signals. For a graph $G = (V, E)$, a *graph signal* is a mapping $V \to \mathbb{R}$ that associates a feature to a vertex. Thus, each graph has a *graph signal matrix* $\mathbf{X} \in \mathbb{R}^{n \times m}$, where $n = |V|$ is the number of vertices in the graph and each vertex v_i is associated with graph signals $x_i \in \mathbb{R}^m$.

To quantify how graph signals change from a vertex to its neighboring vertices, we formulate the notion of feature local variation. Let $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ be the normalised graph Laplacian of G, where $\mathbf{D}$ is the diagonal matrix, $\mathbf{A}$ is the adjacency matrix and $\mathbf{I}$ is the identity matrix. Then the *local variation matrix* of G is defined as:

$$\Delta(\mathbf{X}) = \left| \mathbf{X} - \frac{\mathbf{L}^j \mathbf{X}}{\lambda_{max}(\mathbf{L})} \right|. \tag{6.1}$$

$\mathbf{L}^j\mathbf{X}$ refers to aggregated graph signals of all vertices in G within the j-hop neighborhood. $\lambda_{max}(\mathbf{L})$ is the maximum eigenvalue of $\mathbf{L}$, which normalises $\mathbf{L}^j\mathbf{X}$ to ensure the numerical stability. $\Delta(\mathbf{X})$ represents the local variations of features computed by taking the difference between the original graph signal matrix $\mathbf{X}$ and the aggregated graph signal matrix $\mathbf{L}^j\mathbf{X}$.

Let $\xi_f : V \to \mathbb{R}^m$ and $\xi_s : V \to \mathbb{R}^k$ refer to the *feature embedding* function that associates with feature representation on each vertex in a metric space $(\mathbb{R}^m, d_f)$ and the *structure embedding* function that associates with structural representation on each vertex in a metric space $(\mathbb{R}^k, d_s)$, respectively. Let $x_i \in \mathbb{R}^m$ and $\Delta(x_i) \in \mathbb{R}^m$ refer to the graph signals of a vertex v_i and its local variation in G, respectively. Then, each vertex v_i corresponds to a feature embedding vector $a_i = \xi_f(v_i) \in \mathbb{R}^{2m}$ such that $a_i = x_i \oplus \Delta(x_i)$, where $\oplus$ refers to the concatenation. Given two graphs G_1 and G_2, a *feature similarity matrix* between G_1 and G_2 is defined upon the concatenation of their graph signals and local variations, i.e., $\mathbf{C}^V(i,j) = (d_f(a_i, a_j))_{i,j} \in \mathbb{R}^{n_1 \times n_2}$, where a_i and a_j are the feature embedding vectors of the *i-th* vertex of G_1 and *j-th* vertex of G_2, respectively.

6.2.2 Structure Similarity

For each vertex $v_i \in V$ in a graph, we associate it with a node embedding vector $e_i = \xi_s(v_i) \in \mathbb{R}^k$. Node embedding methods with random-walk follow two steps: (1) compute node sequences by random-walk simulation and (2) feed those sequences into word2vec algorithm [138] to learn node embeddings. Abu-El-Haija et al. [1] have been proposed a method that allows to learn parameters used in random-walk simulation (i.e., context window length c, which is sampled from a uniform

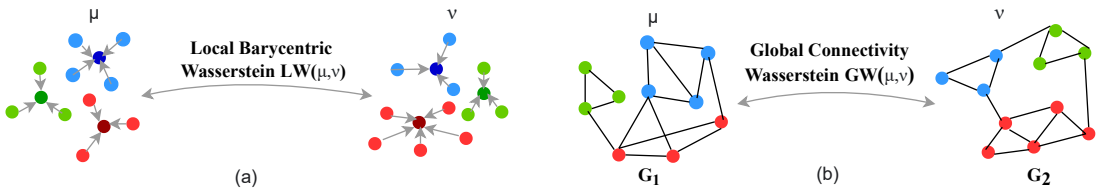


Figure 6.2: (a) shows the local barycentric Wasserstein distance that transports each vertex in μ to a spatially localized barycenter of its corresponding neighbors in ν and vice versa; (b) shows the global connectivity Wasserstein distance that captures the pairwise similarity between vertices under the preservation of degree distributions.

distribution $c \in \mathrm{U}\{1, C\}$) and learn context sampling in word2vec (i.e., learn context distribution S, which is used by word2vec when select the context node). In particular, Abu-El-Haija et al. [1] compute a pair-wise node-to-node co-occurrence matrix $\mathbf{B}$ from random-walk simulations to measure how many times that each node $u \in V$ is sampled as part of the context node $v \in \mathcal{N}(u)$. Then, we can compute node embeddings by factorization the matrix $\mathbf{B}$ [194]. There are many different ways to factorizing a matrix such as singular value decomposition (SVD) [123] and probabilistic objectives. Abu-El-Haija et al. [1] consider a probabilistic objective called, Negative Log Graph Likelihood for matrix factorization, which is defined as follows:

$$\min_{\mathbf{Y}} \big|\big| - \mathbf{B} \odot log(\sigma(g(\mathbf{Y}))) - \mathbb{1} \odot [\mathbf{A} = 0] log(1 - \sigma(g(\mathbf{Y})))\big|\big|_1, \tag{6.2}$$

where $g(\mathbf{Y})$ refers to the model, $\mathbf{Y}$ is a node embedding dictionary, σ is the logistic activation function, $\odot$ is a hadamard product and $\mathbb{1}$ is an indicator function. The model $g(Y) = g([L|R]) = L \times R^T$ is defined as an outer product of two matrices. Then, Eq. 6.2 minimize the expression w.r.t matrices L and R to factorizing the matrix $\mathbf{B}$.

Let S be a context distribution with c-dimensional vector $S = (S_1, S_2, \ldots, S_c)$, where $S_t \geq 0$, $t \leq c$ and $\sum_t S_t = 1$. Then, we can assign t-th coefficient S_t to random-walk probability transition matrix $\mathbf{M}^t$. We can use expectation on $\mathbf{B}$ (i.e., $\mathbb{E}[\mathbf{B}]$), instead of using $\mathbf{B}$ itself [1]. Formally, we can formulate the $\mathbb{E}[\mathbf{B}]$ by parameterisation with S as follows:

$$\mathbb{E}[\mathbf{B}; S_1, \cdots, S_c] = \tilde{\mathbf{P}}^{(0)} \sum_{t=1}^{C} S_t (\mathbf{M})^t = \tilde{\mathbf{P}}^{(0)} \mathop{\mathbb{E}}_{t \sim S} [(\mathbf{M})^t], \tag{6.3}$$

where $\tilde{\mathbf{P}}^{(0)}$ is a diagonal matrix with initial positions that corresponds to the number of walks starting at each node. In this way, authors of [1] show that embeddngs learning with random-walk sequences using Glove or word2vec can be considered as a special case of Eq. 6.3. In the final objective, they learn a context distribution S, which is expressed as a softmax attention model with the power-series of the random-walk probability transition matrix $\mathbf{M}$. More specifically, we construct a probability transition matrix $\mathbf{M}$ using heat kernel random walks, where $\mathbf{M}^t = e^{-t\mathbf{L}}$, where t

is the length of random walks and $\mathbf{L}$ is the graph Laplacian. This graph attention mechanism guides the sampling process of random walks to optimize an objective Negative Log Graph Likelihood [1].

Based on the node embeddings, we consider the following two kinds of structure similarity:

(1) *Neighbourhood similarity.* For two graphs G_1 and G_2, we define a *neighbourhood similarity matrix* as $\mathbf{C}^N(i,j) = (d_s(e_i, e_j))_{i,j} \in \mathbb{R}^{n_1 \times n_2}$ where e_i and e_j represent the node embeddings of the *i-th* vertex of G_1 and the *j-th* vertex of G_2, respectively.

(2) *Pairwise similarity.* For a graph G, we construct a *pairwise similarity matrix* by $\mathbf{C}^P(i,j) = (d_s(e_i, e_j))_{i,j} \in \mathbb{R}^{n \times n}$, where e_i and e_j represent the node embeddings of the *i-th* vertex and and the *j-th* vertex of G. Let $\mathbf{C}_1^P \in \mathbb{R}^{n_1 \times n_1}$ and $\mathbf{C}_2^P \in \mathbb{R}^{n_2 \times n_2}$ represent the pairwise similarity matrices of two graphs G_1 and G_2, respectively. Then, the pairwise similarity between G_1 and G_2 is defined as a 4-*dimensional* tensor:

$$L_2(\mathbf{C}_1^P(i,j), \mathbf{C}_2^P(k,l)) = \frac{1}{2}|\mathbf{C}_1^P(i,j) - \mathbf{C}_2^P(k,l)|^2.$$

6.3 Regularized Wasserstein Framework

In this section, we introduce a novel optimal transport framework for graphs. This framework can preserve local and global graph structures by jointly optimizing two regularized optimal transports on graphs: (1) local barycentric Wasserstein distance; (2) global connectivity Wasserstein distance. We discuss these two kinds of Wasserstein distances in turn.

6.3.1 Local Barycentric Wasserstein Distance

We first propose a local-structure-preserving optimal transport based on Laplacian regularization [76, 79]. To preserve the local structure of graphs, we observe that a relaxed mapping (i.e., cluster-to-cluster) between locally connected vertices of two graphs is often more desirable than a strict one-to-one correspondence between vertices of two graphs. Thus, we design a regularization term $\Theta_w(\gamma)$ under a relaxation of transport mass conservation [80] to regularize a Wasserstein distance defined on the neighbourhood similarity matrix $\mathbf{C}^N$:

$$LW(\mu, \nu) = \min_{\gamma \in \pi(\mu,\nu)} \left\langle \gamma, \mathbf{C}^N \right\rangle_F + \Theta_w(\gamma), \tag{6.4}$$

where $\left\langle ., . \right\rangle_F$ denotes the Frobenius dot product.

In the following, we discuss how $\Theta_w(\gamma)$ is designed. Essentially, $\gamma(i,j)$ indicates how much the probability mass of the *i-th* vertex in one graph μ is transported to the *j-th* vertex in the other graph ν. Thus, we define a transport map T from μ to ν by mapping the node embedding of each vertex e_i^μ in μ to a weighted average $\hat{e}_i^\mu$ of the

node embeddings of vertices in ν:

$$\hat{e}_i^{\mu} = T(e_i^{\mu}) = \frac{\sum_{j=1}^{n_2} \gamma(i,j) e_j^{\nu}}{\sum_{j=1}^{n_2} \gamma(i,j)}. \tag{6.5}$$

Let $\mathbf{E}_{\mu} \in \mathbb{R}^{n_1 \times k}$ (resp. $\mathbf{E}_{\nu} \in \mathbb{R}^{n_2 \times k}$) be a node embedding matrix of μ (resp. ν). We thus have the following matrix of local barycentric embeddings:

$$\hat{\mathbf{E}}_{\mu} = T(\mathbf{E}_{\mu}) = (diag(\gamma \mathbf{1}_{n_2}))^{-1} \gamma \mathbf{E}_{\nu}, \tag{6.6}$$

where $diag(.)$ is a diagonal matrix in $\mathbb{R}^{n_1 \times n_1}$. To preserve the local structure of vertices in μ under T, we define a spatially localized barycentric term as the *source regularization*:

$$\begin{aligned} \Omega_{\mu}(\gamma) &= \frac{1}{n_1^2} \sum_{i,j} a_{i,j} \|\hat{e}_i^{\mu} - \hat{e}_j^{\mu}\|_2^2 \\ &= \frac{1}{n_1^2} tr(\hat{\mathbf{E}}_{\mu}^T \mathbf{L}_{\mu} \hat{\mathbf{E}}_{\mu}). \end{aligned} \tag{6.7}$$

$\mathbf{L}_{\mu}$ is the graph Laplacian and $\mathbf{A}_{\mu} = (a_{i.j})_{i,j=1}^{n_1}$ is the adjacency matrix of μ. When μ and ν are uniform distributions, $\hat{\mathbf{E}}_{\mu} = n_1 \gamma \mathbf{E}_{\nu}$ and thus

$$\Omega_{\mu}(\gamma) = tr(\mathbf{E}_{\nu}^T \gamma^T \mathbf{L}_{\mu} \gamma \mathbf{E}_{\nu}). \tag{6.8}$$

Similarly, we define a spatially localized barycentric term $\Omega_{\nu}(\gamma)$ as the *target regularization* to preserve the local structure of vertices in ν under the transport map T^{-1}. By $\Omega_{\mu}(\gamma)$ and $\Omega_{\nu}(\gamma)$, we obtain the following regularization term to constrain local barycentric Wasserstein distance, where $0 \leq \lambda_{\mu}, \lambda_{\nu} \leq 1$:

$$\Theta_w(\gamma) = \lambda_{\mu} \Omega_{\mu}(\gamma) + \lambda_{\nu} \Omega_{\nu}(\gamma) + \frac{\rho}{2} ||\gamma||_F^2. \tag{6.9}$$

This regularization term enables us to avoid the strict mass conservation (i.e, a bijective mapping between μ and ν) because each vertex in μ is transported to a spatially localized barycenter of its corresponding neighbors in ν and vice versa. A penalty term $||\gamma||_F^2$ is introduced to smooth the transport mass conservation. The parameter $\rho \in (0, 1]$ controls the degree of smoothness.

Lemma 3. *$LW(\mu, \nu)$ is strongly convex and smooth w.r.t. γ.*

Proof. Let $f_1(\gamma) = \lambda_{\mu} \Omega_{\mu}(\gamma) + \lambda_{\nu} \Omega_{\nu}(\gamma)$ and $f_2(\gamma) = \frac{\rho}{2} ||\gamma||_F^2$. The Hessian of $\Omega_{\mu}(\gamma)$ is,

$$\nabla^2 \Omega_{\mu}(\gamma) = \mathbf{L}_{\mu} \otimes \mathbf{E}_{\nu} \mathbf{E}_{\nu}^T + \mathbf{L}_{\mu}^T \otimes \mathbf{E}_{\nu} \mathbf{E}_{\nu}^T, \tag{6.10}$$

where $\otimes$ denotes the Kronecker product. $\mathbf{L}_{\mu}$ is positive semi-definite since its eigenvalues are non-negative. We also have $z^T(\mathbf{E}_{\nu} \mathbf{E}_{\nu}^T) z = ||\mathbf{E}_{\nu}^T z||_2^2 \geq 0$ for every $z \neq 0$ and $z \in \mathbb{R}^{n_2 \times 1}$, which is positive semi-definite. Thus, $\mathbf{L}_{\mu} \otimes \mathbf{E}_{\nu} \mathbf{E}_{\nu}^T$ is positive semi-

definite since the Kronecker product of two positive semi-definite matrices is positive semi-definite [219]. Therefore, $\Omega_\mu(\gamma)$ is convex, and similarly, we can show $\Omega_\nu(\gamma)$ is convex. Hence, $f_1(\gamma)$ is convex w.r.t γ. Since the function $||\gamma||_F^2$ is quadratic w.r.t γ, the Hessian of $f_2(\gamma)$ is positive definite. Hence, $f_2(\gamma)$ is strongly convex. Then, the sum of $f_1(\gamma) + f_2(\gamma)$ (i.e., $\Theta_w(\gamma)$) is ρ-strongly convex. Since $f_1(\gamma)$ is positive semi-definite and $f_2(\gamma)$ is positive definite, $\Theta_w(\gamma)$ is positive definite. Hence, $\Theta_w(\gamma)$ is L-smooth for some constant $L > 0$.

Since $\langle \gamma, \mathbf{C}^N \rangle_F$ is convex and $\Theta_w(\gamma)$ is strongly convex and smooth, $LW(\mu, \nu)$ is strongly convex and smooth. □

6.3.2 Global Connectivity Wasserstein Distance

To preserve the global structure of graphs during the transport, such as structure connectivity, a straightforward approach is to use a Gromov-Wasserstein discrepancy based on pairwise similarity between vertices. However, solving such an unregularized Gromov-Wasserstein optimization problem on $\langle \gamma, L_2(\mathbf{C}_\mu^P, \mathbf{C}_\nu^P) \otimes \gamma \rangle_F$ may lead to a sparse coupling matrix γ, i.e. the entries of γ become mostly zero. As a result, only few vertices between two graphs can be matched. Further, the degree distributions between graphs need to be considered for preserving structure connectivity, Thus, we design a *degree-entropy regularization term* $\Theta_g(\gamma)$ to regularize a Gromov-Wasserstein distance on the pairwise similarity matrix $\mathbf{C}^P$:

$$GW(\mu, \nu) = \min_{\gamma \in \pi(\mu,\nu)} \langle \gamma, L_2(\mathbf{C}_\mu^P, \mathbf{C}_\nu^P) \otimes \gamma \rangle_F - \lambda_g \Theta_g(\gamma),$$

where $\lambda_g \in (0, 1]$ and $\langle \gamma, L_2(\mathbf{C}_\mu^P, \mathbf{C}_\nu^P) \otimes \gamma \rangle_F = \sum_{i,j,k,l} L_2(\mathbf{C}_\mu^P(i,j), \mathbf{C}_\nu^P(k,l)) \gamma(i,k) \gamma(j,l)$. Specifically, we define $\Theta_g(\gamma)$ as a KL divergence between γ and a prior node degree distribution γ':

$$\Theta_g(\gamma) = KL(\gamma \| \gamma') = \sum_{i,j} \gamma(i,j) log\Big(\frac{\gamma(i,j)}{\gamma'(i,j)}\Big). \tag{6.11}$$

Let $D_\mu \in \mathbb{R}^{n_1}$ and $D_\nu \in \mathbb{R}^{n_2}$ represent the node degree vectors of graphs G_1 and G_2, respectively. We have:

$$\begin{aligned} \gamma'(i,j) &= \frac{\tilde{\gamma}(i,j)}{||\sum_j \tilde{\gamma}(i,j)||_1} \\ \tilde{\gamma}(i,j) &= 1 - \frac{|D_\mu^i - D_\nu^j|}{\max\{D_\mu^i, D_\nu^j\}} \end{aligned} \tag{6.12}$$

Note that, depending on how pairwise similarity matrices are defined, different kinds of global structures can be preserved. When $\mathbf{C}_\mu^P$ and $\mathbf{C}_\nu^P$ are shortest path distance matrices, we preserve the connectivity structure of graphs. Other options include adjacency matrices and graph Laplacians [220].

Lemma 4. *$KL(\gamma||\gamma')$ is strongly convex w.r.t γ.*

Proof. We can compute the Hessian of $KL(\gamma||\gamma')$ as follows,

$$\nabla^2 KL(\gamma||\gamma') = diag\Big(\frac{1}{\gamma(i,j)}\Big), \tag{6.13}$$

where $\gamma(i,j) \in [0,1]$. Since a function f is *σ-strongly convex* iff there exists a constant $\sigma > 0$ s.t. its Hessian satisfies $\nabla^2 f(\gamma) \succeq \sigma \mathbf{I}$, $\forall \gamma \in$ dom f, where $\mathbf{I}$ refers to an identity matrix, $KL(\gamma||\gamma')$ is 1-strongly convex because $z^T(\nabla^2 KL(\gamma||\gamma'))z \geq \sigma||z||^2$ and $\sigma = 1$. □

Although $GW(\mu,\nu)$ remains non-convex, the strong convexity of $KL(\gamma||\gamma')$ enables better optimization convergence (will be discussed further in Section 6.4).

6.3.3 A New Optimal Transport Distance Metric on Graphs

In the following, we present the *Regularized Wasserstein* (RW) discrepancy to preserve both features and structure of graphs. The main idea is to consider local barycentric and global connectivity Wasserstein distances, as well as Wasserstein distance for features and their local variations, in a unified framework. We also discuss our optimization technique and analyze the theoretical properties.

Let $\beta_1, \beta_2 \in (0,1]$ and $\mathbf{C}^V$ be a feature similarity matrix containing the information of features and their local variations. Formally, the RW discrepancy is defined as follows:

$$\begin{aligned} RW(\mu,\nu) = \min_{\gamma \in \pi(\mu,\nu)} & \big\langle \gamma, \mathbf{C}^V \big\rangle_F \\ & + \beta_1 LW(\mu,\nu) + \beta_2 GW(\mu,\nu). \end{aligned} \tag{6.14}$$

In a nutshell, the RW discrepancy derives an optimal coupling γ by minimizing a linear combination of costs of transporting graph features and their local variations, transporting vertices and transporting edges across two graphs.

Solving an unregularized Gromov-Wasserstein optimization problem in its full generality is known to be NP-hard [4, 244]. The optimization problem for Eq. 6.14 is thus also NP-hard. Therefore, the convergence to the optimality of RW is a non-trivial and difficult problem. Below, we present a solution to tackle this difficult problem.

Firstly, we transform the optimization problem for Eq. 6.14 into an equivalent problem with the following form of objective:

$$\min_{\gamma \in \pi(\mu,\nu)} H(\gamma) = \min_{\gamma \in \pi(\mu,\nu)} f(\gamma) + g(\gamma) - h(\gamma), \tag{6.15}$$

where we have:

$$\begin{aligned} f(\gamma) =& \big\langle \gamma, \mathbf{C}^V \big\rangle_F + \beta_1 LW(\mu,\nu); \\ g(\gamma) =& \big\langle \gamma, \beta_2 (L_2(\mathbf{C}^P_\mu, \mathbf{C}^P_\nu) \otimes \gamma) \big\rangle_F; \\ h(\gamma) =& \beta_2 (\lambda_g \Theta_g(\gamma)). \end{aligned}$$

Algorithm 2: Training for RW Discrepancy

initialize i=0, $\gamma^0 \leftarrow \mu\nu^T$, and $c^0 \leftarrow H(\gamma^0)$
while $i \leq t$ **do**
  $i \leftarrow i + 1$
  $\nabla H(\gamma) \leftarrow$ Gradient of $H(\gamma)$ w.r.t $\gamma^{(i-1)}$
  $\hat{\gamma}^{(i-1)} \leftarrow$ *Sinkhorn-knopp* (μ, ν, $\nabla H(\gamma)$, λ, b)
  $\Delta\gamma \leftarrow \hat{\gamma}^{(i-1)}$-$\gamma^{(i-1)}$
  $\alpha^{(i)}, c^{(i)} \leftarrow$ *Line-search* ($\gamma^{(i-1)}$, $\Delta\gamma$, $\nabla H(\gamma)$, $c^{(i-1)}$) w.r.t. Eq. 6.15
  $\gamma^{(i)} \leftarrow \gamma^{(i-1)} + \alpha^{(i)}\Delta\gamma$
  $\delta^{(i-1)} \leftarrow \left\langle \Delta\gamma, -\nabla H(\gamma) \right\rangle_F$
  if $\delta^{(i-1)} \leq \tilde{\epsilon}$ **then**
    stop
  end
end

Then, we design a training algorithm for RW discrepancy, namely *Sinkhorn Conditional Gradient* (SCG), based on Conditional Gradient [114], which is described in Algorithm 2. The main idea is to linearize the composite objective function in Eq. 6.15, where t is the maximum number of iterations for SCG and b is the maximal number of Sinkhorn iterations. In each iteration, we compute an optimal coupling matrix $\hat{\gamma}^{(i-1)}$ based on the gradient of $H(\gamma)$ by *Sinkhorn-knopp*, where $\lambda \in [0, \infty]$ and obtain the descent direction $\Delta\gamma$. Then, we use *Line-search* to determine the step size $\alpha^{(i)}$ based on the gradient of $H(\gamma)$ along the descent direction $\Delta\gamma$. The algorithm terminates if the suboptimality gap converges under a threshold $\tilde{\epsilon}$, i.e., $\delta^{(i-1)} \leq \tilde{\epsilon}$.

The gradients of $f(\gamma)$ and $g(\gamma)$ are calculated as follows:

$$\begin{aligned}
\nabla f(\gamma) =& \mathbf{C}^V + \beta_1(\mathbf{C}^N) + \\
& \beta_1(\lambda_\mu \nabla\Omega_\mu(\gamma) + \lambda_\nu \nabla\Omega_\nu(\gamma) + \rho\gamma); \\
\nabla g(\gamma) =& 2\beta_2(L_2(\mathbf{C}_\mu^P, \mathbf{C}_\nu^P) \otimes \gamma); \\
\nabla h(\gamma) =& \beta_2(\lambda_g(1 + log(\gamma) - log(\gamma'))),
\end{aligned}$$

where

$$\begin{aligned}
\nabla\Omega_\mu(\gamma) =& \frac{\partial(\Omega_\mu(\gamma))}{\partial\gamma} = \mathbf{L}_\mu^T \gamma \mathbf{E}_\nu \mathbf{E}_\nu^T + \mathbf{L}_\mu \gamma \mathbf{E}_\nu \mathbf{E}_\nu^T; \\
\nabla\Omega_\nu(\gamma) =& \frac{\partial(\Omega_\nu(\gamma))}{\partial\gamma} = \mathbf{E}_\mu \mathbf{E}_\mu^T \gamma \mathbf{L}_\nu^T + \mathbf{E}_\mu \mathbf{E}_\mu^T \gamma \mathbf{L}_\nu.
\end{aligned}$$

SCG has nice convergence properties. It is guaranteed to converge to a stationary point. We define suboptimality gap [114] for SCG and present the theoretical results in Section 6.4.

6.3.4 A Regularized Wasserstein Kernel (RWK)

We introduce a new graph kernel, namely *Regularized Wasserstein Kernel* (RWK), based on our RW discrepancy presented in Section 6.3.3. Given a set of graphs $\mathcal{G}$, RWK has a *kernel matrix* $\mathbf{K} \in \mathbb{R}^{|\mathcal{G}|\times|\mathcal{G}|}$ defined as

$$\mathbf{K}_{\mu\nu} = e^{-\eta RW(\mu,\nu)},$$

where $\eta > 0$ is a parameter, μ and ν correspond to any two graphs in $\mathcal{G}$, and $RW(\mu,\nu)$ is the RW discrepancy between μ and ν as defined in Eq. 6.14.

Here, $\mathbf{K}$ is an indefinite kernel matrix. Following SVM with indefinite kernels introduced by Luss and d'Aspremont [159], we treat $\mathbf{K}$ as the noisy observation of a true positive semi-definite kernel (i.e., a proxy kernel). Thus, our graph classification problem with an indefinite RWK can be expressed as a robust classification problem under a perturbation of the true positive semidefinite kernel. This formulation allows us to learn support vector weights and a proxy kernel simultaneously, while penalizing the distance between the indefinite RWK and the proxy kernel in the same way as studied in [159].

6.4 Theoretical Analysis

In this section, we prove that our training algorithm (SCG) can guarantee the convergence. We also provide the complexity analysis of our RWK method in Section 6.4.2.

6.4.1 Convergence Analysis

Definition 4 (L-smooth function)**.** *A function $f : \mathbb{R}^{n_1\times n_2} \to \mathbb{R}$ is* L-smooth *if it is differentiable and gradient is L-Lipschitz continuous for some Lipschitz constant $L > 0$, i.e.,*

$$||\nabla f(\gamma) - \nabla f(\bar{\gamma})||_F \leq L||\gamma - \bar{\gamma}||_F \tag{6.16}$$

where γ and $\bar{\gamma}$ are any two points in the domain f.

Lemma 5. *Assuming $f(\cdot)$ and $g(\cdot)$ in Eq.6.15 are L-smooth for some constants $L_f > 0$ and $L_g > 0$, respectively. Then, the sum $f(\cdot) + g(\cdot) : \mathbb{R}^{n_1\times n_2} \to \mathbb{R}$ is L-smooth for a constant $L = L_f + L_g$, i.e.,*

$$||\nabla(f+g)(\gamma) - \nabla(f+g)(\bar{\gamma})||_F \leq L||\gamma - \bar{\gamma}||_F. \tag{6.17}$$

Proof. We can prove this lemma by applying subadditivity of norms and additivity of

gradients:

$$\begin{aligned}
&||\nabla(f+g)(\gamma) - \nabla(f+g)(\bar{\gamma})||_F \\
=&||\nabla f(\gamma) + \nabla g(\gamma) - \nabla f(\bar{\gamma}) - \nabla g(\bar{\gamma})||_F \\
=&||\nabla f(\gamma) - \nabla f(\bar{\gamma}) + \nabla g(\gamma) - \nabla g(\bar{\gamma})||_F \\
\leq&||\nabla f(\gamma) - \nabla f(\bar{\gamma})||_F + ||\nabla g(\gamma) - \nabla g(\bar{\gamma})||_F \\
\leq&L_f||\gamma - \bar{\gamma}||_F + L_g||\gamma - \bar{\gamma}||_F \\
\leq&(L_f + L_g)||\gamma - \bar{\gamma}||_F
\end{aligned}$$

where γ and $\bar{\gamma}$ are any two points in the domain of $f(\cdot) + g(\cdot)$. Thus, $f(\cdot) + g(\cdot)$ is L-smooth where $L = (L_f + L_g)$ is the Lipschitz constant. □

Lemma 6. *By the definition of L-smooth function, $(f+g)(\gamma)$ in Eq. 6.15 satisfies the following inequality [178]:*

$$(f+g)(\bar{\gamma}) - (f+g)(\gamma) - \Big\langle (\bar{\gamma} - \gamma), \nabla(f+g)(\gamma) \Big\rangle_F \leq \frac{L}{2}||\bar{\gamma} - \gamma||_F^2. \tag{6.18}$$

Lemma 7. *By the definition of strongly convexity, $h(\gamma)$ in Eq. 6.15 satisfies the following inequality [178]:*

$$h(\bar{\gamma}) - h(\gamma) - \Big\langle (\bar{\gamma} - \gamma), \nabla h(\gamma) \Big\rangle_F \geq \frac{\sigma}{2}||\bar{\gamma} - \gamma||_F^2. \tag{6.19}$$

Lemma 8. *$H(\gamma)$ in Eq. 6.15 satisfies the following inequality:*

$$H(\bar{\gamma}) - H(\gamma) - \Big\langle (\bar{\gamma} - \gamma), \nabla H(\gamma) \Big\rangle_F \leq \frac{(L-\sigma)}{2}||\bar{\gamma} - \gamma||_F^2. \tag{6.20}$$

Proof. By Lemma 6 and Lemma 7, we know $(f+g)(\gamma)$ is L-smooth for a constant L and $h(\gamma)$ is σ-strongly convex for a constant σ ($0 < \sigma < L$), respectively. Thus, we can reformulate Eq. 6.20 as follows:

$$\begin{aligned}
&H(\bar{\gamma}) - H(\gamma) - \Big\langle (\bar{\gamma} - \gamma), \nabla H(\gamma) \Big\rangle_F \\
=&\Big((f+g)(\bar{\gamma}) - h(\bar{\gamma})\Big) - \Big((f+g)(\gamma) - h(\gamma)\Big) - \Big\langle (\bar{\gamma} - \gamma), \nabla(f+g)(\gamma) - \nabla h(\gamma) \Big\rangle_F \\
=&(f+g)(\bar{\gamma}) - (f+g)(\gamma) - \Big\langle (\bar{\gamma} - \gamma), \nabla(f+g)(\gamma) \Big\rangle_F \\
&- \Big(h(\bar{\gamma}) - h(\gamma) - \Big\langle (\bar{\gamma} - \gamma), \nabla h(\gamma) \Big\rangle_F\Big) \\
\leq&\frac{L}{2}||\bar{\gamma} - \gamma||_F^2 - \frac{\sigma}{2}||\bar{\gamma} - \gamma||_F^2 \\
\leq&\frac{(L-\sigma)}{2}||\bar{\gamma} - \gamma||_F^2.
\end{aligned}$$

□

In the following, we consider $\gamma := \gamma^i, \hat{\gamma} := \hat{\gamma}^i$ and $\bar{\gamma} := \gamma^{i+1} = \gamma + \alpha(\hat{\gamma} - \gamma)$ where γ^i, $\hat{\gamma}^i$ and γ^{i+1} refer to the ones being computed at each i-th iteration in Algorithm 2. Specifically, at each i-th iteration, our algorithm SCG moves from the current point γ^i along the descent direction for the length $\alpha(\hat{\gamma} - \gamma)$ (i.e., determined by the step size α and the difference between the linear minimizer $\hat{\gamma}^i$ and the current point γ^i) to obtain the next iterate point γ^{i+1}.

As $H(\cdot)$ is non-convex, we define a generalized curvature constant C_{f+g-h} to measure the non-linearity of $H(\cdot)$ over a convex and compact feasible set $\pi(\mu, \nu)$. If C_{f+g-h} is small, then $H(\cdot)$ is close to being linear; conversely, if C_{f+g-h} is getting larger, then $H(\cdot)$ gets more curvature. $C_{f+g-h} = 0$ when $H(\cdot)$ is linear.

Lemma 9. *Let C_{f+g-h} be a generalized curvature constant for $H(\gamma)$ defined as:*

$$\sup_{\gamma, \hat{\gamma} \in \pi(\mu,\nu)} \frac{2}{\alpha^2} \Big(H(\bar{\gamma}) - H(\gamma) - \Big\langle (\bar{\gamma} - \gamma), \nabla H(\gamma) \Big\rangle_F \Big),$$

where $\alpha \in [0, 1]$ and $\bar{\gamma} = \gamma + \alpha(\hat{\gamma} - \gamma)$. Then, we have

$$C_{f+g-h} \leq (L - \sigma) \cdot diam_{||.||}(\pi(\mu, \nu))^2, \tag{6.21}$$

where $diam_{||.||}(\pi(\mu, \nu))^2$ is the $||.||_F$-diameter of $\pi(\mu, \nu)$.

Proof. We can obtain the upper bound of C_{f+g-h} by applying Eq. 6.20 in Lemma 8 to the definition of C_{f+g-h}, i.e.,

$$\begin{aligned} C_{f+g-h} \leq & \sup_{\gamma, \hat{\gamma}, \bar{\gamma} \in \pi(\mu,\nu)} \frac{2}{\alpha^2} \cdot \frac{(L - \sigma)}{2} \cdot ||\bar{\gamma} - \gamma||_F^2 \\ = & \sup_{\gamma, \hat{\gamma} \in \pi(\mu,\nu)} (L - \sigma) \cdot ||\hat{\gamma} - \gamma||_F^2 \\ \leq & (L - \sigma) \cdot diam_{||.||}(\pi(\mu, \nu))^2. \end{aligned} \tag{6.22}$$

□

Suboptimality gap is a good criterion to measure the distance to a stationary point at each iteration [114]. Below, we define the suboptimality gap for SCG.

Definition 5 (Suboptimality gap)**.** *For each i-th iteration of SCG, the suboptimality gap δ_i is defined by*

$$\delta_i = \max_{\hat{\gamma} \in \pi(\mu,\nu)} \Big\langle (\gamma - \hat{\gamma}), \nabla H(\gamma) \Big\rangle_F. \tag{6.23}$$

We know that, by Lemma 3 $f(\gamma)$ is L-smooth, and by the results of [40] $g(\gamma)$ is also L-smooth. Thus, $f(\gamma) + g(\gamma)$ is L-smooth. Further, by Lemma 4, $h(\gamma)$ is strongly convex. Thus, we obtain a generalized curvature constant $C_{f+g-h} \leq (L - \sigma) \cdot diam_{||.||}(\pi(\mu, \nu))^2$ where $0 < \sigma < L$. By the results of Frank-Wolfe algorithm for non-convex functions [134], we have $\min_{0 \leq i \leq k} \delta_i \leq \frac{max\{2h_0, C_{f+g-h}\}}{\sqrt{k+1}}$, for $k \geq 0$. Hence, we obtain the following theorem.

Theorem 8 (Convergence). *SCG has the minimal suboptimality gap δ_i that satisfies the following condition:*

$$\min_{0\leq i\leq k} \delta_i \leq \frac{max\{2h_0, (L-\sigma)\cdot diam_{||.||}(\pi(\mu,\nu))^2\}}{\sqrt{k+1}} \tag{6.24}$$

where $\sigma = 1$, $h_0 = H(\gamma^0) - \min_{\gamma\in\pi(\mu,\nu)} H(\gamma)$ is the initial suboptimality gap, L is a Lipschitz constant of $\nabla(f+g)(\gamma)$, and $diam_{||.||}(\pi(\mu,\nu))^2$ denotes the $||.||_F$-diameter of the $\pi(\mu,\nu)$.

To simplify the notation, let $C = (L-\sigma)\cdot diam_{||.||}(\pi(\mu,\nu))^2$ be the upper bound of C_{f+g-h} as in Lemma 9. Then, we can prove the aforementioned theorem for the convergence of SCG.

Proof. Our proof follows the steps of the conditional gradient method [114] and Frank-Wolfe algorithm for non-convex functions [134]. Our SCG algorithm is based on the Armijo rule in line searches for selecting the step size α into the descent direction. According to [18], we know that, if α is selected by the Armijo rule, then every limit point of $\{\gamma^i\}$ is a stationary point. Further, by Lemma 9, there exists a finite curvature constant $C_{f+g-h} \leq C$ over the feasible set $\pi(\mu,\nu)$. Thus, we have the following formula by applying $\bar{\gamma} := \gamma + \alpha(\hat{\gamma}-\gamma)$ on Eq. 6.20 in Lemma 8

$$H(\bar{\gamma}) \leq H(\gamma) + \alpha\Big\langle(\hat{\gamma}-\gamma), \nabla H(\gamma)\Big\rangle_F + \frac{\alpha^2}{2}C, \tag{6.25}$$

where $\alpha \in [0,1]$. Then, according to Definition 5, we can rewrite Eq. 6.25 as

$$H(\bar{\gamma}) - H(\gamma) \leq -\alpha\delta_i + \frac{\alpha^2}{2}C. \tag{6.26}$$

Based on this, we derive the optimality condition of the step-size α by minimizing the RHS of the above inequality,

$$\min_{\alpha\in[0,1]} -\alpha\delta_i + \frac{\alpha^2}{2}C. \tag{6.27}$$

By derivating Eq. 6.27 w.r.t. the step-size α, we have the minimizer at α^* s.t.

$$\alpha^* = min\Big\{\frac{\delta_i}{C}, 1\Big\}. \tag{6.28}$$

Now we consider two different cases w.r.t. $\alpha \in [0,1]$ to examine the upper bounds for the RHS of Eq. 6.26.

Case 1: If $\alpha^* < 1$, this case implies $\delta_i < C$. Thus, we have $\alpha^* = \frac{\delta_i}{C}$ and

$$H(\bar{\gamma}) - H(\gamma) \leq -\frac{\delta_i^2}{2C}. \tag{6.29}$$

Case 2: If $\alpha^* = 1$, this case implies $C \leq \delta_i$. Thus, we have

$$H(\bar{\gamma}) - H(\gamma) \leq -\delta_i + \frac{C}{2} \leq -\frac{\delta_i}{2}. \tag{6.30}$$

We can combine these two cases into the following inequality which is valid for $i \leq k$,

$$H(\gamma^{i+1}) - H(\gamma^i) \leq -\frac{\delta_i}{2}\, min\Big\{\frac{\delta_i}{C}, 1\Big\}\mathbb{I}, \tag{6.31}$$

where $\gamma^{i+1} := \bar{\gamma}$ and $\gamma^i := \gamma$. The $\mathbb{I}$ denotes the indicator function which refers to considering both cases of $\alpha^* = min\Big\{\frac{\delta_i}{C}, 1\Big\}$.

By recursively applying Eq. 6.31 from $i = 0$ to k and rearranging the terms, we have

$$H(\gamma^0) - H(\gamma^{k+1}) \geq \sum_{i=0}^{k} \frac{\delta_i}{2}\, min\Big\{\frac{\delta_i}{C}, 1\Big\}\mathbb{I}. \tag{6.32}$$

Below, we derive the explicit convergence rate for SCG when converging to a stationary point. Let $\tilde{\delta_k} := \min_{0 \leq i \leq k} \delta_i$. Then, from Eq. 6.32, we have

$$\begin{aligned} H(\gamma^0) - H(\gamma^{k+1}) \geq & \sum_{i=0}^{k} \frac{\tilde{\delta_k}}{2}\, min\Big\{\frac{\tilde{\delta_k}}{C}, 1\Big\}\mathbb{I} \\ = & (k+1)\frac{\tilde{\delta_k}}{2}\, min\Big\{\frac{\tilde{\delta_k}}{C}, 1\Big\}. \end{aligned} \tag{6.33}$$

We consider each case in $min\{\cdot, \cdot\}$ separately below:

- If $\tilde{\delta_k} \leq C$, then Eq. 6.33 gives $H(\gamma^0) - H(\gamma^{k+1}) \geq \frac{(k+1)\tilde{\delta_k}^2}{2C}$, and we can reorder it as

$$\begin{aligned} \tilde{\delta_k} \leq & \sqrt{\frac{2C(H(\gamma^0) - H(\gamma^{k+1}))}{k+1}} \leq \sqrt{\frac{2Ch_0}{k+1}} \\ \leq & \frac{2h_0 + C}{2\sqrt{k+1}} \leq \frac{max\{2h_0, C\}}{\sqrt{k+1}}, \end{aligned} \tag{6.34}$$

  where $\sqrt{\frac{2Ch_0}{k+1}} \leq \frac{2h_0+C}{2\sqrt{k+1}}$, we use the inequality $\sqrt{ab} \leq \frac{a+b}{2}$ with $a = 2h_0$ and $b = C$. By the definition of the initial suboptimality h_0, we have $H(\gamma^0) - H(\gamma^{k+1}) \leq H(\gamma^0) - \min_{\gamma \in \pi(\mu,\nu)} H(\gamma) = h_0$.

- If $\tilde{\delta_k} > C$, we can obtain a better $\frac{1}{k}$ rate, which is trivially bounded by $\frac{1}{\sqrt{k}}$,

$$\tilde{\delta_k} \leq \frac{2h_0}{k+1} \leq \frac{2h_0}{\sqrt{k+1}} \leq \frac{max\{2h_0, C\}}{\sqrt{k+1}}. \tag{6.35}$$

Since we obtain the same upper bound in the above cases, Theorem 8 is proven,

i.e.,

$$\min_{0\leq i\leq k} \delta_i \leq \frac{max\{2h_0, C\}}{\sqrt{k+1}}. \tag{6.36}$$

□

Following Theorem 8, we have the corollary below.

Corollary 1. *For SCG, the minimal suboptimality gap is* $O(\frac{1}{\sqrt{k}})$ *after the number* k *of iterations. It takes at most* $O(\frac{1}{\epsilon^2})$ *iterations to find an approximate stationary point with a suboptimality gap smaller than* ϵ*.*

6.4.2 Complexity Analysis

A naive implementation of $\nabla f(\gamma)$ has the time complexity $O(N^4)$ due to the tensor-matrix multiplication in $GW(\mu, \nu)$, where $N = max\{n_1, n_2\}$. Nevertheless, as discussed in [197], for a general class of loss functions, the tensor-matrix multiplication can be decomposed into matrix-matrix multiplications and the time complexity of $\nabla f(\gamma)$ can thus be reduced to $O(N^3)$. $\nabla g(\gamma)$ has the time complexity $O(N^3)$. The time complexity of the line search algorithm depends on the computation of $H(\gamma)$ in Eq. 6.15. Since it has the time complexity $O(N^3 + N^2k^2)$, the total time complexity of our algorithm is $O(t(N^3 + N^2k^2))$, where t refers to the total number of iterations and k is the dimension of the node embedding. The memory complexity of our algorithm is $O(N^2)$.

Table 6.1 summarizes the time and memeory complexity of several optimal transport based graph kernels. Note that t is much smaller than N in practice.

Optimal Transport Based Graph Kernel	Time Complexity	Memory Complexity
WL-PM [182]	$O(N^3log(N))$	$O(N^2)$
WWL [245]	$O(N^3log(N))$	$O(N^2)$
FGW [243]	$O(t(N^3))$	$O(N^2)$
RWK (ours)	$O(t(N^3 + N^2k^2))$	$O(N^2)$

Table 6.1: A summary of time and memory complexities.

6.5 Experimental Setup

Our experiments are performed on a Linux server which has 12-core Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz, NVIDIA GeForce GTX Titan Xp with 96GB of main memory. We evaluate regularized wasserstein kernels (RWKs) on graph classification benchmark tasks against the state-of-the-art baselines in order to answer the following questions:

Q1. How well can RWK empirically perform for graph classification tasks?

Q2. What impact do feature local variations have on the performance of RWK?

Q3. How efficiently can RWK perform in comparison with the existing optimal-transport based graph kernels?

Q4. How does each of the key components in RWK (i.e., different distance metrics and regularization terms) contribute to the overall performance of RWK?

Below, we will present our experimental environment. Then, we will discuss the experimental results and answer these questions in Section 6.6.

6.5.1 Datasets

In our experiments, we consider 12 benchmark datasets, which generally fall into two categories:

(1) Graphs with discrete attributes: MUTAG, PTC-MR, NCI1, NCI109 and D&D are bioinformatics datasets [64, 275, 130, 226], and COLLAB is a social network [277] for which we use the same one-hot encoding setup as in [275]. These all datasets are equipped only discrete nodes attributes.

(2) Graphs with continuous attributes: COX2, COX2-MD, BZR, BZR-MD, PROTEINS and ENZYMES are bioinformatics datasets [239, 22, 245]. Specifically, COX2, BZR, PROTEINS and ENZYMES have both continuous nodes and edge attributes; COX2-MD and BZR-MD only contain continuous node attributes.

Table 6.2 provides further details about these datasets, including the availability of node and edge attributes, the number of graphs, and the number of classes.

Dataset	Node Attributes	Edge Attributes	#Classes	#Graphs
MUTAG	✓	-	2	188
PTC-MR	✓	-	2	344
NCI1	✓	-	2	4110
D & D	✓	-	2	1178
NCI109	✓	-	2	4127
COLLAB	✓	-	3	5000
ENZYMES	✓	✓	6	600
PROTEINS	✓	✓	2	1113
COX2	✓	✓	2	467
BZR	✓	✓	2	405
COX2-MD	✓	-	2	303
BZR-MD	✓	-	2	306

Table 6.2: Dataset statistics.

6.5.2 Baseline Methods

We evaluate the performance of RWK against the following 16 state-of-the-art baselines, divided into three groups:

- **Non-Optimal-Transport (Non-OT) graph kernels:** We compare RWK with several state-of-the-art traditional graph kernel methods. This baseline comparison includes WL subtree kernel (WL) [226], WL Optimal Assignment Kernel (WL-OA) [130], Graph Hopper Kernel (GHK) [75], Propagation Kernel (PK) [179], Hash Graph Kernel (HGK-WL; HGK-SP) [173], Return Probabilities of Random Walks Kernel (RetGK) [289], Graph Neural Tangent Kernel (GNTK) [68], and Persistent WL Kernel (P-WL) [208].
- **Optimal-Transport-based (OT-based) graph kernels:** We compare RWK with three state-of-the-art OT-based graph kernel methods such as WL Pyramid Match Kernel (WL-PM) [182], Wasserstein WL Graph Kernel (WWL) [245] and Fused-Gromov Wasserstein (FGW) [243].
- **Graph Neural Network (GNN-based) methods:** We compare RWK with four state-of-the-art GNNs methods. GNN-based baseline comparison includes PATCHY-SAN [181], Deep Graph Convolutional Neural Network (DGCNN) [286], Capsule Neural Network (CapsGNN) [271], and Graph Isomorphism Network (GIN) [275].

6.5.3 Hyperparameter Settings

To benchmark the baseline methods, we follow the work of Titouan et al. [243] and use the same setup and data splits. The hyperparameters of our method are selected using the nested cross validation [243]. C-SVM classifier is used with $C \in \{10^{-5}, 10^{-4}, \ldots, 10^{5}\}$. We choose the following parameter ranges: $\eta \in \{2^{-5}, 2^{-4}, \ldots, 2^{5}\}$, $\beta_1, \beta_2 \in \{0.1, 0.2, \ldots, 1\}$, $\lambda_\mu, \lambda_\nu, \lambda_g, \rho \in \{10^{-1}, 10^{-2}, \ldots, 10^{-5}\}$, $t \in \{5, 10\}$, $b \in \{10, 20, \ldots, 50\}$, $\lambda \in \{0.1, 0.2, \ldots, 0.9\}$, $\epsilon \in \{10^{-3}, 10^{-4}, \ldots, 10^{-9}\}$, and set $\alpha^{(0)} = 0.99$ as the initial value of step size. For graphs with discrete attributes, we define feature similarity matrices on the Weisfeiler-Lehman sequence of graphs [262]. For BZR-MD and COX2-MD, we follow the same approach in [245] to obtain node attributes. We consider the number of Weisfeiler-Lehman iterations $h \in \{1, 2\}$.

We choose the l_2 distance for d_f and hamming distance for d_s. For the dimension of node embeddings, we set $k = 64$. For feature local variation, we set $j = 2$ (2-hop) as the default setting for RWK. The number and length of random walks are selected from $\{10, 20, 30\}$ and $\{2, 3, 4, 5, 6, 7, 8\}$, respectively. We train the model of node embeddings using 200 epochs and select the best learning rate from $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$.

6.6 Results and Discussion

In this section, we discuss the experimental results to answer the aforementioned four questions. The results demonstrate the effectiveness of our method on real-

	Method	MUTAG	PTC-MR	NCI1	D&D	NCI109	COLLAB
Non-OT graph kernels	WL	90.4 ± 5.7	59.9 ± 4.3	86.0 ± 1.8	79.4 ± 0.3	85.9 ± 1.5	78.9 ± 1.9
	WL-OA	84.5 ± 1.7	63.6 ± 1.5	86.1 ± 0.2	79.2 ± 0.4	86.3 ± 0.2	80.7 ± 0.1
	RetGK	90.3 ± 1.1	62.5 ± 1.6	84.5 ± 0.2	-	-	81.0 ± 0.3
	GNTK	90.0 ± 8.5	67.9 ± 6.9	84.2 ± 1.5	75.6 ± 3.9	-	83.6 ± 1.0
	P-WL	90.5 ± 1.3	64.0 ± 0.8	85.4 ± 0.1	78.6 ± 0.3	84.9 ± 0.3	-
OT-based graph kernels	WL-PM	87.7 ± 0.8	61.4 ± 0.8	86.4 ± 0.2	78.6 ± 0.2	85.3 ± 0.2	81.5 ± 0.5
	WWL	87.2 ± 1.5	66.3 ± 1.2	85.7 ± 0.2	79.6 ± 0.5	-	-
	FGW	88.4 ± 5.6	65.3 ± 7.9	86.4 ± 1.6	-	-	-
GNN-based methods	PATCHY-SAN	92.6 ± 4.2	60.0 ± 4.8	78.6 ± 1.9	77.1 ± 2.4	-	72.6 ± 2.2
	DGCNN	85.8 ± 0.0	58.6 ± 0.0	74.4 ± 0.0	76.6 ± 0.0	75.0 ± 0.0	73.7 ± 0.0
	CapsGNN	86.6 ± 1.5	66.0 ± 1.8	78.3 ± 1.3	75.3 ± 2.3	81.1 ± 3.1	79.6 ± 2.9
	GIN	89.4 ± 5.6	64.6 ± 7.0	82.7 ± 1.7	75.3 ± 3.5	86.5 ± 1.5	80.2 ± 1.9
Our work	RWK	**93.6 ± 3.7**	**69.5 ± 6.1**	**88.0 ± 4.5**	**81.6 ± 3.5**	**87.3 ± 6.1**	**83.8 ± 4.6**
	RWK-1	92.5 ± 3.1	68.9 ± 5.1	87.7 ± 6.1	81.0 ± 4.3	86.9 ± 5.2	83.2 ± 3.1
	RWK-0	90.7 ± 4.2	67.8 ± 3.6	87.0 ± 5.1	79.6 ± 3.1	86.4 ± 4.6	81.5 ± 3.9

Table 6.3: Classification accuracy (%) averaged over 10 runs on graphs with discrete attributes. The results of WL and RetGK are taken from [68] and the results of the other baselines are from their original papers.

world graphs, i.e., considerably and consistently outperforming all the state-of-the-art methods on all benchmark datasets.

6.6.1 Graph Classification

We first benchmark the performance of our RWK method against the baselines on graph classification tasks to answer the question **Q1**. The results are reported in Tables 6.3-6.4. Table 6.3 and Table 6.4 present the classification accuracy with discrete attributes and continuous attributes, respectively. We ignore the GNN-based methods in Table 6.4 since GNNs are not performed well on graphs with continuous attributes.

We see that, in Table 6.3, compared with the non-OT graph kernels, RWK improves upon their best results by a margin ranging from 0.2% to 3.1% on all datasets. Similarly, RWK improves upon the best results of the OT-based graph kernels by a margin ranging from 1.6% to 5.2% on all datasets, and upon the best results of the GNN-based baselines by a margin ranging from 0.8% to 5.3%.

In Table 6.4, RWK also consistently performs better than all the baselines on all graphs with continuous attributes. Specifically, RWK improves upon the best results of the non-OT graph kernels by a margin ranging from 2.8% to 6.7% and the best results of the OT-based graph kernels by a margin ranging from 1.1% to 5.1% across the datasets.

It is worthy to mention that none of the baselines have achieved the best performance on all datasets, in comparison with the other baselines. However, in contrast, RWK consistently performs best on all datasets. Specifically, RWK improves upon the best results of the baselines by a margin of 1.0% (PATCHY-SAN), 1.6% (GNTK), 1.6% (FGW), 2.0% (WWL), 0.8% (GIN), and 0.2% (GNTK) on the datasets MUTAG, PTC-MR, NCI1, D&D, NCI109 and COLLAB, respectively. A similar situation exists for graphs with continuous attributes.

	Method	COX2	ENZYMES	PROTEINS	BZR	COX2-MD	BZR-MD
Non-OT graph kernels	GHK	76.4 ± 1.3	65.6 ± 0.8	74.7 ± 0.2	76.4 ± 0.9	66.2 ± 1.0	69.1 ± 2.0
	PK	77.6 ± 0.6	71.6 ± 0.5	61.3 ± 0.8	79.5 ± 0.4	-	-
	HGK-WL	78.1 ± 0.4	63.0 ± 0.6	75.9 ± 0.1	78.5 ± 0.6	74.6 ± 1.7	68.9 ± 0.6
	HGK-SP	72.5 ± 1.1	66.3 ± 0.3	75.7 ± 0.1	76.4 ± 0.7	68.5 ± 1.0	66.1 ± 1.0
OT-based graph kernels	WWL	78.2 ± 0.4	73.2 ± 0.8	77.9 ± 0.8	84.4 ± 2.0	76.3 ± 1.0	69.7 ± 0.9
	FGW	77.2 ± 4.8	71.0 ± 6.7	74.5 ± 2.7	85.1 ± 4.1	-	-
Our work	RWK	**81.2 ± 5.3**	**78.3 ± 4.1**	**79.3 ± 6.1**	**86.2 ± 5.6**	**78.1 ± 4.3**	**71.9 ± 4.6**
	RWK-1	80.7 ± 4.6	77.5 ± 5.3	78.9 ± 4.5	85.8 ± 5.5	77.4 ± 3.7	71.3 ± 4.3
	RWK-0	79.6 ± 3.1	76.4 ± 4.5	78.2 ± 5.6	85.2 ± 4.3	76.7 ± 5.5	70.5 ± 3.7

Table 6.4: Classification accuracy (%) averaged over 10 runs on graphs with continuous attributes. The results of GHK, HGK-WL and HGK-SP are taken from [245] and the results of the other baselines are from their original papers.

6.6.2 Impact of Local Variations

In our experiments, we also notice that performance increases with 2-hops rather than the 1-hop, however when we increase the number of hops (hops > 2) does not necessarily lead to improved performance due to the issue of oversmoothing. We thus restrict feature local variations within 2 hops. To analyze the impact of feature local variations to answer the question **Q2**, we compare the performance of RWK that uses 2-hop feature local variations against the following two additional settings:

- **RWK-0:** without using any feature local variations;
- **RWK-1:** with using 1-hop feature local variations.

The results for this experiment are presented in Tables 6.3-6.4. We can see the following. First, feature local variations help to improve the performance considerably and consistently on all datasets, including both graphs with discrete attributes and graphs with continuous attributes. Second, on all these datasets, RWK consistently performs better than RWK-1, and RWK-1 consistently performs better than RWK-0.

6.6.3 Runtime Analysis

We evaluate the running time of RWK against the other OT-based graph kernel methods, i.e., FGW [243] and WWL [245] to answer the question **Q3**. All these methods and our method were implemented in python. For a fair comparison, we do not consider WL-PM [182] because its implementation was done using MATLAB. The runtime results are averaged over 10 runs.

Figure 6.3 shows the results. We see that: (1) FGW is the fastest one, compared with WWL and RWK, over all benchmark datasets; (2) RWK is slower than WWL on the 4 small datasets but faster than WWL on the other 7 larger datasets. This demonstrates the good scalability of RWK for large datasets. The reason why RWK is more scalable than WWL is as follows. WWL considers an unregularized Wasserstein optimization problem, which is usually cast as a linear programming problem and costly to solve [58]. In its algorithm implementation, WWL uses the EMD solver

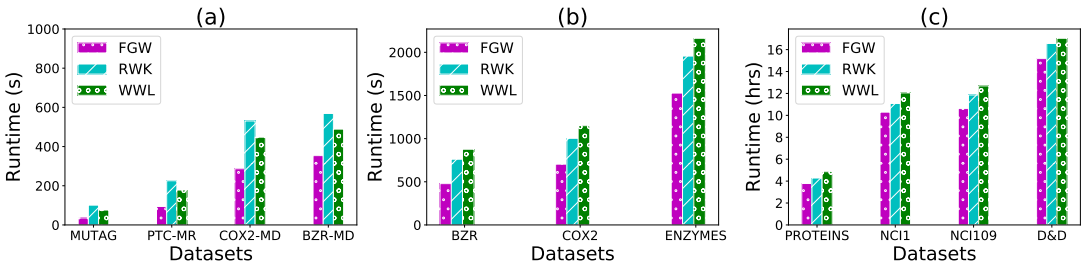


Figure 6.3: Running time averaged over 10 runs on graphs with discrete and continuous attributes. There are no result for the COLLAB dataset because all methods take more than 24 hours to obtain the results

[58]. Different from WWL, RWK considers a regularized optimal transport problem, which is solved by our SCG algorithm being designed upon the *Sinkhorn-knopp* matrix scaling for speeding up the computation.

6.6.4 Ablation Analysis

To demonstrate the effectiveness of each component in the proposed method RWK, we conduct an ablation study on the following variants to answer the question **Q4**:

- **NoLaplacianReg:** This variant removes only the Laplacian regularization term $\Theta_w(\gamma)$ from RWK;
- **NoEntropyReg:** This variant removes only the degree-entropy regularization term $\Theta_g(\gamma)$ from RWK;
- **NoRegs:** This variant removes both regularization terms $\Theta_w(\gamma)$ and $\Theta_g(\gamma)$ from RWK;
- **RWK-LW:** This variant removes only the global connectivity Wasserstein distance $GW(\mu, \nu)$ from RWK;
- **RWK-GW:** This variant removes only the local barycentric Wasserstein distance $LW(\mu, \nu)$ from RWK.

The results are presented in Tables 6.5-6.6. We observe that both local barycentric Wasserstein distance and global connectivity Wasserstein distance are crucial to the performance. The regularization terms $\Theta_g(\gamma)$ and $\Theta_w(\gamma)$ help reduce the performance variance while boosting the performance. Specifically, on graphs with discrete attributes, compared with RWK, the performance decreases by a margin ranging from 1.5% to 3.5% in NoLaplacianReg, from 0.7% to 1.4% in NoEntropyReg, and from 2.6% to 4.7% in NoRegs. A similar trend exists on graphs with continuous attributes, where the performance decreases by a margin ranging from 1.4% to 3.2% in NoLaplacianReg, from 0.5% to 2.1% in NoEntropyReg, and from 2.2% to 4.0% in NoRegs. For RWK-LW,

compared with RWK, the performance decreases by a margin ranging from 3.1% to 6.2% on graphs with discrete attributes and from 3.4% to 5.1% on graphs with continuous attributes. Similarly, for RWK-GW, the performance decreases by a margin ranging from 6.1% to 10.8% on graphs with discrete attributes and from 5.7% to 7.8% on graphs with continuous attributes.

Variants	MUTAG	PTC-MR	NCI1	D&D	NCI109	COLLAB
NoLaplacianReg	90.1 ± 3.5	67.0 ± 3.7	86.2 ± 5.3	79.4 ± 4.5	85.8 ± 5.2	81.5 ± 3.9
NoEntropyReg	92.2 ± 3.5	68.3 ± 6.5	87.3 ± 6.1	80.4 ± 3.6	86.5 ± 4.7	82.4 ± 3.8
NoRegs	88.9 ± 3.5	66.2 ± 4.6	85.3 ± 5.8	78.2 ± 3.9	84.7 ± 5.1	80.8 ± 4.1
RWK-LW	87.4 ± 4.2	64.8 ± 6.5	84.9 ± 3.6	77.8 ± 3.8	83.8 ± 5.7	79.5 ± 3.6
RWK-GW	82.8 ± 5.4	61.2 ± 5.8	81.9 ± 4.3	75.3 ± 4.8	80.7 ± 5.5	75.1 ± 3.9

Table 6.5: Classification accuracy (%) averaged over 10 runs on graphs with discrete attributes.

Variants	COX2	BZR	ENZYMES	PROTEINS	COX2-MD	BZR-MD
NoLaplacianReg	79.1 ± 3.9	84.8 ± 4.2	76.2 ± 3.8	77.5 ± 5.5	76.1 ± 4.6	68.7 ± 3.9
NoEntropyReg	80.5 ± 5.4	85.7 ± 6.3	77.2 ± 3.7	78.5 ± 5.1	77.2 ± 4.1	69.8 ± 4.9
NoRegs	78.2 ± 4.6	83.7 ± 5.6	75.4 ± 3.6	76.6 ± 4.8	75.9 ± 3.6	67.9 ± 4.5
RWK-LW	77.1 ± 4.1	82.8 ± 3.8	74.5 ± 5.2	75.5 ± 4.4	74.7 ± 4.3	66.8 ± 5.1
RWK-GW	75.3 ± 5.4	79.6 ± 6.0	72.6 ± 3.3	73.2 ± 5.6	71.3 ± 4.1	64.1 ± 3.6

Table 6.6: Classification accuracy (%) averaged over 10 runs on graphs with continuous attributes.

6.7 Summary

In this chapter, we have proposed a learning framework for graph kernels, which is theoretically grounded on regularizing optimal transport*. This framework provides a novel optimal transport distance metric, namely *Regularized Wasserstein (RW) discrepancy*, which can preserve both features and structure of graphs via Wasserstein distances on features and their local variations, local barycenters and global connectivity. Two *strongly convex* regularization terms are introduced to improve the learning ability. One is to relax an optimal alignment between graphs to be a cluster-to-cluster mapping between their locally connected vertices, thereby preserving the local clustering structure of graphs. The other is to take into account node degree distributions in order to better preserve the global structure of graphs. We also designed an efficient algorithm to enable a fast approximation for solving the optimization problem. Theoretically, our framework is robust and can guarantee the convergence and numerical stability in optimization. We have empirically validated our method using 12 datasets against 16 state-of-the-art baselines. The experimental results showed that our method consistently outperforms all state-of-the-art methods on all benchmark databases for both graphs with discrete attributes and graphs with continuous attributes.

*RWK implementation can be found at: https://github.com/wokas36/RWK

Chapter 7

Conclusions and Future Work

In this thesis, we have studied two fundamental paradigms in graph learning: (1) graph representation learning and (2) graph similarity learning. We first explored three different approaches to develop expressive, yet simple and efficient GNNs for graph representation learning. Then, we introduced a graph kernel function that can incorporate intrinsic graph properties into a learning framework with guaranteed convergence and numerical stability. In summary, we have presented the following main results in Chapters 3-6:

- In Chapter 3, we have introduced a new perspective of designing powerful spatial GNNs that can go beyond the 1-WL test with a theoretically provable guarantee. In particular, we have developed a generalized message-passing framework, which allows to incorporate structural properties of graphs into a message-passing aggregation scheme. Theoretically, we have developed a new local isomorphism hierarchy based on neighborhood subgraphs. Then, we have proposed a novel neural network model, namely *GraphSNN*, and proved that this message-passing GNN is strictly more expressive than the 1-WL test in distinguishing graph structures. We have empirically evaluated our model on different benchmark tasks. The experimental results show that our model improves the baseline methods on different graph learning tasks without sacrificing the simplicity and efficiency.

- In Chapter 4, we have proposed a powerful, yet efficient spectral GNN model on graphs, called *Distributed Feedback-looped Networks (DFNets)*. This model is equipped with a new class of spectral graph filters, namely called *feedback-looped filters*, which provides a better localization on vertices. We have also formulated a convex constrained least square problem to learn optimal coefficients of the proposed filters. This model can theoretically guarantee the convergence with linear memory requirements. Furthermore, DFNets can provide strong gradient flows since the propagation rule of this model can diversify the features. We have empirically evaluated our model with two benchmark tasks: semi-supervised document classification with citation networks and semi-supervised entity-classification on knowledge graph. It is shown that our model considerably outperforms the baseline methods over all benchmark tasks.

- In Chapter 5, we have introduced a diffusion GNN architecture, called *Dynamic PageRank Networks* (DPRNs). We have further developed two variants of this GNN architecture: *forward-euler solution* and *invariable feature solution*. These two variants incorporate dynamic PageRank techniques to capture rich and varying graph structures. In order to improve the discriminative power of PageRank diffusion on graphs, we have also encoded local topological information into a learnable PageRank transition matrix through a quadratic convex constrained optimization problem to learn polynomial filter coefficients. Despite the fact that dynamic PageRank does not converge, we have theoretically proved that our GNN architecture with forward-euler solution is guaranteed to converge to a stationary distribution. We have also showed that a shallow GNN architecture with deeper single layers is a promising direction to capture different graph structures, i.e., homophilic and heterophilic graphs.

- In Chapter 6, we have proposed a regularized optimal transport-based framework for graph kernels. This framework introduces a new optimal transport distance metric, called *Regularized Wasserstein (RW) discrepancy*, which can preserve both local and global structural information, and feature local variations between graphs during the transportation. Furthermore, to find an optimal assignment between two graphs, we have proposed two *strongly convex* regularization terms to theoretically guarantee the convergence and numerical stability. One regularization term is used to relax the optimal assignment between graphs to preserve local clustering structures between locally connected vertices. The other one is to consider node degree distributions to preserve global structures of graphs. We have designed an efficient and a numerically stable algorithm for solving the optimization problem. We have empirically evaluated our method over two benchmark tasks: graph classification with discrete attributed graphs and graph classification with continuous attributed graphs. It is shown that this method outperforms all baseline methods on all benchmark tasks.

To conclude, we have developed powerful GNN models for graph representation learning from three different perspectives and a graph kernel approach for graph similarity learning, which allow us to take the advantage of rich structure of graphs for better prediction. Nonetheless, our work still has limitations and further work can be extended to address these limitations in several directions.

One limitation of our message-passing GNN model presented in Chapter 3 is that structural coefficients are pre-computed. In the literature, attention-based GNNs [284, 25] are typically considered as a promising direction to learn weights adaptively so as to exploit rich structural information. Following this direction, we can further develop a message-passing aggregation scheme for spatial GNN models with adaptively learnable structural coefficients to alleviate this limitation.

Similarly, desired frequency responses of DFNets in Chapter 4 and DPRNs in Chapter 5 are also restricted to predefined conditions, for which we need to select cut-off thresholds based on trial-and-error. In future, we plan to learn cut-off thresholds adaptively. Another limitation in DPRNs is that our current filter design is homoge-

neous isotropic, which restricts the expressive power of diffusion GNNs. Thus, we plan to incorporate anisotropic diffusion filters to enhance the expressive power of GNNs. In addition to these, we also plan to extend our spectral GNNs to time-varying graph structures. As discussed in [113], feedback-looped graph filters are practically appealing for time-varying settings, and similar to static graphs, some nice properties would likely hold for graphs that are a function of time.

Inspired by the success of deep generative models such as variational autoencoders (VAEs) [121] and generative adversarial networks (GANs) [89], graph generative models have been widely studied in the past several years [230, 88, 185, 222, 133, 62, 279]. Despite the considerable progress made by existing graph generative models, they cannot jointly leverage local clustering structures and global connectivity structures into a generative model. Therefore, in future works, we can extend our graph kernel framework developed in Chapter 6 to optimal transport-based graph generative models for preserving both global and local structures of generated graphs. Another limitation in RWK is that our current graph kernel design is required a higher running time for large graphs. Thus, we plan to scale-up our kernel method for large graphs. In addition to these, we also plan to extend our graph kernel on different graph structured data such as heterophilic graphs.

Bibliography

1. S. Abu-El-Haija, B. Perozzi, R. Al-Rfou, and A. A. Alemi. Watch your step: Learning node embeddings via graph attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. (cited on pages 97, 98, and 99)

2. S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. Ver Steeg, and A. Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning (ICML)*, pages 21–29. PMLR, 2019. (cited on pages 83 and 84)

3. A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola. Distributed large-scale natural graph factorization. In *International Conference on World Wide Web (WWW)*, pages 37–48, 2013. (cited on page 3)

4. D. Alvarez-Melis and T. Jaakkola. Gromov-wasserstein alignment of word embedding spaces. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1881–1890, 2018. (cited on pages 96 and 102)

5. M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning (ICML)*, pages 214–223. PMLR, 2017. (cited on page 6)

6. J. Atwood and D. Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1993–2001, 2016. (cited on page 20)

7. H. Avron and L. Horesh. Community detection using time-dependent personalized pagerank. In *International Conference on Machine Learning (ICML)*, pages 1795–1803, 2015. (cited on pages 74 and 78)

8. W. Azizian and M. Lelarge. Expressive power of invariant and equivariant graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2020. (cited on page 16)

9. L. Babai and L. Kucera. Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science (SFCS)*, pages 39–46, 1979. (cited on page 15)

10. A. T. Balaban. Applications of graph theory in chemistry. *Journal of Chemical Information and Computer Sciences*, pages 334–343, 1985. (cited on page 1)

11. I. I. Baskin, V. A. Palyulin, and N. S. Zefirov. A neural device for searching direct correlations between structures and properties of chemical compounds. *Journal of Chemical Information and Computer Sciences*, pages 715–721, 1997. (cited on page 4)

12. D. Beaini, S. Passaro, V. Létourneau, W. Hamilton, G. Corso, and P. Liò. Directional graph networks. In *International Conference on Machine Learning (ICML)*, pages 748–758, 2021. (cited on pages 9 and 22)

13. M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in Neural Information Processing Systems (NeurIPS)*, 14, 2001. (cited on page 3)

14. Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013. (cited on page 2)

15. J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research (JMLR)*, 13(2), 2012. (cited on pages 46 and 63)

16. S. Berretti, A. Del Bimbo, and E. Vicario. Efficient matching and indexing of graph models in content-based retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1089–1105, 2001. (cited on page 6)

17. M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Annual Conference on Computer Graphics and Interactive Techniques*, pages 417–424, 2000. (cited on page 20)

18. D. P. Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997. (cited on page 107)

19. S. Bhagat, G. Cormode, and S. Muthukrishnan. Node classification in social networks. In *Social Network Data Analytics*, pages 115–148. Springer, 2011. (cited on page 2)

20. F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi. Graph neural networks with convolutional ARMA filters. *arXiv preprint arXiv:1901.01343*, 2019. (cited on pages 5, 9, 17, 18, 19, 57, 62, 63, 65, and 83)

21. K. M. Borgwardt. *Graph kernels*. PhD thesis, lmu, 2007. (cited on page 5)

22. K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *IEEE International Conference on Data Mining (ICDM)*, pages 8–pp. IEEE, 2005. (cited on pages 6, 22, 24, 44, and 110)

23. G. Bouritsas, F. Frasca, S. Zafeiriou, and M. M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv preprint arXiv:2006.09252*, 2020. (cited on pages 4, 16, 45, 47, and 55)

24. X. Bresson and T. Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017. (cited on page 15)

25. S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2021. (cited on page 118)

26. M. M. Bronstein and I. Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1704–1711. IEEE, 2010. (cited on page 20)

27. M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34 (4):18–42, 2017. (cited on pages 2 and 5)

28. J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and deep locally connected networks on graphs. In *International Conference on Learning Representations (ICLR)*, 2014. (cited on pages 5, 17, and 57)

29. A. Buades, B. Coll, and J.-M. Morel. A non-local algorithm for image denoising. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 60–65. IEEE, 2005. (cited on page 20)

30. H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, pages 245–253, 1983. (cited on page 6)

31. H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*, pages 255–259, 1998. (cited on page 6)

32. H. Cai, V. W. Zheng, and K. C.-C. Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, pages 1616–1637, 2018. (cited on pages 1 and 2)

33. J.-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992. (cited on page 16)

34. S. Cai, L. Li, J. Deng, B. Zhang, Z.-J. Zha, L. Su, and Q. Huang. Rethinking graph neural architecture search from message-passing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6657–6666, 2021. (cited on page 15)

35. S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *ACM International on Conference on Information and Knowledge Management (CIKM)*, pages 891–900, 2015. (cited on page 3)

36. A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2010. (cited on pages 43 and 63)

37. V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *International Journal of Computer Vision*, pages 61–79, 1997. (cited on page 20)

38. B. P. Chamberlain, J. Rowbottom, M. Gorinova, S. Webb, E. Rossi, and M. M. Bronstein. Grand: Graph neural diffusion. In *International Conference on Machine Learning (ICML)*, 2021. (cited on pages 9, 20, 21, 71, and 84)

39. T. F. Chan and L. A. Vese. Active contours without edges. *IEEE Transactions on Image Processing*, pages 266–277, 2001. (cited on page 20)

40. L. Chapel, M. Alaya, and G. Gasso. Partial optimal transport with applications on positive-unlabeled learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. (cited on page 106)

41. K. Chaudhuri, S. M. Kakade, K. Livescu, and K. Sridharan. Multi-view clustering via canonical correlation analysis. In *International Conference on Machine Learning (ICML)*, pages 129–136, 2009. (cited on page 2)

42. D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 3438–3445, 2020. (cited on page 32)

43. J. Chen, T. Ma, and C. Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations (ICLR)*, 2018. (cited on page 83)

44. M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning (ICML)*, pages 1725–1735, 2020. (cited on page 71)

45. M. X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, M. Schuster, N. Shazeer, N. Parmar, et al. The best of both worlds: Combining recent advances in neural machine translation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 76–86, 2018. (cited on page 2)

46. R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018. (cited on page 20)

47. Y. Chen, M. R. Gupta, and B. Recht. Learning kernels from indefinite similarities. In *International Conference on Machine Learning (ICML)*, 2009. (cited on page 23)

48. Z. Chen, L. Chen, S. Villar, and J. Bruna. Can graph neural networks count substructures? *Advances in neural information processing systems (NeurIPS)*, 2020. (cited on pages 17 and 45)

49. E. Chien, J. Peng, P. Li, and O. Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations (ICLR)*, 2021. (cited on pages xxii, 5, 20, 71, 77, 79, 82, 84, 85, and 89)

50. K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation*, page 103, 2014. (cited on page 2)

51. K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014. (cited on page 2)

52. S. Choudhury, K. Agarwal, S. Purohit, B. Zhang, M. Pirrung, W. Smith, and M. Thomas. Nous: Construction and querying of dynamic knowledge graphs. In *IEEE International Conference on Data Engineering (ICDE)*, pages 1563–1565. IEEE, 2017. (cited on page 2)

53. F. R. Chung and F. C. Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997. (cited on pages 5, 9, 17, 18, and 57)

54. G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. (cited on page 45)

55. C. Cortes, M. Mohri, and A. Rostamizadeh. L2 regularization for learning kernels. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 109–116, 2009. (cited on page 61)

56. M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho. Lagrangian neural networks. *arXiv preprint arXiv:2003.04630*, 2020. (cited on page 20)

57. P. Cui, X. Wang, J. Pei, and W. Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, pages 833–852, 2018. (cited on pages 2 and 4)

58. M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems (NeurIPS)*, pages 2292–2300, 2013. (cited on pages 27, 113, and 114)

59. G. Da San Martino, N. Navarin, and A. Sperduti. A tree-based kernel for graphs. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 975–986. SIAM, 2012. (cited on page 24)

60. H. Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning (ICML)*, pages 2702–2711. PMLR, 2016. (cited on page 5)

61. V. S. Dave, M. A. Hasan, B. Zhang, and C. K. Reddy. Predicting interval time for reciprocal link creation using survival analysis. *Social Network Analysis and Mining*, pages 1–20, 2018. (cited on page 2)

62. N. De Cao and T. Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018. (cited on page 119)

63. P. de Haan, T. Cohen, and M. Welling. Natural graph networks. *arXiv preprint arXiv:2007.08349*, 2020. (cited on pages 4 and 14)

64. A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991. (cited on pages 44 and 110)

65. M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems (NeurIPS)*, pages 3844–3852, 2016. (cited on pages 5, 8, 17, 18, 57, 60, 62, 63, 65, and 68)

66. R. Dijkman, M. Dumas, and L. García-Bañuelos. Graph matching algorithms for business process model similarity search. In *International Conference on Business Process Management*, pages 48–63. Springer, 2009. (cited on page 6)

67. S. C. Douglas, S.-i. Amari, and S.-Y. Kung. On gradient adaptation with unit-norm constraints. *IEEE Transactions on Signal processing*, 48(6):1843–1847, 2000. (cited on page 61)

68. S. S. Du, K. Hou, B. Póczos, R. Salakhutdinov, R. Wang, and K. Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *arXiv preprint arXiv:1905.13192*, 2019. (cited on pages xxi, xxiii, 45, 49, 111, and 112)

69. E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019. (cited on page 20)

70. F. Durand and J. Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *Annual Conference on Computer Graphics and Interactive Techniques*, pages 257–266, 2002. (cited on page 20)

71. D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems (NeurIPS)*, pages 2224–2232, 2015. (cited on page 5)

72. M. Eliasof, E. Haber, and E. Treister. Pde-gcn: Novel architectures for graph neural networks motivated by partial differential equations. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021. (cited on pages 5, 20, 71, and 84)

73. F. Errica, M. Podda, D. Bacciu, and A. Micheli. A fair comparison of graph neural networks for graph classification. 2020. (cited on pages 47 and 50)

74. W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference (WWW)*, pages 417–426, 2019. (cited on page 1)

75. A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt. Scalable kernels for graphs with continuous attributes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2013. (cited on pages 24 and 111)

76. S. Ferradans, N. Papadakis, J. Rabin, G. Peyré, and J.-F. Aujol. Regularized discrete optimal transport. In *International Conference on Scale Space and Variational Methods in Computer Vision (SSVM)*, 2013. (cited on page 99)

77. S. Ferradans, N. Papadakis, G. Peyré, and J.-F. Aujol. Regularized discrete optimal transport. *SIAM Journal on Imaging Sciences*, pages 1853–1882, 2014. (cited on page 6)

78. J. H. Fitschen, F. Laus, and B. Schmitzer. Optimal transport for manifold-valued images. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 460–472. Springer, 2017. (cited on page 6)

79. R. Flamary, N. Courty, A. Rakotomamonjy, and D. Tuia. Optimal transport with laplacian regularization. 2014. (cited on page 99)

80. N. Fournier and A. Guillin. On the rate of convergence in wasserstein distance of the empirical measure. *Probability Theory and Related Fields*, 162(3-4):707–738, 2015. (cited on page 99)

81. H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell. Optimal assignment kernels for attributed molecular graphs. In *International Conference on Machine Learning (ICML)*, pages 225–232, 2005. (cited on page 24)

82. C. Gallicchio and A. Micheli. Graph echo state networks. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010. (cited on pages 4 and 13)

83. V. Garg, S. Jegelka, and T. Jaakkola. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning (ICML)*, pages 3419–3430, 2020. (cited on page 15)

84. T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*, pages 129–143. 2003. (cited on page 24)

85. J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, pages 1263–1272, 2017. (cited on page 13)

86. D. F. Gleich. Pagerank beyond the web. *Siam Review*, pages 321–363, 2015. (cited on page 20)

87. X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AIStats)*, pages 249–256, 2010. (cited on page 61)

88. R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, pages 268–276, 2018. (cited on page 119)

89. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. (cited on page 119)

90. M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 729–734, 2005. (cited on pages 4 and 13)

91. P. Goyal and E. Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, pages 78–94, 2018. (cited on pages 1 and 4)

92. D. Greene and P. Cunningham. A matrix factorization approach for integrating multiple data views. In *Joint European conference on Machine Learning and Knowledge Discovery in Databases*, pages 423–438. Springer, 2009. (cited on page 3)

93. M. Grohe. *Descriptive complexity, canonisation, and definable graph structure theory*, volume 47. Cambridge University Press, 2017. (cited on pages 15 and 16)

94. J. L. Gross, J. Yellen, and M. Anderson. *Graph theory and its applications*. Chapman and Hall/CRC, 2018. (cited on page 1)

95. A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 855–864, 2016. (cited on page 3)

96. S. Gu and Y. Guo. Learning svm classifiers with indefinite kernels. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 26, 2012. (cited on page 23)

97. M. Guillaumin, J. Verbeek, and C. Schmid. Is that you? metric learning approaches for face identification. In *International Conference on Computer Vision (ICCV)*, pages 498–505. IEEE, 2009. (cited on page 5)

98. N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, pages 217–288, 2011. (cited on page 76)

99. W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1024–1034, 2017. (cited on pages 4, 5, 14, 15, 16, 37, 38, 42, 45, and 84)

100. W. L. Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020. (cited on page 8)

101. D. K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011. (cited on pages 17 and 18)

102. D. Haussler. Convolution kernels on discrete structures. Technical report, Technical report, Department of Computer Science, University of California ..., 1999. (cited on pages 22 and 23)

103. M. He, Z. Wei, H. Xu, et al. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021. (cited on pages 71 and 84)

104. M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015. (cited on pages 17 and 57)

105. S. Hido and H. Kashima. A linear-time graph kernel. In *2009 Ninth IEEE International Conference on Data Mining*, pages 179–188. IEEE, 2009. (cited on page 24)

106. T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171–1220, 2008. (cited on page 22)

107. T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD)*, 2004. (cited on pages 5, 6, 22, and 24)

108. W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. (cited on pages xxi, 32, 43, 44, 45, 46, 48, 52, 82, and 84)

109. G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *IEEE conference on computer vision and pattern recognition (CVPR)*, pages 4700–4708, 2017. (cited on pages 58 and 61)

110. N. T. Huang and S. Villar. A short tutorial on the weisfeiler-lehman test and its variants. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8533–8537. IEEE, 2021. (cited on page 16)

111. K. Ishiguro, S.-i. Maeda, and M. Koyama. Graph warp module: an auxiliary module for boosting the power of graph neural networks in molecular graph analysis. *arXiv preprint arXiv:1902.01020*, 2019. (cited on page 48)

112. E. Isufi, A. Loukas, and G. Leus. Autoregressive moving average graph filters: a stable distributed implementation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4119–4123, 2017. (cited on pages 59 and 62)

113. E. Isufi, A. Loukas, A. Simonetto, and G. Leus. Autoregressive moving average graph filtering. *IEEE Transactions on Signal Processing*, 65(2):274–288, 2017. (cited on pages 19, 58, 59, 62, and 119)

114. M. Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *International Conference on Machine Learning (ICML)*, 2013. (cited on pages 103, 106, and 107)

115. N. Jiang, W. Liu, and Y. Wu. Order determination and sparsity-regularized metric learning adaptive visual tracking. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1956–1963. IEEE, 2012. (cited on page 5)

116. F. D. Johansson and D. Dubhashi. Learning with similarity functions on graphs using matchings of geometric embeddings. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 467–476, 2015. (cited on page 25)

117. L. Kantorovich. On the transfer of masses (in russian). In *Doklady Akademii Nauk*, volume 37, pages 227–229, 1942. (cited on page 25)

118. S. M. Kay and A. K. Shaw. Frequency estimation by principal component ar spectral estimation method without eigendecomposition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(1):95–101, 1988. (cited on page 8)

119. N. Keriven and G. Peyré. Universal invariant and equivariant graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. (cited on page 15)

120. D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. (cited on pages 46, 64, 84, and 85)

121. D. P. Kingma and M. Welling. Auto-encoding variational bayes. 2014. (cited on page 119)

122. T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017. (cited on pages 5, 13, 15, 17, 18, 20, 22, 31, 37, 38, 42, 45, 46, 57, 63, 64, 65, 77, 78, 82, 83, and 84)

123. V. Klema and A. Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on Automatic Control*, 25(2):164–176, 1980. (cited on page 98)

124. J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations (ICLR)*, 2019. (cited on pages 20, 71, 73, 77, 79, 83, and 84)

125. J. Klicpera, S. Weißenberger, and S. Günnemann. Diffusion improves graph learning. *Advances in Neural Information Processing Systems (NeurIPS)*, pages 13354–13366, 2019. (cited on pages 5, 20, 71, 73, 77, 78, 79, and 83)

126. P. A. Knight. The sinkhorn–knopp algorithm: convergence and applications. *SIAM Journal on Matrix Analysis and Applications (SIMAX)*, 30(1):261–275, 2008. (cited on page 96)

127. M. Koda, A. H. Dogru, and J. H. Seinfeld. Sensitivity analysis of partial differential equations with application to reaction and diffusion processes. *Journal of Computational Physics*, pages 259–282, 1979. (cited on page 20)

128. R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In *International Conference on Machine Learning (ICML)*, pages 315–322, 2002. (cited on pages 5 and 73)

129. N. Kriege and P. Mutzel. Subgraph matching kernels for attributed graphs. In *International Conference on Machine Learning (ICML)*, pages 291–298, 2012. (cited on page 24)

130. N. M. Kriege, P.-L. Giscard, and R. Wilson. On valid optimal assignment kernels and applications to graph classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1623–1631, 2016. (cited on pages 23, 25, 44, 110, and 111)

131. N. M. Kriege, F. D. Johansson, and C. Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020. (cited on pages 1, 22, 23, and 27)

132. A. Kumar, P. Rai, and H. Daume. Co-regularized multi-view spectral clustering. *Advances in Neural Information Processing Systems (NeurIPS)*, 24, 2011. (cited on page 2)

133. M. J. Kusner, B. Paige, and J. M. Hernández-Lobato. Grammar variational autoencoder. In *International Conference on Machine Learning*, pages 1945–1954. PMLR, 2017. (cited on page 119)

134. S. Lacoste-Julien. Convergence rate of frank-wolfe for non-convex objectives. *arXiv preprint arXiv:1607.00345*, 2016. (cited on pages 106 and 107)

135. A. N. Langville and C. D. Meyer. Google's pagerank and beyond: The science of search engine rankings, 2006. (cited on page 73)

136. J.-E. Lee, R. Jin, and A. K. Jain. Rank-based distance metric learning: An application to image retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2008. (cited on page 5)

137. R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2017. (cited on pages xxii, 5, 8, 9, 17, 18, 19, 57, 60, 62, 63, 65, and 68)

138. O. Levy, Y. Goldberg, and I. Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics (ACL)*, 3:211–225, 2015. (cited on page 97)

139. J. Li, R. Zhao, H. Hu, and Y. Gong. Improving rnn transducer modeling for end-to-end speech recognition. In *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 114–121. IEEE, 2019. (cited on page 2)

140. P. Li, I. Chien, and O. Milenkovic. Optimizing generalized pagerank methods for seed-expansion community detection. *Advances in Neural Information Processing Systems (NeurIPS)*, pages 11710–11721, 2019. (cited on page 20)

141. Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018. (cited on page 32)

142. R. Li, T. Zeng, H. Peng, and S. Ji. Deep learning segmentation of optical microscopy images improves 3-d neuron reconstruction. *IEEE Transactions on Medical Imaging*, pages 1533–1541, 2017. (cited on page 2)

143. S. Li, D. Kim, and Q. Wang. Beyond low-pass filters: Adaptive feature propagation on graphs. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, 2021. (cited on page 84)

144. Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *International Conference on Learning Representations (ICLR)*, 2015. (cited on pages 4 and 13)

145. Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020. (cited on page 20)

146. R. Liao, Z. Zhao, R. Urtasun, and R. S. Zemel. LanczosNet: Multi-scale deep graph convolutional networks. In *Proceedings of the seventh International Conference on Learning Representation (ICLR)*, 2019. (cited on pages 17, 18, 19, 62, 63, 64, and 65)

147. D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, pages 1019–1031, 2007. (cited on page 2)

148. D. Lim, G. Lanckriet, and B. McFee. Robust structural metric learning. In *International Conference on Machine Learning (ICML)*, pages 615–623. PMLR, 2013. (cited on page 5)

149. Z. Lin and Z. Kang. Graph filter-based multi-view attributed graph clustering. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2723–2729, 2021. (cited on page 3)

150. Z. Lin, Z. Kang, L. Zhang, and L. Tian. Multi-view attributed graph clustering. *IEEE Transactions on Knowledge and Data Engineering*, 2021. (cited on page 3)

151. R. Litman and A. M. Bronstein. Learning spectral descriptors for deformable shape correspondence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 171–180, 2013. (cited on page 20)

152. L. Litwin. Fir and iir digital filters. *IEEE potentials*, pages 28–31, 2000. (cited on page 81)

153. M. Liu, Z. Wang, and S. Ji. Non-local graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. (cited on page 84)

154. W. Liu, Z. Liu, F. Yu, P.-Y. Chen, T. Suzumura, and G. Hu. A scalable attribute-aware network embedding system. *Neurocomputing*, 339:279–291, 2019. (cited on page 3)

155. X. Liu, H. Pan, M. He, Y. Song, X. Jiang, and L. Shang. Neural subgraph isomorphism counting. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD)*, pages 1959–1969, 2020. (cited on page 16)

156. G. Loosli, S. Canu, and C. S. Ong. Learning svm in krein spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 38(6):1204–1216, 2015. (cited on page 23)

157. A. Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations (ICLR)*, 2020. (cited on pages 4 and 14)

158. Q. Lu and L. Getoor. Link-based classification. In *International Conference on Machine Learning (ICML)*, pages 496–503, 2003. (cited on pages 63 and 65)

159. R. Luss and A. d'Aspremont. Support vector machine classification with indefinite kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2008. (cited on page 104)

160. G. Ma, N. K. Ahmed, T. L. Willke, and P. S. Yu. Deep graph similarity learning: A survey. *Data Mining and Knowledge Discovery*, pages 688–725, 2021. (cited on page 5)

161. Y. Ma, X. Liu, N. Shah, and J. Tang. Is homophily a necessity for graph neural networks? In *International Conference on Learning Representation (ICLR)*, 2022. (cited on page 84)

162. L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. (cited on page 67)

163. P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. Extensions of marginalized graph kernels. In *International Conference on Machine Learning (ICML)*, page 70, 2004. (cited on page 23)

164. H. P. Maretic, M. El Gheche, G. Chierchia, and P. Frossard. GOT: an optimal transport framework for graph comparison. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. (cited on pages 6, 26, 95, and 97)

165. H. P. Maretic, M. E. Gheche, M. Minder, G. Chierchia, and P. Frossard. Wasserstein-based graph alignment. *arXiv preprint arXiv:2003.06048*, 2020. (cited on page 27)

166. H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations (ICLR)*, 2018. (cited on page 15)

167. H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. (cited on pages xxi, 4, 16, and 49)

168. F. Mémoli. Gromov–wasserstein distances and the metric approach to object matching. *Foundations of computational mathematics*, 11(4):417–487, 2011. (cited on pages 25 and 26)

169. T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. Metric learning for large scale image classification: Generalizing to new classes at near-zero cost. In *European Conference on Computer Vision (ECCV)*, pages 488–501. Springer, 2012. (cited on page 5)

170. G. Monge. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences de Paris*, 1781. (cited on page 25)

171. F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017. (cited on pages 5 and 8)

172. F. Monti, K. Otness, and M. M. Bronstein. Motifnet: a motif-based graph convolutional network for directed graphs. In *2018 IEEE Data Science Workshop (DSW)*, pages 225–228, 2018. (cited on page 16)

173. C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel. Faster kernels for graphs with continuous attributes via hashing. In *ICDM*, pages 1095–1100, 2016. (cited on page 111)

174. C. Morris, K. Kersting, and P. Mutzel. Glocalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 327–336. IEEE, 2017. (cited on page 24)

175. C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 4602–4609, 2019. (cited on pages 4, 16, 31, 45, and 47)

176. C. Morris, G. Rattan, and P. Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. (cited on pages 4 and 16)

177. R. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning (ICML)*, pages 4663–4673, 2019. (cited on page 17)

178. Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003. (cited on page 105)

179. M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102(2): 209–245, 2016. (cited on pages 24 and 111)

180. T. Nguyen, H. Le, T. P. Quinn, T. Nguyen, T. D. Le, and S. Venkatesh. Graphdta: Predicting drug–target binding affinity with graph neural networks. *Bioinformatics*, 37(8):1140–1147, 2021. (cited on page 1)

181. M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning (ICML)*, pages 2014–2023, 2016. (cited on page 111)

182. G. Nikolentzos, P. Meladianos, and M. Vazirgiannis. Matching node embeddings for graph similarity. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017. (cited on pages 6, 23, 27, 45, 95, 109, 111, and 113)

183. H. S. K.-i. Noma and K. Shimodaira. Dynamic time-alignment kernel in support vector machine. *Advances in Neural Information Processing Systems (NeurIPS)*, 2002. (cited on page 23)

184. D. Oglic and T. Gärtner. Learning in reproducing kernel kreın spaces. In *International Conference on Machine Learning (ICML)*, 2018. (cited on page 23)

185. M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen. Molecular de-novo design through deep reinforcement learning. *Journal of Cheminformatics*, pages 1–14, 2017. (cited on page 119)

186. F. Orsini, P. Frasconi, and L. De Raedt. Graph invariant kernels. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015. (cited on page 24)

187. A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, pages 808–828, 2018. (cited on page 1)

188. M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 1105–1114, 2016. (cited on page 3)

189. L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. (cited on pages 20 and 73)

190. E. Pardouxt. Stochastic partial differential equations and filtering of diffusion processes. *Stochastics*, pages 127–167, 1980. (cited on page 20)

191. G. Patané. Star-laplacian spectral kernels and distances for geometry processing and shape analysis. In *Computer Graphics Forum*, pages 599–624. Wiley Online Library, 2016. (cited on page 20)

192. H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2020. (cited on pages 46, 83, and 84)

193. E. Pekalska, P. Paclik, and R. P. Duin. A generalized kernel approach to dissimilarity-based classification. *The Journal of Machine Learning Research (JMLR)*, 2(Dec):175–211, 2001. (cited on page 23)

194. J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. (cited on page 98)

195. P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 629–639, 1990. (cited on page 20)

196. B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD)*, pages 701–710, 2014. (cited on pages 3, 63, and 65)

197. G. Peyré, M. Cuturi, and J. Solomon. Gromov-wasserstein averaging of kernel and distance matrices. In *International Conference on Machine Learning (ICML)*, 2016. (cited on pages 26, 96, and 109)

198. G. Peyré, M. Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019. (cited on pages 25 and 26)

199. A. Qamra, Y. Meng, and E. Y. Chang. Enhanced perceptual distance functions and indexing for image replica recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 27(3):379–391, 2005. (cited on page 23)

200. S. R. Qasim, H. Mahmood, and F. Shafait. Rethinking table recognition using graph neural networks. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 142–147. IEEE, 2019. (cited on page 1)

201. J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *ACM International Conference on Web Search and Data Mining (WSDM)*, pages 459–467, 2018. (cited on page 3)

202. M. Qu, J. Tang, J. Shang, X. Ren, M. Zhang, and J. Han. An attention-based collaboration framework for multi-view network representation learning. In *ACM on Conference on Information and Knowledge Management (CIKM)*, pages 1767–1776, 2017. (cited on page 3)

203. A. F. Queiruga, N. B. Erichson, D. Taylor, and M. W. Mahoney. Continuous-in-depth neural networks. *arXiv preprint arXiv:2008.02389*, 2020. (cited on page 20)

204. J. Rabin and N. Papadakis. Convex color image segmentation with optimal transport distances. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 256–269. Springer, 2015. (cited on page 25)

205. M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017. (cited on page 20)

206. W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, pages 2352–2449, 2017. (cited on page 2)

207. J. W. Raymond, E. J. Gardiner, and P. Willett. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal*, pages 631–644, 2002. (cited on page 6)

208. B. Rieck, C. Bock, and K. Borgwardt. A persistent weisfeiler-lehman procedure for graph classification. In *International Conference on Machine Learning (ICML)*, pages 5448–5458, 2019. (cited on pages 45 and 111)

209. R. A. Rossi and D. F. Gleich. Dynamic pagerank using evolving teleportation. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 126–137. Springer, 2012. (cited on pages 71, 73, 74, and 79)

210. V. Roth, J. Laub, M. Kawanabe, and J. M. Buhmann. Optimal cluster preserving embedding of nonmetric proximity data. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 25(12):1540–1551, 2003. (cited on page 23)

211. L. Rout, A. Korotin, and E. Burnaev. Generative modeling with optimal transport maps. In *International Conference on Learning Representations (ICLR)*, 2021. (cited on page 25)

212. S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, pages 2323–2326, 2000. (cited on page 3)

213. H. Sak, A. Senior, and F. Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014. (cited on page 2)

214. A. Sanchez-Gonzalez, V. Bapst, K. Cranmer, and P. Battaglia. Hamiltonian graph networks with ode integrators. *arXiv preprint arXiv:1909.12790*, 2019. (cited on page 20)

215. A. Sandryhaila and J. M. Moura. Discrete signal processing on graphs: Graph fourier transform. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6167–6170, 2013. (cited on page 18)

216. R. Sato. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078*, 2020. (cited on pages 4, 14, and 15)

217. R. Sato, M. Yamada, and H. Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341, 2021. (cited on pages 4 and 17)

218. F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008. (cited on pages 4 and 13)

219. K. Schacke. On the kronecker product. *Master's thesis, University of Waterloo*, 2004. (cited on page 101)

220. T. A. Schieber, L. Carpi, A. Díaz-Guilera, P. M. Pardalos, C. Masoller, and M. G. Ravetti. Quantification of network structural dissimilarities. *Nature communications*, 8(1):1–10, 2017. (cited on page 101)

221. B. Schölkopf, A. J. Smola, F. Bach, et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002. (cited on page 22)

222. M. H. Segler, T. Kogej, C. Tyrchan, and M. P. Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS Central Science*, pages 120–131, 2018. (cited on page 119)

223. P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008. (cited on pages 43, 63, and 82)

224. D. Shasha, J. T. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 39–52, 2002. (cited on page 6)

225. N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 488–495, 2009. (cited on pages 6, 22, and 23)

226. N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011. (cited on pages 6, 22, 24, 44, 45, 110, and 111)

227. W. Shi and R. Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1711–1719, 2020. (cited on page 1)

228. J. Shlomi, P. Battaglia, and J.-R. Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, page 021001, 2020. (cited on page 20)

229. D. Shuman, S. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30:83–98, 2013. (cited on pages 17 and 18)

230. M. Simonovsky and N. Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks (ICANN)*, pages 412–422. Springer, 2018. (cited on page 119)

231. A. P. Singh and G. J. Gordon. Relational learning via collective matrix factorization. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD)*, pages 650–658, 2008. (cited on page 3)

232. R. Sinkhorn. Diagonal equivalence to matrices with prescribed row and column sums. *The American Mathematical Monthly*, 74(4):402–405, 1967. (cited on page 27)

233. N. Sochen, R. Kimmel, and R. Malladi. A general framework for low level vision. *IEEE Transactions on Image Processing*, pages 310–318, 1998. (cited on page 20)

234. Q. P. Spaen. *Applications and advances in similarity-based machine learning*. University of California, Berkeley, 2019. (cited on page 5)

235. A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997. (cited on pages 4 and 13)

236. H. Stärk, D. Beaini, G. Corso, P. Tossou, C. Dallago, S. Günnemann, and P. Lio. 3d infomax improves gnns for molecular property prediction. In *NeurIPS 2021 AI for Science Workshop*, 2021. (cited on page 1)

237. J. Sun, M. Ovsjanikov, and L. Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *Computer Graphics Forum*, pages 1383–1392. Wiley Online Library, 2009. (cited on page 20)

238. Y. Sun, N. Bui, T.-Y. Hsieh, and V. Honavar. Multi-view network embedding via graph factorization clustering and co-regularized multi-view agreement. In *IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 1006–1013. IEEE, 2018. (cited on page 3)

239. J. J. Sutherland, L. A. O'brien, and D. F. Weaver. Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *Journal of chemical information and computer sciences*, 43(6):1906–1915, 2003. (cited on pages 44 and 110)

240. R. Takapoui and S. Boyd. Linear programming heuristics for the graph isomorphism problem. *arXiv preprint arXiv:1611.00711*, 2016. (cited on page 15)

241. J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *International Conference on World Wide Web (WWW)*, pages 1067–1077, 2015. (cited on page 3)

242. M. Thorpe, T. M. Nguyen, H. Xia, T. Strohmer, A. Bertozzi, S. Osher, and B. Wang. Grand++: Graph neural diffusion with a source term. In *International Conference on Learning Representations (ICLR)*, 2021. (cited on page 21)

243. V. Titouan, N. Courty, R. Tavenard, and R. Flamary. Optimal transport for structured data with application on graphs. In *International Conference on Machine Learning (ICML)*, pages 6275–6284, 2019. (cited on pages 23, 26, 27, 45, 47, 95, 109, 111, and 113)

244. V. Titouan, R. Flamary, N. Courty, R. Tavenard, and L. Chapel. Sliced gromov-wasserstein. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. (cited on page 102)

245. M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt. Wasserstein weisfeiler-lehman graph kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6439–6449, 2019. (cited on pages xxiii, 6, 23, 25, 27, 45, 95, 109, 110, 111, and 113)

246. C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *IEEE International Conference on Computer Vision (ICCV)*, pages 839–846. IEEE, 1998. (cited on page 20)

247. P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *International Conference on Learning Representations (ICLR)*, 2017. (cited on pages 4, 14, 15, 22, 37, 39, 42, 45, 57, 63, 64, 65, 83, and 84)

248. P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm. Deep graph infomax. In *International Conference on Learning Representations (ICLR)*, 2019. (cited on page 83)

249. J.-P. Vert. The optimal assignment kernel is not positive definite. *arXiv preprint arXiv:0801.4061*, 2008. (cited on page 25)

250. C. Vignac, A. Loukas, and P. Frossard. Building powerful and equivariant graph neural networks with structural message-passing. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. (cited on pages 4 and 17)

251. C. Villani. *Topics in optimal transportation*. Number 58. American Mathematical Soc., 2003. (cited on page 25)

252. C. Villani. *Optimal transport: old and new*, volume 338. Springer, 2009. (cited on page 6)

253. S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *The Journal of Machine Learning Research (JMLR)*, 11:1201–1242, 2010. (cited on pages 22 and 25)

254. N. Wale, I. A. Watson, and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14 (3):347–375, 2008. (cited on page 44)

255. W. D. Wallis, P. Shoubridge, M. Kraetz, and D. Ray. Graph distances using graph union. *Pattern Recognition Letters*, pages 701–704, 2001. (cited on page 6)

256. B. Wang, J. Jia, and N. Z. Gong. Semi-supervised node classification on graphs: Markov random fields vs. graph neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021. (cited on page 84)

257. D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *ACM SIGKDD international conference on Knowledge Discovery & Data Mining (SIGKDD)*, pages 1225–1234, 2016. (cited on page 3)

258. X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang. Community preserving network embedding. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017. (cited on page 3)

259. Y. Wang, Y. Shen, and D. Cremers. Explicit pairwise factorized graph neural network for semi-supervised node classification. In *Uncertainty in Artificial Intelligence*, pages 1979–1987. PMLR, 2021. (cited on page 84)

260. Y. Wei, W. Xia, M. Lin, J. Huang, B. Ni, J. Dong, Y. Zhao, and S. Yan. Hcp: A flexible cnn framework for multi-label image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1901–1907, 2015. (cited on page 2)

261. J. Weickert. *Anisotropic diffusion in image processing*, volume 1. Teubner Stuttgart, 1998. (cited on page 20)

262. B. Weisfeiler and A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, 2(9):12–16, 1968. (cited on pages 15, 27, 31, and 111)

263. J. Weston, F. Ratle, H. Mobahi, and R. Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012. (cited on pages 63 and 65)

264. A. Wijesinghe and Q. Wang. Dfnets: Spectral cnns for graphs with feedback-looped filters. *Advances in neural information processing systems (NeurIPS)*, 2019. (cited on pages 45 and 52)

265. P. Willett, J. M. Barnard, and G. M. Downs. Chemical similarity searching. *Journal of Chemical Information and Computer Sciences*, pages 983–996, 1998. (cited on page 6)

266. F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning (ICML)*, pages 6861–6871. PMLR, 2019. (cited on page 83)

267. J. Wu, J. He, and J. Xu. Demo-net: Degree-specific graph neural networks for node and graph classification. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD)*, pages 406–415, 2019. (cited on page 31)

268. Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020. (cited on page 1)

269. L.-P. Xhonneux, M. Qu, and J. Tang. Continuous graph neural networks. In *International Conference on Machine Learning (ICML)*, pages 10432–10441. PMLR, 2020. (cited on page 20)

270. C. Xie, M. Tan, B. Gong, A. Yuille, and Q. V. Le. Smooth adversarial training. *arXiv preprint arXiv:2006.14536*, 2020. (cited on page 76)

271. Z. Xinyi and L. Chen. Capsule graph neural network. In *International conference on learning representations (ICLR)*, 2018. (cited on pages 45, 46, 51, and 111)

272. H. Xu, D. Luo, and L. Carin. Scalable gromov-wasserstein learning for graph partitioning and matching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. (cited on page 27)

273. H. Xu, D. Luo, H. Zha, and L. C. Duke. Gromov-wasserstein learning for graph matching and node embedding. In *International Conference on Machine Learning (ICML)*, 2019. (cited on page 27)

274. K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning (ICML)*, pages 5453–5462. PMLR, 2018. (cited on pages 31 and 84)

275. K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019. (cited on pages xxi, 4, 15, 16, 31, 37, 39, 41, 42, 45, 46, 47, 49, 51, 52, 71, 110, and 111)

276. X. Yan, P. S. Yu, and J. Han. Graph indexing: a frequent structure-based approach. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 335–346, 2004. (cited on page 6)

277. P. Yanardag and S. Vishwanathan. Deep graph kernels. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD)*, pages 1365–1374, 2015. (cited on pages 16, 27, 44, and 110)

278. Z. Yang, W. W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning (ICML)*, pages 40–48, 2016. (cited on pages 63, 64, and 65)

279. J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning (ICML)*, pages 5708–5717. PMLR, 2018. (cited on page 119)

280. J. You, J. Gomes-Selman, R. Ying, and J. Leskovec. Identity-aware graph neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021. (cited on pages 4, 17, 45, 47, and 55)

281. Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, pages 25–36, 2009. (cited on page 6)

282. A. Zeyer, P. Doetsch, P. Voigtlaender, R. Schlüter, and H. Ney. A comprehensive study of deep bidirectional lstm rnns for acoustic modeling in speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2462–2466. IEEE, 2017. (cited on page 2)

283. B. Zhang, S. Choudhury, M. A. Hasan, X. Ning, K. Agarwal, S. Purohit, and P. P. Cabrera. Trust from the past: Bayesian personalized ranking based link prediction in knowledge graphs. *arXiv preprint arXiv:1601.03778*, 2016. (cited on page 2)

284. K. Zhang, Y. Zhu, J. Wang, and J. Zhang. Adaptive structural fingerprints for graph attention networks. In *International Conference on Learning Representations*, 2019. (cited on page 118)

285. M. Zhang and Y. Chen. Weisfeiler-lehman neural machine for link prediction. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD)*, pages 575–583, 2017. (cited on page 31)

286. M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018. (cited on pages 45, 47, and 111)

287. W. Zhang, R. Li, H. Deng, L. Wang, W. Lin, S. Ji, and D. Shen. Deep convolutional neural networks for multi-modality isointense infant brain image segmentation. *NeuroImage*, pages 214–224, 2015. (cited on page 2)

288. X.-M. Zhang, L. Liang, L. Liu, and M.-J. Tang. Graph neural networks and their current applications in bioinformatics. *Frontiers in Genetics*, 12, 2021. (cited on page 1)

289. Z. Zhang, M. Wang, Y. Xiang, Y. Huang, and A. Nehorai. Retgk: Graph kernels based on return probabilities of random walks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3964–3974, 2018. (cited on pages 45 and 111)

290. Z. Zhang, J. Bu, M. Ester, Z. Li, C. Yao, Z. Yu, and C. Wang. H2mn: Graph similarity learning with hierarchical hypergraph matching networks. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD)*, pages 2274–2284, 2021. (cited on page 5)

291. J. Zhao, Y. Dong, M. Ding, E. Kharlamov, and J. Tang. Adaptive diffusion in graph neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021. (cited on pages 5, 20, 71, 77, and 79)

292. L. Zhao and L. Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations (ICLR)*, 2019. (cited on page 32)

293. D. Zhou, S. Niu, and S. Chen. Efficient graph computation for node2vec. *arXiv preprint arXiv:1805.00280*, 2018. (cited on page 3)

294. J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. (cited on page 1)

295. Y. Zhou, A. Amimeur, C. Jiang, D. Dou, R. Jin, and P. Wang. Density-aware local siamese autoencoder network embedding with autoencoder graph clustering. In *IEEE International Conference on Big Data*, pages 1162–1167. IEEE, 2018. (cited on page 3)

296. H. Zhu and P. Koniusz. Simple spectral graph convolution. In *International Conference on Learning Representations (ICLR)*, 2021. (cited on page 83)

297. J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 2020. (cited on pages 5, 84, and 85)

298. X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *International conference on Machine learning (ICML)*, pages 912–919, 2003. (cited on pages 63 and 65)

299. M. Zitnik, M. Agrawal, and J. Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 2018. (cited on page 5)