Decoding the Neighborhood Aggregation Mechanism of GNNs: From Generalization to Interpretive Analysis

by Shuming Liang

Supervisors: Prof. Yang Wang
A/Prof. Zhidong Li
Dr. Bin Liang

Faculty of Engineering and Information Technology
University of Technology Sydney

This dissertation is submitted for the degree of
Doctor of Philosophy

November 2023

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, *Shuming Liang*, declare that this thesis is submitted in fulfillment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution. This research is supported by the Australian Government Research Training Program.

Signature: Production Note:
Signature removed prior to publication.

Date:

4 Dec 2023

Acknowledgements

I am humbled and deeply grateful as I reflect on the completion of my Ph.D. journey. This significant milestone would not have been possible without the unwavering support and encouragement of numerous individuals who have touched my life in profound ways.

I extend my heartfelt appreciation to my supervisory panel, including Prof. Yang Wang, Dr. Zhidong Li, and Dr. Bin Liang. Your expertise, mentorship, and dedication have been instrumental in shaping my research endeavors. Your insightful feedback, constructive criticism, and endless patience have guided me towards intellectual growth and academic excellence. I am grateful for the trust you placed in me, allowing me the freedom to explore new ideas while providing invaluable guidance along the way. Your unwavering support has been the cornerstone of my development as a researcher, and I am honored to have had the privilege to work under your mentorship.

To my family, I am forever indebted to your unconditional love and continuous encouragement. Your unwavering support has provided me with the strength and resilience to overcome challenges and pursue my dreams. You have been my constant source of inspiration, and I am eternally grateful for your presence in my life.

To my friends, thank you for being a constant source of joy, laughter, and companionship. Your unwavering camaraderie provided the much-needed respite from the academic rigors and reminded me of the importance of balance and human connection.

To my beloved wife, Dr. Yu Ding, you have been my rock and my guiding light throughout this arduous journey. Your love and companionship have been a source of inspiration and comfort, providing balance and perspective during the most challenging times. Your sacrifices and selflessness have allowed me to fully immerse myself in my research, knowing that I had your unwavering support every step of the way. Your unwavering belief in me, even during moments of self-doubt, has been the driving force behind my perseverance and determination. This achievement would not have been possible without you by my side, and I am honored to share this success with you.

I would like to express my deepest gratitude to every individual who has contributed to my personal and academic growth. I am truly fortunate to have crossed paths with such remarkable individuals. May this acknowledgement serve as a testament to my appreciation and as a reminder of the profound impact you have had on my journey.

Publications

1. **Shuming Liang**, Zhidong Li, Bin Liang, Yu Ding, Yang Wang, Fang Chen. Failure Prediction for Large-scale Water Pipe Networks Using GNN and Temporal Failure Series. In Proceedings of the 30th ACM International Conference on Information Knowledge Management, CIKM '21, page 3955–3964, New York, NY, USA.

2. **Shuming Liang**, Yu Ding, Zhidong Li, Bin Liang, Yang Wang, Fang Chen. Analytical and Empirical Insights into GNNs for Link Prediction. Under triple-blind submission and review.

3. **Shuming Liang**, Bin Liang, Zhidong Li, Yu Ding, Siqi Zhang, Yang Wang, Fang Chen. VAGNN: Enhancing Generalization and Scalability in Graph Neural Networks. Under triple-blind submission and review.

4. **Shuming Liang**, Andy Guo, Bin Liang, Zhidong Li, Yu Ding, Yang Wang, Fang Chen. Machine Learning and Computer Vision Applications in Civil Infrastructure Inspection and Monitoring. Infrastructure Robotics: Methodologies, Robotic Systems and Applications.

5. Yu Ding, Lei Wang, Bin Liang, **Shuming Liang**, Yang Wang, Fang Chen. Domain Generalization by Learning and Removing Domain-specific Features. In Thirty-sixth Annual Conference on Neural Information Processing Systems (NeurIPS 2022), pp.24226-24239.

6. Bin Liang, Zhidong Li, Hongda Tian, **Shuming Liang**, Yang Wang, Fang Chen. Optimising Water Quality with Data Analytics and Machine Learning. In Advances in Data Science and Analytics: Concept and Paradigm.

7. Bin Liang, Sunny Verma, Jie Xu, **Shuming Liang**, Zhidong Li, Yang Wang, Fang Chen. A Data Driven Approach for Leak Detection with Smart Sensors. 2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV), Shenzhen, China, 2020, pp. 1311-1316.

Abstract

Graph Neural Networks (GNNs) have emerged as a powerful tool for analyzing graph-structured data, enabling effective graph representation learning and capturing complex dependencies among nodes. This thesis presents three research works that contribute to the advancement of GNNs in generalization, scalability, and interpretability.

The first research work proposes a general and scalable GNN framework called Virtual Adjacency Graph Neural Network (VAGNN). VAGNN introduces the concept of an optimizable virtual adjacency matrix, enabling the flexible construction of neighborhood sets for aggregation in GNNs. It overcomes the limitations of existing GNNs by customizing the inclusion of local and global nodes, offering a selection of attention mechanisms for the aggregation function, and providing the ability to incorporate supplementary information for further control over the aggregation process. VAGNN demonstrates improved generalization and scalability capabilities, with experimental evaluations on real-world datasets validating its performance.

The second work focuses on the interpretive analysis of GNNs in link prediction. By exploring the underlying mechanisms of GNNs in link prediction, this research investigates the learning of pair-specific structural information and the effectiveness of node embeddings in GNNs. Our analysis shows that traditional link heuristics like common neighbors convey effective information that is difficult to be learned by GNNs. Also, we discover that node embeddings can enhance the performance of GNN-based link prediction, and the denser the graph, the more the enhancement. This work reveals the limitations of existing GNN-based link prediction models and highlights the importance of incorporating common neighbor-based heuristics. Empirical evaluations on real-world datasets support our findings and provide insights for the design of more robust link prediction algorithms.

The third research work undertakes an investigation into the inner working mechanism of diverse GNN techniques in real-world scenarios. Our objective revolves around identifying the challenges that arise in practical problems, proposing subsequent solutions to these challenges through the utilization of various techniques, and providing insightful interpretations regarding the effectiveness of these techniques. Specifically, our investigation is concentrated in the context of node classification, where the nodes in a graph are distributed geospatially and associated with temporal information. We introduce a GNN-based framework called Multi-hop Attention-based GNN (MAG). This framework

develops a novel geographical graph construction method, aiming to capture the geographical neighboring effects among nodes. Furthermore, MAG addresses a multitude of challenges, including differentiating information from different neighbors through an attention mechanism, mitigating the over-smoothing issue via the use of residual connections, and capturing temporal effects with the integration of a point process module. Experimental evaluations on real-world datasets demonstrate the superiority of MAG over previous models, validating the effectiveness of our proposed framework and supporting our interpretations and findings.

These research works contribute to the development and understanding of GNNs. They present novel GNN frameworks, provide insights into the strengths and limitations of GNNs in link prediction, and interpret the working mechanism of different techniques in GNNs. The findings and methodologies presented in this thesis contribute to advancing the field of graph representation learning and open up new possibilities for analyzing complex systems represented as graphs.

Contents

List of Figures xv

List of Tables xix

1 Introduction 1

1.1 Background . . . 1
1.2 Research Motivations, Objectives and Contributions . . . 2
1.2.1 VAGNN: A General and Scalable GNN Framework . . . 2
1.2.2 An Interpretive Analysis of GNNs in Link Prediction . . . 4
1.2.3 Exploring the Functionalities of Diverse GNN Techniques in Real-World Scenarios . . . 5
1.2.4 A Summary of Contributions . . . 7
1.3 Thesis Organization . . . 7

2 Literature Review 9

2.1 Notations . . . 9
2.2 Graph Neural Networks . . . 10
2.2.1 Graph Convolutional Network . . . 10
2.2.2 Graph Attention Network . . . 12
2.2.3 Addressing the Scalability Challenges in GNNs . . . 14
2.2.4 High-hop GNNs . . . 17
2.2.5 Graph Transformers . . . 19
2.2.6 More Noteworthy GNNs . . . 22
2.3 Other Graph Representation Learning Methods . . . 24
2.3.1 Graph Autoencoders . . . 24
2.3.2 Random Walk Methods . . . 24
2.3.3 Representation Leaning on Dynamic Graphs . . . 26
2.3.4 Point Process Embedding Methods . . . 30
2.4 Summary . . . 33

3 VAGNN: A General and Scalable GNN Framework **35**
3.1 Introduction . . . 35
3.2 Preliminary . . . 37
3.3 The Proposed VAGNN . . . 38
3.3.1 Attention Methodologies . . . 40
3.3.2 The Extra Weight . . . 42
3.3.3 Virtual Adjacency Matrix . . . 42
3.3.4 The Expressiveness of VAGNN . . . 44
3.4 Experiments . . . 45
3.4.1 Datasets . . . 45
3.4.2 Implementation . . . 49
3.4.3 Examination of VAGNN Generalization Capabilities . . . 50
3.4.4 Parameter Sensitivity Study on Virtual Adjacency Matrix . . . 53
3.4.5 Attention Methods Assessment . . . 54
3.4.6 Exploration of the Superior Model Performance . . . 55
3.5 Summary . . . 55

4 An Interpretive Analysis of GNNs in Link Prediction **57**
4.1 Introduction . . . 57
4.2 Preliminaries . . . 60
4.2.1 Problem Definition . . . 60
4.2.2 NCN-dependent Structural Information . . . 60
4.2.3 GNNs in Learning the Number of Neighbors . . . 61
4.3 NCN-dependent Structural Information cannot be Effectively Learned via GNNs . . . 63
4.3.1 Analytical Study . . . 63
4.3.2 Empirical Study . . . 65
4.3.3 Experimental Settings . . . 66
4.4 Node Embedding in Link Prediction . . . 67
4.4.1 Experimental Observations . . . 67
4.4.2 Analytical Insights into Node Embeddings . . . 67
4.5 Limitation Analysis of Existing Methods . . . 69
4.5.1 A Survey of Link Prediction Methods . . . 69
4.5.2 Limitation Analysis . . . 71
4.5.3 Further Analysis of Experimental Results . . . 74
4.6 More on the Experiments . . . 76
4.6.1 GNN Models Used in the Experiments . . . 76
4.6.2 SEAL-type Methods. . . . 78
4.6.3 Implementation Details . . . 78

4.6.4 Additional Experimental Results . . . 78
4.7 Summary and Implication . . . 78

5 Exploring the Functionalities of Diverse GNN Techniques in Real-World Scenarios 81
5.1 Introduction . . . 81
5.2 Related Work in Pipe Failure Prediction . . . 83
5.3 Problem Formulation . . . 84
5.4 Proposed Framework . . . 85
5.4.1 GNN Module . . . 85
5.4.2 Multi-hop Aggregation . . . 85
5.4.3 Attention-based GNN Layer . . . 86
5.4.4 Residual Connections and Layer-wise Aggregation . . . 87
5.4.5 Learning Temporal Failure Pattern . . . 88
5.4.6 Failure Predictor . . . 88
5.5 Experiments . . . 89
5.5.1 Datasets . . . 89
5.5.2 Data Preprocessing . . . 89
5.5.3 Experimental Settings . . . 92
5.5.4 Main Results . . . 93
5.6 Further Analysis . . . 94
5.6.1 The Sensitivity of Aggregation Methods . . . 94
5.6.2 Failure Detection Rate . . . 95
5.6.3 Hyper-parameters and Training Time . . . 95
5.7 Application . . . 96
5.7.1 Pipe Prioritization Platform . . . 96
5.7.2 Feature Importance . . . 99
5.8 Summary . . . 100

6 Conclusion and Future Work 101
6.1 Conclusion . . . 101
6.2 Future Work . . . 102

Bibliography 105

List of Figures

2.1 Over smoothing when the number of graph convolution layers increases. The points are the node embeddings of Zachary's karate club graph with GCNs. Figure is extracted from Li et al [78]. 11

2.2 The standard GAT (left) computes static attention - the ranking of attention coefficients is global for all nodes in the graph, and is unconditioned on the query node. In contrast, GATv2 (right) can actually compute dynamic attention, where every query has a different ranking of attention coefficients of the keys. This figure is from the work of [14]. 13

2.3 Performance comparisons of using one cluster versus multiple clusters. The former approach uses 300 partitions, while the latter involves the utilization of 1500 partitions, with a random selection of 5 partitions to compose each batch. The results present the epoch (x-axis) versus the F1 score (y-axis). This figure is from the work of Cluster-GCN [25]. 16

2.4 The architecture of a representation learning method for dynamic graphs from Manessi et al. [91]. 28

2.5 Visualization of four models. Image extracted from MHDNE [155]. . . . 33

3.1 An illustration of the methods constructing a virtual adjacency matrix on an undirected graph. 39

3.2 The results of the parameter sensitivity study on the virtual adjacency matrix construction. The three plots (a) (b) (c) correspond to three different configurations for the 1-hop conversion of high-hop local neighboring nodes. For example, in plot (a), the term "1-hop" represents the virtual adjacency matrix that uses the local neighboring nodes within 1 hop (i.e., only the actual 1-hop nodes). In plot (b), the term "2-hop" refers to the conversion of local neighboring nodes within 2 hops into 1-hop neighbors in the virtual adjacency matrix. The colors of grid cells in the plots indicate the performance (Hits@20) of the proposed VAGNN on the *ogbl-ddi* dataset. 50

3.3 The performance improvement of attention-based GNN models over the baseline GCN [64]. Boxplots present the statistical analysis based on the results obtained from the nine datasets used in this work. 54

4.1 An illustration of neighborhood information propagation and aggregation in GNNs, where $a_{i,j}$ can be an edge weight or attention weight from node j to i. 62
4.2 The results of Algorithm 3 using heuristic encoding (HE) only, node features (X) only, node embeddings (NE) only, or their combinations. The nodes in ogbl-ddi do not have any features and we use the node degree as the node feature. 65
4.3 Node labeling in SEAL-type methods. The left is a subgraph specific for a positive link sample and the right is a negative one. The labeling features are based on the SPDs from every node (here only show the first-order neighbors of node v or w) to the target pair of nodes. For example, on the left, the node with the labeling $(1,10)$ indicates that the SPD from this node to node v and u is 1 and 10, respectively. 73
4.4 Results of different methods for link prediction on four OGB datasets. For MLP and general GNNs, we present their results obtained by utilizing node embeddings, considering the dominant performance of node embeddings as shown in Fig. 4.2. 74
4.5 The algorithmic flow chart of SEAL-type methods. 77
4.6 Results of encoding different numbers of link heuristics. We can find that the best result on each dataset is achieved by encoding a certain number of heuristics rather than all heuristics. 79

5.1 The architecture of our proposed framework MAG. It contains two main procedures: data preprocessing and failure prediction. Data preprocessing includes data collection, geographical graph construction, feature engineering, and temporal failure series extraction. In graph construction, two nodes are geographically linked (red edge) if two corresponding pipes without physical joint are geographically close to each other. In each GNN layer, we employ an attention mechanism and multi-hop aggregation. We add a layer-wise aggregation layer following the last GNN layer and the hidden representations of previous GNN layers are reserved in this layer through residual connections. In addition to the GNN module, We use a module to learn the temporal failure pattern including the base evolutionary effect and historical failures' time-decayed excitement on the current state of a pipe. MLP is used as the final failure predictor. 83

5.2 An example of our data shown on the map. It includes water pipes (green lines) and failures (yellow dots), sewer pipes (blue line) and failures (light blue dots), as well as tree canopy (red polygons). 90
5.3 Failure analysis on elevation-based features. Failure rate is defined as the number of pipe failures per 100KM per year. Slope is the elevation difference between two ends of a pipe divided by pipe length. More than 95% of pipes have the slope less than 0.08. There are three types of shapes for three adjacent pipes in the elevation direction. 90
5.4 Comparison of baseline GCN [64] and GraphSAGE [46]. GS1 is the GraphSAGE with concatenation as COMBINE$(\cdot)$ and mean pooling as AGG$(\cdot)$, GS2: concatenation and LSTM aggregator, GS3: concatenation and max pooling, GS4: max and mean pooling, GS5: max pooling and LSTM aggregator, GS6: max and max pooling. 95
5.5 Failure detection rate vs ranking predictions on water pipe dataset. 96
5.6 Failure detection rate vs ranking predictions on sewer pipe dataset. 97
5.7 An example of risk map based on pipe prioritization. 97
5.8 Feature importance. 98

List of Tables

2.1 Random partition versus clustering partition of the graph on the GNN performance. Clustering partition leads to better performance (in terms of test F1 score) since it removes fewer between-partition links. Results are from the work of Cluster-GCN [25]. 15

3.1 A brief summary of the notations used in VAGNN 38
3.2 Statistics of OGB link prediction datasets used in our experiments. 48
3.3 Results of node classification. 51
3.4 Results of link prediction 51
3.5 Results of graph classification 52

4.1 NCN-dependent link heuristics between nodes v, u. 60
4.2 Results on test sets of OGB link prediction datasets. Higher is better. For a fair comparison, we only list the results that do not take advantage of the validation dataset as training data. 77

5.1 Basic statistics for two pipe datasets 89
5.2 Feature description 91
5.3 Results for two pipe networks 93
5.4 Best tuned hyper-parameters for our framework 96

Chapter 1

Introduction

1.1 Background

Graph Neural Networks (GNNs) are a class of neural networks that have achieved significant success in recent years for their ability to effectively analyze and model graph-structured data. Unlike traditional neural networks that operate on regular grid-like or sequential data such as images [68, 49], texts [135, 30], and audio signals [50, 8], GNNs are designed to handle graph-structured data, where nodes (also known as vertices) are interconnected by edges (also known as links), and the number of neighboring nodes varies across different nodes. GNNs have demonstrated remarkable efficacy in a wide range of real-world scenarios, such as social network analysis [9], recommendation systems [157], and molecular chemistry analysis [84].

The development of GNNs can be traced back to the early 2000s or possibly even earlier when researchers began exploring ways to extend traditional neural networks to effectively process diverse graph structures [175, 42]. A significant breakthrough in GNNs came with the introduction of Graph Convolutional Network (GCN) by Kipf and Welling in 2016 [64]. Specifically, the convolutional operation is originally developed for analyzing regular grid-like data, leveraging local contextual relationships and sharing weights to learn hierarchical representations [74, 50]. Different from the regular grid-like data in which each node (a pixel in an image) has fixed number of neighboring nodes, nodes in graph-structured data have diverse numbers of neighbors, which brings traditional neural networks algorithmic challenges in propagating information from neighbors to the central nodes [175, 42, 64]. GCN adapts the concept of convolution to the graph domain. It introduces a localized neighborhood aggregation scheme, which iteratively updates the representation of each node by aggregating information from its neighboring nodes. This iterative process is typically performed over multiple layers, enabling the GCN to propagate and incorporate information over the irregular graph structures and capture higher-order dependencies among nodes. Consequently, the GCN enables effective

node-level representation learning from both node features (attributes associated with each node) and the graph topology (connectivity pattern between nodes).

Over the years, numerous variations and extensions of GNNs have been proposed, each addressing different challenges and leveraging different aspects of graph data. Some notable GNN architectures include GraphSAGE [47], Graph Attention Network (GAT) [136], Distance Encoding GNNs (DEGNN) [77], Graph Transformers [135, 30], and others. These models differ in terms of neighborhood construction strategies, attention mechanisms, global node utilization, and other architectural designs.

The development and advancement of GNNs have significantly contributed to the field of graph representation learning and have opened up new possibilities for analyzing and understanding complex structured data. Ongoing research continues to explore advancements in GNNs, addressing challenges such as scalability, interpretability, etc. Researchers are also exploring novel applications and adapting GNNs to domains such as computer vision [123], natural language processing [146], and reinforcement learning [96]. As the field continues to evolve, GNNs are expected to play a crucial role in advancing our understanding and analysis of complex systems represented as graphs.

1.2 Research Motivations, Objectives and Contributions

This thesis presents a comprehensive exploration of GNNs, encompassing three distinct research works that contribute to the advancement of GNNs in terms of generalization, scalability, and interpretability.

1.2.1 VAGNN: A General and Scalable GNN Framework

The first research work focuses on the development of a general and scalable GNN framework called Virtual Adjacency Graph Neural Network (VAGNN). The motivation behind VAGNN stems from a comprehensive commonality analysis of various GNN architectures. Specifically, a majority of modern GNNs belong to the family of the neighborhood aggregation algorithm [150], also referred to as the message-passing mechanism [40]. These models introduce innovations mainly centered on the neighborhood construction methods and aggregation functions. For example, GraphSAGE [46] removes certain actual edges to improve the computational efficiency of GNNs, where a fixed-size set of 1-hop neighbors is uniformly sampled from the full neighborhood sets. High-hop GNNs, such as DEGNN [77], MixHop [1], and KPGNN [35], incorporate higher-hop neighboring nodes along with 1-hop nodes for neighborhood aggregation. Graph Transformers [32, 159, 156, 19] employ the self-attention mechanism that takes into account all nodes, where the neighborhood of each node for aggregation is all the other nodes in the graph.

However, existing GNNs exhibit certain deficiencies. Most previous GNNs primarily concentrate on local neighbors within limited hops, neglecting the incorporation of global nodes. On the other hand, graph transformers [167, 32, 156, 102] take into account all nodes in the process of neighborhood information aggregation, with each node attending to every other node during self-attention computation. Nevertheless, such graph transformers come with several limitations. Firstly, the inclusion of numerous global nodes would excessively dilute the information from local neighbors, leading to the over-dilution issue [69]. Secondly, this setting makes the graph Transformer computationally expensive or even infeasible for large graphs [156]. Moreover, it can be likened to a language Transformer [135] taking the entire dictionary as input instead of a single sentence, which is unreasonable.

The objective of this research work is to explore common characteristics among diverse GNN architectures and graph Transformers [64, 137, 156], with the goal of unifying them through the introduction of a more general GNN framework. We propose VAGNN, a novel approach that incorporates an optimizable virtual adjacency matrix to improve the generalization and scalability of GNNs. Our method introduces the concept of the virtual adjacency matrix, allowing for a more flexible construction of the neighborhood set for aggregation in GNNs. Firstly, with the concept of the virtual adjacency matrix, we can construct the neighborhood set for the aggregation in GNNs from a general perspective. Unlike conventional GNNs that construct a fixed neighborhood set, VAGNN enables customization of the set by selectively removing 1-hop neighbors and incorporating high-hop local neighbors and global nodes. Additionally, VAGNN offers a selection of attention mechanisms for the aggregation function and the option to incorporate supplementary information for further control over the aggregation process. Theoretically, VAGNN can be transformed into a majority of existing GNNs and graph Transformers[137, 77, 35, 156]. Also, it holds the potential to achieve superior performance by optimizing the virtual adjacency matrix and tailoring aggregation methods specifically for the given graph dataset.

The virtual adjacency matrix in VAGNN serves as a mask to determine whether the nodes require attention calculation and information aggregation or not. When implementing VAGNN based on sparse matrix operations, its computational complexity exhibits a linear relationship with the number of node pairs used for neighborhood aggregation. This linear complexity demonstrates the scalability of VAGNN, rendering it well-suited for large graphs [54]. Especially, by selectively choosing global nodes, VAGNN effectively overcomes the scalability challenges associated with the utilizzation of global nodes in graph Transformers [159, 156, 102], reducing the storage complexity from $O(N^2)$ or even $O(N^3)$ to N, where N denotes the total number of nodes in a graph.

Experimental evaluations on diverse real-world datasets validate the generalization and scalability capabilities of VAGNN. Parameter sensitivity analysis using VAGNN also reveals the importance of carefully selecting and balancing the inclusion of local and global

information. This work lays the foundation for further exploration in developing more efficient techniques for virtual adjacency matrix construction and weighted aggregation functions, opening up possibilities for the design of more robust GNNs in the future.

1.2.2 An Interpretive Analysis of GNNs in Link Prediction

The second research work conducts an analytical and empirical analysis of GNNs in link prediction. Link prediction is a node-pair-specific problem, aiming to estimate the likelihood of the existence of an unknown edge between two nodes in graphs. Link prediction plays a pivotal role in various fields, including social network analysis [9], recommendation systems [157], and biological network analysis [84]. Understanding the underlying mechanisms and factors that drive link formation and evolution is crucial for accurate link predictions. This work investigates the inner workings of GNNs for link prediction, exploring the impact of different components including node features, node embeddings, heuristic information like common neighbors, etc. Through a combination of theoretical analysis and empirical evaluations on real-world datasets, this research work aims to provide valuable insights into the strengths and limitations of GNNs for link prediction tasks.

GNNs have demonstrated powerful expressiveness in graph representation learning [173]. However, what structural information specific to two nodes can be learned via GNNs remains an open question. A prominent task that relies on such information is link prediction. Nevertheless, existing GNN-based link prediction works [168, 124, 142, 171, 44, 145] primarily pay their attention to the model design, rare of them touch on the question: Do their models learn pair-specific structural information for link prediction?

Motivated by this, we present analytical and empirical investigations into the link prediction capability of GNNs, with a focus on two fundamental questions: 1) *whether GNNs can effectively learn the pair-specific structural information related to the number of common neighbors*? and 2) *node embeddings can improve the performance of GNN-based link prediction models, and the denser the graph, the more the improvement. How do we interpret this?*

In GNNs, a neighborhood aggregation scheme is typically employed, where the representation of each node is iteratively updated by aggregating the representations of that node and its neighbors [27]. However, it has been recognized that the learned node-wise representations can hardly capture information related to the number of its neighbors, primarily because the set-based pooling of the aggregation scheme inherently disregards the size of the neighborhood set of each node [150, 170].

A strategy for applying node-wise representations learned by GNNs to downstream multiple-node tasks (e.g., link prediction, graph classification, etc.) is to combine the representations of the nodes involved in these tasks. For link prediction, we find that

the combination of two nodes' representations essentially lacks the ability to capture information related to the number of common neighbors. This is mainly because node-wise representations learned by GNNs inherently lack information about the number of neighbors of each node, and most operations of combining two nodes' representations (e.g., concatenation, Hadamard production, etc.) also do not contain any behaviors of counting how many common neighbors between two nodes. To empirically verify this, we incorporate traditional link heuristics (e.g., Common Neighbors) into the GNN and compare the link prediction performance between the GNN combined with link heuristics and only the GNN. The approach yields results either superior or comparable to those obtained by using only GNNs, experimentally supporting our analysis.

In our experiments, we find that trainable node embeddings can enhance the performance of GNN-based link prediction models, and the denser the graph, the stronger the enhancement. Our explanation is as follows. Compared to the model weights of a GNN that are shared across all nodes [64, 27], each trainable node embedding is unique to its respective node. This characteristic of node embeddings can benefit the model. When the training is supervised by positive and negative link samples (i.e., two nodes are not linked), the link state of two nodes in every link sample could be encoded into the embeddings of that two nodes and the embeddings of their neighboring nodes under the help of the neighborhood aggregation process of the GNN. This would enable each node embedding to remember the relationships of that node to other nodes, allowing the model to know better which two nodes are more likely to be linked or not. Furthermore, in the neighborhood aggregation of the GNN, the denser graphs would allow each node to see more other nodes, allowing for better learning of node embeddings for link prediction.

This research work provides deeper insights into the expressiveness of GNNs in link prediction. These insights can help identify and interpret the limitations of existing link prediction methods, potentially directing the search for more robust algorithms. In addition, we empirically compare the performance of multiple types of link prediction methods on real-world datasets. The results can be interpreted with our insights, further underlining the significance of our findings.

1.2.3 Exploring the Functionalities of Diverse GNN Techniques in Real-World Scenarios

Various advanced techniques have been developed within the realm of GNNs, including attention mechanisms, high-hop neighborhood construction, residual connections, and more. Despite the progress, certain fundamental questions remain open regarding the inner working functionality as well as the effectiveness of these techniques in addressing challenges inherent in the practical applications of GNNs. Without the loss of generalization, this study is dedicated to answering these questions within the context of a real-world

application: pipe failure prediction in water pipe networks. Several characteristics of the pipe networks make such an application an ideal testing-ground for validating and exploring different GNN techniques. To begin, the pipes possess a multitude of attributes, such as length, material, and diameter, with each attribute spanning a distinguishable range of values. Secondly, the pipes are accompanied by temporal information, including pipe age, historical failure occurrences, maintenance records, etc. Furthermore, the pipes are distributed geographically, which requires the construction of the graph representing the pipe network to consider not only physical connections but also geospatial neighboring effects among pipes[1]. Lastly, we treat pipes as nodes in the graph construction and the problem of pipe failure prediction can be transformed to a node classification task.

In this research, we introduce a novel GNN-based framework called Multi-hop Attention-based GNN (MAG). We integrate a diverse set of advanced techniques into MAG and provide an in-depth interpretation of the roles played by these techniques in tackling challenges that arise in pipe failure prediction applications.

The first challenge we address is capturing both the structural information of pipe connectivity and geographical neighboring information by the GNN. To achieve this, we represent the pipes as nodes and construct the edges between nodes according to the physical joints between pipes and the geographical distance between the two corresponding pipes. Secondly, the neighborhood information aggregation of GNN may lead to the loss of key information associated with pipes, especially when the connected pipes vary significantly. To address this concern, MAG adopts an attention mechanism to learn adaptive weights for the adjacent nodes and thus differentiate the dissimilar neighbors in the aggregation process. Additionally, GNNs provably suffer from the over-smoothing [78] issue that the representation of the target central node in deep GNN layers may be over-smoothed by averaging the information from a too wide range of neighbors. To overcome this issue, we introduce residual connections and layer-wise aggregation [151, 75] to our GNN model. Lastly, we acknowledge the significant temporal impact of historical failures on the current state of pipes [31]. Consequently, we develop a temporal pattern learning module to complement the GNN module in capturing and leveraging temporal information encoded in historical failure records.

Overall, the work introduces novel elements to address challenges in real-world scenarios using the GNN, including the neighborhood construction by considering the geographical distribution of nodes, the attention mechanism to capture the dissimilarity

[1]There exist two critical pieces of information embedded in pipe networks, i.e., the structure of pipe connectivity and geographical neighboring effects. Regarding the pipe connectivity, industry practices indicate that when a pipe fails, additional failures are likely to occur on the pipes that are on the same route (connected to the failed pipe) due to physical effects such as water hammer [115]. In terms of geographical neighboring effects, nearby pipes, even those not directly connected to the failed pipe, may still exhibit a higher likelihood of failures due to shared environmental factors, including soil properties, ground vibrations, and so forth [11, 98]

of nodes with regard to their properties, and the residual connections to overcome the over-smoothing issue of the GNN. Our framework is evaluated on two real-world pipe network datasets. Experimental results show that it outperforms the statistical and machine learning models, as well as the GNN baselines. These results affirm our analysis of the challenges encountered in practical applications and demonstrate the effectiveness of the techniques employed to tackle these challenges.

1.2.4 A Summary of Contributions

The three studies collectively advance the field of GNNs by addressing distinct challenges and offering innovative solutions. The first study introduces VAGNN, a scalable framework overcoming limitations in existing GNNs by introducing a virtual adjacency matrix for flexible neighborhood set construction. Demonstrating linear computational complexity and validated through real-world experiments, VAGNN lays the groundwork for future advancements of GNNs in virtual adjacency matrix construction and weighted aggregation functions. The second study conducts an interpretive analysis of GNNs in link prediction, revealing their limitations in capturing pair-specific structural information represented by link heuristics like Common Neighbors. It also experimentally discovers that trainable node embeddings can effectively enhance GNN-based link prediction performance, especially in denser graphs. This study provides valuable insights into GNNs in link prediction, guiding the development of more robust link prediction algorithms. The third study addresses the challenges of GNNs in real-world pipe networks, presenting the Multi-hop Attention-based GNN (MAG) framework that integrates various techniques to effectively capture structural, geographical, and temporal information embedded in pipe networks. MAG outperforms statistical models, machine learning algorithms, and GNN baselines in real-world datasets, contributing crucial insights for applying GNN techniques to complex scenarios.

1.3 Thesis Organization

This thesis presents a comprehensive investigation of GNNs, which contributes to the growing body of knowledge in this field and demonstrates their potential across multiple domains by facilitating effective analysis, prediction, and decision-making in complex graph-structured data. The subsequent chapters are organized as follows:

- Chapter 2 provides the notations used in this thesis and a thorough literature survey of GNNs as well as related techniques.
- Chapter 3 introduces our VAGNN, a novel framework designed to enhance the generalization and scalability of GNNs with an optimizable virtual adjacency matrix.

- Chapter 4 explores the inner workings of GNNs in the link prediction task. By investigating the impact of various components such as node features, node embeddings, and heuristic information, we offer valuable insights into the strengths and limitations of GNNs for link prediction.
- Chapter 5 demonstrates our investigation into the roles played by different GNN techniques in real-world applications.
- Chapter 6 concludes this thesis.

Chapter 2

Literature Review

In today's interconnected world, data often comes in complex structures where relationships between entities play a crucial role in revealing insights. Graphs, a fundamental concept in mathematics and computer science, offer a powerful framework to model and analyze such relationships. Graph representation learning has emerged as a critical research area to effectively extract valuable information from graphs. This chapter provides a comprehensive overview of graph representation learning, with a special focus on GNNs [172] and other significant branches of graph representation learning methods, such as Matrix Factorization methods [65, 93], random walk methods [104, 43], graph autoencoders[139, 18, 133], and point process encoding approaches [178, 155].

2.1 Notations

In this thesis, we denote a *set* using a blackboard bold uppercase letter (e.g., $\mathbb{N}$), a *matrix* using a bold uppercase letter, a *vector* using a bold lowercase letter, and a *scalar* using a lowercase letter.

Let $\mathscr{G} = (\mathbb{V}, \mathbb{E}, \mathbf{X})$ denotes a graph $\mathscr{G}$ with N nodes, where $\mathbb{V}$ is the set of nodes, $|\mathbb{V}| = N$, $\mathbb{E}$ is the set of edges, and $\mathbf{X} \in \mathbb{R}^{N \times f}$ is the feature matrix for all nodes. $\mathbf{x}_i \in \mathbb{R}^f$ (i.e., the i-th row of $\mathbf{X}$) is the feature vector of a node v_i. A set of nodes connected directly to a node $v_i \in \mathbb{V}$ is the set of first-order or 1-hop neighbors of the node v_i.

The adjacency matrix is an $N \times N$ square matrix, storing the actual connections between nodes. It is denoted as $\mathbf{A} \in \mathbb{R}^{N \times N}$, in which the i, j-th entry (i.e., $a_{i,j}$) is 1 if an edge exists from node v_i to v_j, and 0 otherwise. The degree of node v_i is $\deg(i) = \sum_{v_j \in \mathbb{V}} a_{i,j}$. The degree of the graph $\mathscr{G}$ is the average degree of all nodes.

2.2 Graph Neural Networks

GNNs have emerged as a prominent paradigm in machine learning, showing remarkable success across numerous applications such as node classification [64, 136], link prediction [165, 160, 140], and graph classification [150, 19, 138]. Over the past years, significant advancements have been made in the development of various GNN architectures. This section aims to provide a detailed overview of notable GNN architectures.

2.2.1 Graph Convolutional Network

To begin, Graph Convolutional Network (GCN) [64] introduces a simple yet effective neighborhood aggregation mechanism for aggregating node features from neighboring nodes in the graph. An L-layer GCN is composed of L graph convolution layers. In each layer, the representation vector for each node is updated by aggregating the representations of that node itself and its neighbors from the previous layer. Such aggregation process iteratively updates the representations of nodes throughout the network. Formally, the l-th GCN layer can be expressed based on matrix multiplication as below:

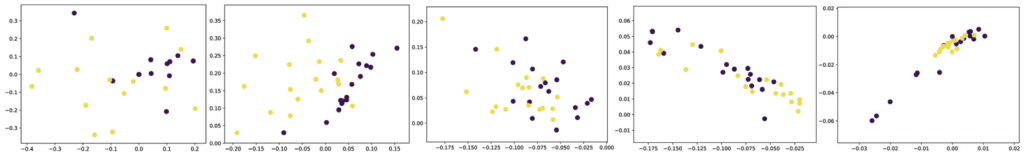
$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right), \tag{2.1}$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$, where $\mathbf{I}_N$ is an N-dimensional identity matrix. $\hat{\mathbf{A}}$ is the result from adding self-connection to each node in $\mathbf{A}$. $\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$ is a normalized version of the matrix $\hat{\mathbf{A}}$. $\mathbf{H}^{(l)} \in \mathbb{R}^{N\times d}$ and $\mathbf{H}^{(l+1)} \in \mathbb{R}^{N\times d'}$ are the input and output node representations in the l-th layer of a GCN. The i-th row of $\mathbf{H}$ is denoted as $\mathbf{h}_i \in \mathbb{R}^d$ which is a d-dimensional representation of the node v_i. $\mathbf{H}^{(0)} = \mathbf{X}$ is initialized with the input feature matrix. $\mathbf{W}^{(l)} \in \mathbb{R}^{d\times d'}$ is a trainable weight matrix. $\sigma(\cdot)$ is an activation function.

Graph Laplacian The form $\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{H}^{(l)}$ in Equation 2.1 corresponds to propagating a weighted average of the representations of each node and its 1-hop neighbors to that node. In essence, the work of Li et al [78] has shown that the convolutional operation of spectral GCN is a special form of Laplacian smoothing, i.e., symmetric Laplacian smoothing. Specifically, given the adjacency matrix with self-loop $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ for a graph, the Laplacian smoothing [130] on each channel of the input features is formulated by

$$\hat{\mathbf{y}}_i = (1-\gamma)\mathbf{h}_i + \gamma\sum_j \frac{\hat{a}_{ij}}{d_i}\mathbf{h}_j \tag{2.2}$$

where $1 \le i \le N$, $1 \le \gamma \le 1$, d_i is the degree of node v_i. γ is a parameter that controls the weighting between the representation of the current node $\mathbf{x}_i$ and its neighbors' representa-



(a) 1-layer (b) 2-layer (c) 3-layer (d) 4-layer (e) 5-layer

Figure 2.1 Over smoothing when the number of graph convolution layers increases. The points are the node embeddings of Zachary's karate club graph with GCNs. Figure is extracted from Li et al [78].

tions $\mathbf{h}_j$. The Laplacian smoothing in Equation 2.2 can be written in matrix form:

$$\hat{\mathbf{Y}} = \mathbf{H} - \gamma\hat{\mathbf{D}}^{-1}\hat{\mathbf{L}}\mathbf{H} = (\mathbf{I}_N - \gamma\hat{\mathbf{D}}^{-1}\hat{\mathbf{L}})\mathbf{H} \tag{2.3}$$

where $\hat{\mathbf{L}} = \hat{\mathbf{D}} - \hat{\mathbf{A}}$. Particularly, the standard Laplacian smoothing is in the form of $\hat{\mathbf{Y}} = \hat{\mathbf{D}}^{-1}\hat{\mathbf{A}}\mathbf{X}$, which is the case of $\gamma = 1$ in Equation 2.3, i.e., $\hat{\mathbf{Y}} = (\mathbf{I}_N - \hat{\mathbf{D}}^{-1}\hat{\mathbf{L}})\mathbf{X} = (\mathbf{I}_N - \hat{\mathbf{D}}^{-1}(\hat{\mathbf{D}} - \hat{\mathbf{A}}))\mathbf{X} = \hat{\mathbf{D}}^{-1}\hat{\mathbf{A}}\mathbf{X}$. The form $\hat{\mathbf{D}}^{-1}\hat{\mathbf{L}}$ is called the normalised Laplacian. The $\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}$ in Equation 2.1 is a symmetric normalised Laplacian $\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{L}}\hat{\mathbf{D}}^{-\frac{1}{2}}$ when $\gamma = 1$.

Over-smoothing Issue Given a GCN model, the first graph convolution layer preserves the first-order proximity while the following convolution layers of GCN can capture high-order proximity. However, a GCN model cannot include too many graph convolution layers since it suffers from the over-smoothing issue as the number of graph convolution layers increases [78].

Figure 2.1 shows a toy experiment in the work of Li et al [78]. The GCN models with different numbers of convolution layers are trained on the Zachary's karate club data [162] which contains two classes of 34 nodes and 78 edges. The input feature vector is a one-hot vector for each node. The nodes are plotted as points in Fig. 2.1, where the color of points indicates the class of nodes. It can be seen that the points are not well separated when the GCN has only 1 graph convolutional layer, and GCN with 2 convolutional layer can separate the points into two classes relatively well, while the points are mixed quickly with the increasing number of graph convolutional layers. Li et al [78] prove that the representations of nodes within each linked subgraph will converge to the same values when repeatedly applying Laplacian smoothing many times. For symmetric normalized Laplacian, the convergence is proportional to the square root of the node degree.

Aggregation-based GNNs GCN serves as a foundational architecture for many subsequent GNN models. As the GNN landscape has evolved, a multitude of architectural variations have emerged. Remarkably, the majority of these GNN architectures adhere to a common framework centered around the scheme of the neighborhood aggregation algorithm [27]. Such the algorithm, which lies at the heart of GNNs' functionality, can be

formalized as follows[1]:

$$\mathbf{h}_i^{(l+1)} = \text{AGGREGATE}^{(l)}\left(\left\{\mathbf{h}_j^{(l)} \mid v_j \in \mathbb{N}(v_i) \cup \{v_i\}\right\}\right), \tag{2.4}$$

where $\mathbb{N}(v_i)$ is the set of neighboring nodes of node v_i. The function AGGREGATE$(\cdot)$ defines the aggregation operation over the set of representations $\left\{\mathbf{h}_j^{(l)} \mid v_j \in \mathbb{N}(v_i) \cup \{v_i\}\right\}$. The formulation of AGGREGATE$(\cdot)$ varies across different GNN models. Indeed, the innovations of most existing GNNs center around two key aspects: the methods for constructing the neighborhood set $\mathbb{N}(v_i)$ and the specific method of AGGREGATE$(\cdot)$. In the following, we present an overview of these existing GNN architectures.

2.2.2 Graph Attention Network

Attention mechanisms have been incorporated into GNNs to dynamically weigh the importance of neighboring nodes during the aggregation process, allowing the model to focus more on relevant nodes while downplaying the influence of less relevant ones. The attention-based GNNs can lead to more effective representation learning and better generalization to complex graph-structured data.

Among the various advancements in the field, Graph Attention Network (GAT) [136] stands out as a widely acclaimed architecture. The GAT architecture consists of multiple attention heads, each computing its own attention coefficients. The outputs from different heads are then concatenated and linearly transformed to produce the final node embeddings. GAT processes each node's neighbors with attention weights, enabling it to adaptively weigh the importance of different neighbors for each node. The core innovation of GAT lies in its approach to computing attention coefficients. Formally, in the l-th layer of the m-th head, the attention coefficient between nodes v_i and v_j can be calculated based on their representations as below:

$$\text{ATTN}\left(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}\right) = \sigma\left(\mathbf{q}^{(l,m)} \cdot \left(\text{CONCAT}\left(\mathbf{h}_i^{(l)}\mathbf{W}_{\text{Q}}^{(l,m)}, \mathbf{h}_j^{(l)}\mathbf{W}_{\text{Q}}^{(l,m)}\right)\right)^{\top}\right), \tag{2.5}$$

where $\sigma(\cdot)$ is an activation function, the $\cdot^{\top}$ represents transposition, CONCAT$(\cdots)$ is the concatenation operation, $\mathbf{W}_{\text{Q}}^{(l,m)} \in \mathbb{R}^{d \times d_{\text{A}}}$ is a trainable weight matrix shared across all nodes, and $\mathbf{q}^{(l,m)} \in \mathbb{R}^{2d_{\text{A}}}$ is a weight vector. Note that $\mathbf{W}_{\text{Q}}^{(l,m)}$ transforms the representations of nodes from the dimension d into d_{A}, which offers the advantage of optimizing computer memory usage through assigning a small value to the hyper-parameter d_{A} when computing attention on large graphs.

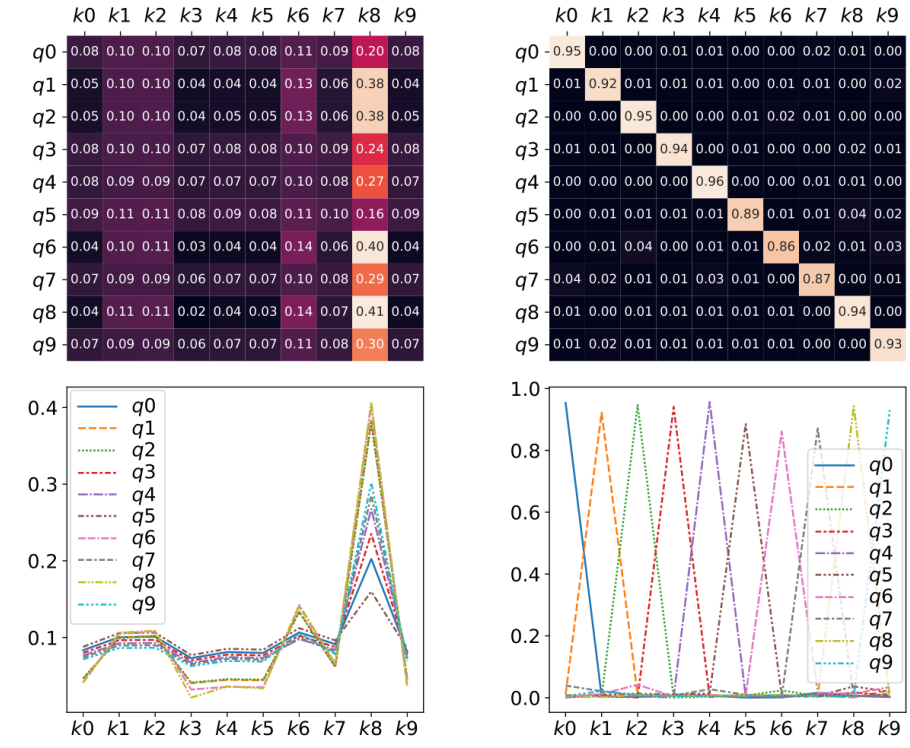
Despite the success of GAT, Brody et al. [14] reveal a limitation in GAT in terms of the attention mechanism. Their findings reveal that GAT employs a restricted "static" form

[1] For simplicity, we omit the residual connections, activation functions, etc.

of attention. In this form, the attention function for any given target (query) node exhibits monotonic behavior with respect to the scores of its neighbors (keys). As illustrated in Fig. 2.2, the ranking order of attention coefficients remains consistent across all nodes within the graph, irrespective of the query node. This fact significantly constrains the expressiveness of GAT.

The primary issue in the standard GAT scoring function (Equation 2.5) is that the learned weights $\mathbf{q}^{(l,m)}$ and $\mathbf{W}_{\mathrm{Q}}^{(l,m)}$ are applied consecutively, resulting in the collapse into a single linear layer. To address this limitation, Brody et al. [14] propose a straightforward yet effective modification. They advocate for applying the $\mathbf{q}^{(l,m)}$ post nonlinearity ($\sigma(\cdot)$), leading to GATv2. Consequently, the attention computation is reformulated as:

$$\mathrm{ATTN}\left(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}\right) = \mathbf{q}^{(l,m)} \sigma\left(\left(\mathrm{CONCAT}\left(\mathbf{h}_i^{(l)}\mathbf{W}_{\mathrm{Q}}^{(l,m)}, \mathbf{h}_j^{(l)}\mathbf{W}_{\mathrm{Q}}^{(l,m)}\right)\right)^{\top}\right), \tag{2.6}$$



(a) Attention in standard GAT

(b) Attention in GATv2

Figure 2.2 The standard GAT (left) computes static attention - the ranking of attention coefficients is global for all nodes in the graph, and is unconditioned on the query node. In contrast, GATv2 (right) can actually compute dynamic attention, where every query has a different ranking of attention coefficients of the keys. This figure is from the work of [14].

2.2.3 Addressing the Scalability Challenges in GNNs

With the goal of addressing the scalability challenges of GNNs and training deeper GNNs on large-scale graphs, a spectrum of innovative techniques has emerged. These techniques significantly reduce computational complexity while preserving the representation quality, thus enabling GNNs to handle real-world graphs more effectively.

GraphSAGE GraphSAGE [47] is a pioneer work in this regard. Instead of using all first-order neighboring nodes of each node in GCN [64], it employs an innovative sampling-based strategy to select a fixed-size neighborhood. This strategy can be implemented through various sampling techniques, such as random walk-based sampling or uniform sampling. The fixed-size neighborhood allows GraphSAGE to control the computational footprint of each batch and thereby efficiently manage the computational demands of each processing batch. This attribute of GraphSAGE proves to be of paramount significance, which paves the way for applying GNNs to large-scale graph analysis.

On the other hand, GraphSAGE offers an array of aggregation methods, including MEAN-pooling, MAX-pooling, and LSTM aggregator [53]. This diversified aggregation toolkit equips practitioners with the flexibility to tailor their approach to the specific demands of their graph data, enhancing the adaptability and performance of the model across diverse scenarios.

GraphSAINT GraphSAINT [164] introduces a novel sampling strategy for training deep GNNs on large graphs. The primary objective of GraphSAINT is to tackle the critical challenges of scalability and computational efficiency that arise when training GNNs on large graphs containing millions or even billions of nodes and edges. Instead of conventional approaches of building a GNN [64] on the entire graph, GraphSAINT samples subgraphs from the entire graph first in the training. Naturally, GraphSAINT resolves “neighbor explosion” for the neighborhood aggregation of each node in GNN training, since the subgraph is a small yet complete graph that contains all the necessary contextual information of nodes for accurate GNN training.

However, the subgraph sampling training method in GraphSAINT also brings new challenges. It’s intuitive that nodes with a higher influence on each other should have a greater likelihood of being included in the same subgraph. This allows these sampled nodes to mutually "support" each other within the confines of the training process. As a result, this strategy leads to non-uniform node sampling probabilities, introducing bias into the learning node representations. To address this potential bias, GraphSAINT incorporates normalization techniques into its training process. These techniques ensure that the feature learning process doesn’t favor nodes that are more frequently sampled. Furthermore, the authors of GraphSAINT conduct a variance reduction analysis to enhance training quality.

This involves designing lightweight sampling algorithms that quantify the "influence" of neighboring nodes, thereby improving the overall quality of the training.

Table 2.1 Random partition versus clustering partition of the graph on the GNN performance. Clustering partition leads to better performance (in terms of test F1 score) since it removes fewer between-partition links. Results are from the work of Cluster-GCN [25].

Dataset	random partition	clustering partition
Cora [116]	78.4	82.5
Pubmed [76]	78.9	79.9
PPI [28]	68.1	92.9

Cluster-GCN Cluster-GCN [25] presents a novel methodology to enhance the efficiency and effectiveness of training deep GNNs on large-scale graphs. The fundamental innovation of Cluster-GCN lies in its utilization of a graph clustering strategy to divide the graph into smaller clusters (subgraphs). In contrast to techniques that extract subgraphs like GraphSAINT, Cluster-GCN employs graph clustering algorithms such as Metis [61] and Graclus [29]. These algorithms are designed to partition the graph's vertices, with the primary goal of generating clusters where within-cluster links are more numerous than between-cluster connections.

Two key motivations underline the choice of utilizing graph clustering. Firstly, the efficient learning of node representations in each batch corresponds to leveraging the within-cluster links. Given that nodes and their neighbors often reside within the same cluster, local information propagation can be more efficient. Secondly, since prediction errors tend to be influenced by between-cluster links, it's essential to minimize these links in order to reduce prediction inaccuracies.

The authors provide empirical evidence to support the efficacy of their approach. In one experiment, they compare two different node partitioning strategies: random partitioning versus clustering-based partitioning. The graph is divided into 10 segments using both random and METIS-based partitioning, and each partition is used as a batch for GNN training. The results are shown in Table 2.1. It demonstrates that, under the same number of training epochs, clustering partitioning leads to higher accuracy. This underscores the importance of utilizing graph clustering methods over random partitioning.

However, the paper acknowledges two potential challenges within the Cluster-GCN framework. Firstly, when the graph is partitioned, certain links might be removed, potentially impacting performance. Secondly, graph clustering algorithms tend to group similar nodes together, which could alter the distribution of a cluster from that of the original dataset. This alteration can result in biased learning of node representations.

To address these issues, Cluster-GCN introduces a stochastic multiple clustering approach by incorporating multiple clusters (subgraphs) together in a training batch. This approach is aimed at incorporating between-cluster links back into the training process to mitigate the removal of important links during partitioning. Moreover, it helps in reducing variance across subgraphs, ensuring that the cluster-based gradient estimations are more representative of the global gradient. This innovation enhances the algorithm's robustness and accuracy, leading to more reliable GNN training results on large-scale graphs.

To validate the efficacy of the stochastic multiple clustering approach, the authors conducted an empirical experiment on the Reddit dataset. The empirical results in Fig. 2.3 demonstrate that the utilization of multiple clusters within a single batch substantially enhances the convergence performance compared to the use of one cluster as a batch.

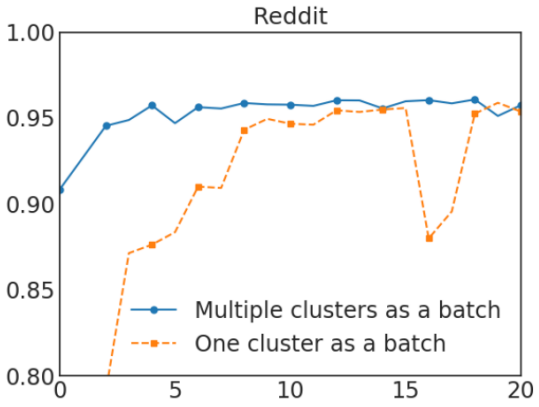


Figure 2.3 Performance comparisons of using one cluster versus multiple clusters. The former approach uses 300 partitions, while the latter involves the utilization of 1500 partitions, with a random selection of 5 partitions to compose each batch. The results present the epoch (x-axis) versus the F1 score (y-axis). This figure is from the work of Cluster-GCN [25].

More Methods In addition to the aforementioned models, the landscape of addressing scalability challenges for training deep GNNs on large-scale graphs has been further enriched by various innovative approaches. With the utilization of Monte Carlo approaches, FastGCN [21] introduces a more economic sampling scheme that saves computational resources compared to other methods like GraphSAGE [47], achieving orders of magnitude faster training while maintaining comparable prediction accuracy. SHADOW [163] employs a design principle that decouples the depth and scope of GNNs to address the scalability limitations of GNNs. It starts by extracting a localized subgraph as the scope, which contains critical neighboring nodes while excluding irrelevant ones. Subsequently, a deep GNN is employed to process this subgraph. These innovative methodologies collectively contribute to a growing toolkit of solutions for enhancing the scalability of GNNs, catering to the demands of processing information within large graphs.

2.2.4 High-hop GNNs

Most traditional GNNs, often relying on 1-hop message passing, are limited in their information aggregation capabilities. To address this limitation, several GNNs have been designed to leverage high-hop neighbors, allowing for better information incorporation across multiple hops.

MixHop MixHop [1] is a novel approach for effectively combining and aggregating feature representations of neighboring nodes at multiple distances. The primary innovation of MixHop lies in its utilization of linear combinations of aggregation results from various neighborhood orders. By doing so, MixHop can capture a wider range of relational information, leading to improved representation learning. Remarkably, MixHop achieves this without introducing any additional memory or computational complexity.

The architecture of MixHop achieves the high-hop GNN capability by performing multiple GCN operations, where each operation utilizes a modified adjacency matrix obtained by applying different powers to the original adjacency matrix. This modification enables the model to consider node relationships across multiple distances simultaneously. Mathematically, the MixHop architecture can be formalized as follows:

$$\mathbf{H}^{(l+1)} = \text{CONCAT}\left\{\sigma\left(\tilde{\mathbf{A}}^{p}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)|p \in P\right\}, \tag{2.7}$$

where $\text{CONCAT}(\cdot)$ is the concatenation operation. $\tilde{\mathbf{A}}$ is the normalized version of the adjacency matrix. P is a hyper-parameter that is a set of integer adjacency powers, $\tilde{\mathbf{A}}^{p}$ represent $\tilde{\mathbf{A}}$ multiplied by itself p times. $\tilde{\mathbf{A}}^{0}$ is the identity matrix $\mathbf{I}_N$. These powers determine the distances of neighboring nodes to be considered. For example, $\tilde{\mathbf{A}}^{1}$ and $\tilde{\mathbf{A}}^{2}$ involve the neighboring nodes within 1-hop and 2-hop distances, respectively.

KPGNN KPGNN [35], short for "*K*-hop *P*eripheral-subgraph-enhanced *GNN*", is a novel high-hop GNN architecture. This architecture is designed to enhance the capabilities of K-hop aggregation-based GNNs by incorporating peripheral subgraph information, resulting in more powerful and expressive GNN models.

In the context of KPGNN, the term "K-hop" refers to the notion of considering nodes within a certain distance (or number of hops) from a central node. The "P" stands for "Peripheral-subgraph-enhanced," indicating that the architecture leverages peripheral subgraph information to augment its neighborhood aggregation process. Peripheral subgraph refers to the subgraph formed by the neighboring nodes within the K-hop distance from the central node. The key innovation of KPGNN lies in its neighborhood construction. While conventional K-hop GNNs aggregate information solely from neighboring nodes within each hop, KPGNN takes a step further by also aggregating information from the peripheral

subgraph induced by the neighbors within the same hop. These additional neighbors enable KPGNN to capture more intricate local structural features around the central node, leading to more expressive node representations.

An important insight is that KPGNN is capable of distinguishing distance regular graphs when equipped with an appropriate encoder for the peripheral subgraph. This showcases the model's potential to uncover and leverage the inherent structural properties of different types of graphs.

KPGNN offers several advantages in terms of its practical applicability and efficiency. Firstly, it can be seamlessly applied to existing high-hop GNN architectures with only minor modifications. This ensures that the incorporation of peripheral subgraph information is adaptable to a wide range of GNN variants. Secondly, despite the enhanced capabilities, the computational complexity added by KPGNN to the standard high-hop GNNs is minimal, maintaining its feasibility for real-world applications.

The effectiveness of KPGNN are validated through rigorous experimental evaluations. The results underscore the improved expressive capacity of KPGNN, showcasing its adeptness at learning and representing local structural features surrounding individual nodes. With the peripheral-subgraph-enhanced approach, KPGNN offers a promising pathway to advancing the capabilities of GNNs in capturing nuanced graph properties, thereby contributing to the broader field of graph representation learning and analysis.

Additional High-hop GNNs Several noteworthy high-hop GNN architectures have been proposed, each introducing unique mechanisms to enhance the modeling capabilities of GNNs across multiple hops in a graph.

DEGNN [77] proposes a series of GNN architectures based on Distance Encoding (DE). By leveraging the distance between nodes. DE can be incorporated into GNNs as extra node features or as controllers of neighborhood aggregation process. By incorporating distance information, DEGNN enables the utilization of high-hop neighbors, allowing the model to capture long-range dependencies and structural patterns. The utilization of DE enriches the GNN's understanding of the graph's topology and fosters improved modeling of complex relationships.

F-MPNN [10] explores the use of local graph parameters in GNNs to incorporate higher-order graph structural information. These local graph parameters are computationally inexpensive to compute and provide valuable information about higher-order structural characteristics of the graph. F-MPNN allows any GNN architecture to incorporate these parameters, enhancing its ability to capture intricate graph patterns and relationships. Tensorized-GNN [55] is an expressive GNN architecture that employs tensor decomposition to capture high-order nonlinear interactions among nodes. By leveraging the symmetric CP decomposition, Tensorized-GNN efficiently parameterizes permutation-invariant multilinear maps for modeling node interactions.

2.2.5 Graph Transformers

Following the success of the Transformer architecture in capturing contextual relationships within sequential data [135, 30], recent efforts have been made to extend the Transformer's capabilities to the domain of GNNs.

The heart of the Transformer is the self-attention mechanism. It allows the model to weigh the importance of different nodes (words) in a graph (sequence) relative to a specific node (word), capturing relationships between nodes (words) regardless of their distance apart. It involves three types of vectors: Query (Q), Key (K), and Value (V). These vectors are used to compute attention scores, which determine how much each node (word) attends to other nodes (words). The scores are then used to weigh the values, producing a weighted sum that represents the attended context for each node (word). The standard Transformer architecture consists of multiple heads of attention modules. It means that the self-attention mechanism is used multiple times in parallel, each with its own set of learned weight matrices. These parallel attention heads allow the model to focus on different aspects of the input data, enabling it to capture various relationships and patterns among nodes (words).

We start by describing a single head of self-attention module. Let the representations of N nodes in the l-th layer be $\mathbf{H}^{(l)} \in \mathbb{R}^{N\times d}$. The i-th row of $\mathbf{H}^{(l)}$ is denoted as $\mathbf{h}_i^{(l)} \in \mathbb{R}^d$ that is a d-dimensional representation of node v_i. Following the setting of Query, Key and Value in the Transformer [135], three weight matrices are used, i.e., $\mathbf{W}_Q^{(l)} \in \mathbb{R}^{d\times d_A}$, $\mathbf{W}_K^{(l)} \in \mathbb{R}^{d\times d_A}$ and $\mathbf{W}_V^{(l)} \in \mathbb{R}^{d\times d'}$, where d_A is the dimension of each node's representation used for calculating self-attention coefficients and d' is the dimension of output representation of each node. This firstly leads to the following formulations:

$$\mathbf{Q}^{(l)} = \mathbf{H}^{(l)}\mathbf{W}_Q^{(l)}, \mathbf{K}^{(l)} = \mathbf{H}^{(l)}\mathbf{W}_K^{(l)}. \tag{2.8}$$

The self-attention matrix $\mathbf{A}^{(l)}$ is subsequently calculated by:

$$\mathbf{A}^{(l)} = \frac{\mathbf{Q}^{(l)}{\mathbf{K}^{(l)}}^\top}{\sqrt{d_A}}. \tag{2.9}$$

Lastly, the output of this self-attention module is expressed as:

$$\mathbf{H}^{(l+1)} = \text{softmax}\left(\mathbf{A}^{(l)}\right)\mathbf{H}^{(l)}\mathbf{W}_V^{(l)}. \tag{2.10}$$

Graphormer Graphormer [156], built directly upon the standard Transformer, has emerged as a pioneering solution for effectively adapting Transformers for graph data. The key innovation introduced by Graphormer centers around the necessity to integrate the inherent structural properties of graphs into the model. To this end, Graphormer proposes

a series of straightforward yet effective structural encoding techniques, which enhances the model's capacity to capture and interpret graph-structured data.

Specifically, Graphormer proposes the concept of Centrality Encoding in Graphormer to differentiate the node importance in the graph. Notably, nodes within a graph can exhibit varying degrees of importance, such as the distinction between influential celebrities and the general population within a social network. However, these importance gradients are not inherently reflected in the conventional self-attention mechanism, which predominantly relies on the semantic attributes of nodes for similarity calculations. To overcome this limitation, Graphormer innovatively incorporates node centrality encoding. Specifically, Graphormer employs degree centrality as the basis for this encoding, assigning a learnable vector to each node according to its degree, which is subsequently integrated into the node features at the input layer. Empirical evaluations convincingly demonstrate the efficacy of this simple yet impactful centrality encoding in enhancing the Graphormer's expressiveness for modeling graph-oriented data.

In addition, Graphormer introduces a novel Spatial Encoding mechanism designed to capture the underlying structural relationships between nodes. One notable geometric property that distinguishes graph-based data from other structured data domains, such as language and images, is the absence of a canonical grid for embedding the graph. Indeed, nodes are represented in a non-Euclidean space and are interconnected by edges. To effectively leverage such structural information, Graphormer pioneers the assignment of learnable embeddings based on the spatial relations between any two nodes. Graphormer employs the shortest path distance as a demonstrative metric, embedding it as a bias term within the softmax attention mechanism. This innovative approach facilitates the accurate capture of spatial dependencies among nodes within the graph.

Furthermore, the authors acknowledge the potential presence of supplementary spatial information in edge features, such as the bond types between atoms in molecular graphs. In response, a novel edge encoding method is designed, wherein Graphormer computes an average of dot products between edge features and learnable embeddings along the shortest path. This computed value is then utilized in the attention module. Armed with these encoding techniques, Graphormer substantially enhances its capability to model relationships between nodes and faithfully represent graph structures.

Moreover, the authors rigorously characterize the expressive capability of Graphormer, showcasing how various prevalent GNN variants can be regarded as special cases within the broader framework of Graphormer. This mathematical exposition underscores the unifying potential of Graphormer in encapsulating and surpassing existing graph-based modeling techniques.

Structure-Aware Transformer (SAT) SAT [19] stands as a remarkable advancement within the domain of graph Transformers. This innovative approach addresses a critical

shortcoming of the self-attention mechanism inherent to node representations: the insufficient capture of structural similarities. The core strength of SAT lies in its integration of structural insights, achieved by extracting a subgraph representation rooted at each node prior to attention computation.

Notably, the nature of SAT aligns with previous findings as exemplified in Mialon et al.'s investigation [94]. In their work, the authors establish a connection between self-attention within the Transformer and a trainable kernel—a kernel constructed over node features. Specifically, Equation 2.13 can be rewritten as follows:

$$\text{Attn}(\mathbf{h}_v) = \sum_{u \in \mathbb{V}} \frac{\kappa(\mathbf{h}_v, \mathbf{h}_u)}{\sum_{w \in \mathbb{V}} \kappa(\mathbf{h}_v, \mathbf{h}_w)}, \tag{2.11}$$

where $\kappa(\cdot,\cdot)$ a kernel parameterized by $\mathbf{W}_Q$ and $\mathbf{W}_K$:

$$\kappa(\mathbf{h}_v, \mathbf{h}_u) = \frac{\mathbf{h}_v \mathbf{W}_Q, (\mathbf{h}_u \mathbf{W}_k)^\top}{\sqrt{d_A}} \tag{2.12}$$

The significant contribution of SAT [19] lies in its recognition that the aforementioned kernel (Equation 2.11) primarily captures attributed similarities among node pairs. However, this kernel might fail to differentiate structurally different nodes with similar or identical node features. To overcome this limitation and incorporate similarity, SAT introduces a generalized kernel. This novel kernel incorporates local substructures around each node, thereby broadening the scope of structural context. In light of this, the structure-aware attention function in SAT is defined as follows:

$$\text{Attn}(\mathbf{h}_v) = \sum_{u \in \mathbb{V}} \frac{\kappa_g(\mathscr{G}_v, \mathscr{G}_u)}{\sum_{w \in \mathbb{V}} \kappa(\mathscr{G}_v, \mathscr{G}_w)}, \tag{2.13}$$

where $\mathscr{G}_v$ represents a subgraph in the graph G centered around node v. Notably, the kernel function $\kappa_g(\cdot,\cdot)$ offers versatility in its design, allowing for diverse comparisons between pairs of subgraphs. This novel self-attention mechanism not only captures attribute-based similarities but also effectively learns structural similarities among subgraphs, consequently yielding more expressive node representations than traditional self-attention mechanisms.

More Graph Transformers Furthermore, the extension of the Transformer mechanism into the realm of graph analysis has led to the development of various innovative approaches. Graph Transformer[32] serves as a prime example of how the foundational Transformer concept can be tailored for graph data. In this adaptation, Laplacian eigenvectors are employed as a fundamental encoding method. Attention computation is focused on the immediate neighborhood of each node, in contrast to the conventional approach of considering the entire graph. An advancement in this direction is seen in the Self-Attention

Network (SAN) [67], which utilizes Laplacian eigenvectors for absolute encoding. It distinguishes itself from the Graph Transformer [32] by conducting attention computations that span the entirety of the graph, rather than limiting attention to local neighborhoods.

The incorporation of both absolute and relative encodings has become a common strategy in graph Transformer methods. Relative encoding, initially proposed in the context of the vanilla Transformer by Shaw et al. [121], is an additional layer that augments the encoding process. Unlike absolute encoding, which is applied once to input node features, relative encoding integrates information about the relative positions or distances between nodes into the self-attention mechanism. Mialon et al. [94] have further expanded on this concept by introducing relative encoding through kernels. These kernels bias the self-attention calculation, effectively infusing positional information into the Transformer through kernel selection. Subsequent advancements have continued to push the boundaries of this field by encoding structural attributes, such as centrality metrics and shortest path distances, into positional representations [156], or by leveraging GNNs to learn representations of the underlying graph structure [110, 147, 94].

In summary, the expansion of the Transformer mechanism to analyze graph data has yielded a diverse array of methodologies. These methodologies span from variations in encoding strategies—utilizing both absolute and relative encodings—to integrating graph structural properties via innovative techniques like learning representations of subgraphs. This vibrant landscape of advancements showcases the continuing evolution and growing versatility of graph Transformers.

2.2.6 More Noteworthy GNNs

In addition to the previously discussed GNN models, we highlight several pioneering contributions that have significantly advanced the field of GNNs. These works have played a pivotal role in shaping the landscape of GNN research and development.

Graph Isomorphism Networks (GIN) [150] introduces a highly expressive model architecture by aggregating features using permutation-invariant functions. This approach empowers GIN to capture intricate local and global graph structures with remarkable efficiency. Graph U-Net [38] , drawing inspiration from conventional U-Net architectures in computer vision [111], adopts an encoder-decoder design to cater to graph-based applications. By incorporating skip connections, this model effectively preserves both local and global graph characteristics, enhancing its capability to handle complex graph data.

JKGNN [151] adaptively incorporates different neighborhood ranges for each node with the novel jump connections, resulting in improved representation. GCNII [22] utilizes initial residual connections and identity mapping at each layer. This unique approach effectively mitigates the issue of over-smoothing that tends to arise with increasing network

depth, ensuring the model's sustained efficacy. DeeperGCN [75] introduces pioneering concepts of residual connections and message normalization layers. These novel components serve to facilitate the training of deeper GNNs, enabling the model to delve into more intricate graph structures while maintaining optimal performance. AirGNN [88] introduces a node-level adaptive transition that seamlessly combines feature aggregation and residual connection. It highlights the limitations of directly aggregating single-node representations obtained by GNNs and introduces the concept of the "labeling trick" that unifies previous successful methods for multi-node representation learning. PEG [140] is a new class of GNN layer that aims to address the limitations of GNNs in accurately predicting tasks based on sets of nodes. It leverages positional encoding techniques to incorporate positional features of nodes. PEG achieves permutation equivariance with respect to the original node features and rotation equivariance with respect to the positional features.

Recognizing the inherent diversity in graph structures, researchers have focused on developing tailored GNN architectures that cater to specific graph tasks. These specialized architectures demonstrate superior performance by effectively leveraging domain-specific characteristics. Neo-GNN [160] stands out by effectively estimating overlapped neighborhoods and handling multi-hop neighborhoods, capturing the structural information required for accurate link prediction. Heterogeneous GNNs [89] address graph analysis on heterogeneous information networks by extending GNNs to handle different types of nodes and edges with varying semantics.

Graph HyperNetwork [165] is a method for neural architecture search (NAS) on graph-structured data. It utilizes a hypernetwork to generate architectural parameters, allowing for the automatic design and optimization of GNN architectures. TFGW [138] proposes a design of a GNN layer, which computes Frechet Graph Wasserstein (FGW) distances between an input graph and learnable graph templates. The method enables simultaneous learning of GNN preprocessing layers and graph templates. GraphMix [54] introduces a graph-level augmentation technique to enhance GNN training. It combines data from multiple graphs by mixing node features and labels, leading to improved generalization and robustness. NBFNet [176] defines node pair representations as a sum of path representations, with each path representation being a product of edge representations. This innovative approach enriches the model's ability to capture intricate node relationships and contributes to improved accuracy in graph-related tasks.

2.3 Other Graph Representation Learning Methods

2.3.1 Graph Autoencoders

With the success of neural networks, which have demonstrated a remarkable capacity for capturing intricate information from data [12, 68, 126, 50], auto-encoder approaches have emerged as promising tools for representation learning. An auto-encoder [51] is a feed-forward neural network that contains two essential components: *encoder* and *decoder*. The encoder takes a vector $\mathbf{v} \in \mathbb{R}^N$ as input and feed forward it through several layers producing $\mathbf{e} \in \mathbb{R}^d$, and then the decoder continues to feed forward $\mathbf{e}$ through several layers to generate $\hat{\mathbf{v}} \in \mathbb{R}^N$. The decoder actually aims to reconstruct $\mathbf{v}$. The training of the auto-encoder is achieved by minimizing the reconstruction loss $||\mathbf{v} - \hat{\mathbf{v}}||$, where $\mathbf{e}$ can be considered as an embedding vector of the original vector $\mathbf{v}$.

In the realm of graph representation learning, the auto-encoder technique has found application through various approaches [139, 18, 133]. Wang et al. [139] introduce the Structural Deep Network Embedding (SDNE) technique. Given the adjacency matrix $\mathbf{A}$ of a graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$, SDNE takes as input a vector $\mathbf{a}_i \in \mathbb{R}^{|\mathbb{V}|}$, where $\mathbf{a}_i$ corresponds to the i-th row of the adjacency matrix $\mathbf{A}$ and describes the neighboring connectivity of node v_i. SDNE produces $\mathbf{e}_i \in \mathbb{R}^d$ by the encoder and feed $\mathbf{e}_i$ into the decoder to reconstruct $\mathbf{a}_i$. After the model optimization, the vector $\mathbf{e}_i$ can be construed as an effective embedding of the nodes v_i. Cao et al. [18] proposed a similar approach to SDNE. It computes a high-order proximity matrix $\mathbf{S} \in \mathbb{R}^{|\mathbb{V}| \times |\mathbb{V}|}$. Tran [133] proposed an auto-encoder that can take into account the node attributes by concatenating the attribute values $\mathbf{x}_i$ of node v_i to $\mathbf{a}_i$.

2.3.2 Random Walk Methods

A family of pioneering techniques within the realm of graph representation learning revolves around the utilization of random walk-based methodologies. In this category of methods, the input is primarily composed of nodes and links, with only limited reliance on node attributes. Inspired by the conceptual framework of word embeddings within the domain of natural language processing [95], a suite of techniques, including DeepWalk [104] and node2vec [43], have emerged to harness the power of node embeddings. In this context, given a graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$, a random walk is a sequence of nodes $v_1, \cdots, v_i, \cdots, v_l$ where $v_i \in \mathbb{V}$, where l is the length of the walk, node v_i and node v_{i+1} are connected, the distances between nodes in this sequence are greater than zero, i.e., $s_{i,j} > 0$ for $1 \leq i \leq l-1$. A random walk can be generated along a path that starts at a node v_i, then transitions to its neighbor v_j, then transitions to a neighbor of v_j and ends after l steps. The initial node and the following steps on this path can be selected based on some strategies, such as a uniformly random scheme.

Akin to the principles underpinning natural language processing, random walk-based techniques generate multiple such random walks across a graph. In this analogy, nodes can be likened to words, a walk can be deemed analogous to a sentence, and the set of all nodes in a graph corresponds to the lexical dictionary of words. This parallel enables the adoption of techniques analogous to those used in word embedding approaches, like Skip-Gram [95], to effectively learn node embeddings.

These strategies are typically unsupervised and leverage shallow neural networks for model training. DeepWalk [104], the pioneering work in this domain, synergistically blends random walk principles with language modeling methodologies to learn node embeddings. The initial node selection and transition sequencing in the walk are governed by uniform random choices. DeepWalk adeptly captures higher-order proximity, encompassing the last k vertices and the next k vertices centered around node v_i in the context of the random walk. Notably, k is a hyperparameter that abides by the condition $2k+1 <= l$. Building upon this foundation, subsequent studies [43, 107] have introduced further refinements. Specifically, node2vec [43] incorporates a random walk strategy that strikes a balance between depth-first exploration to comprehend community structure and breadth-first traversal to capture localized information.

Several works have extended the random walk-based graph representation learning from static graph to dynamic graph. One such approach, outlined by Mahdavi et al. [90], involves adapting random walks to incorporate new events occurring within a graph. Specifically, given a series of graph snapshots $\{\mathscr{G}_1,\ldots,\mathscr{G}_T\}$ representing a dynamic graph, the set of random walks for the t-th snapshot consists of two sets. One set contains the common random walks in $(t-1)$-th and t-th snapshots. Another set contains the newly generated walks for the affected nodes which are either newly added or are involved in new events in the t-th snapshot. A static model is firstly learned for $\mathscr{G}_1$. The model for the t-th snapshot ($t > 1$) is initialized with the parameters from the fine-trained model for $t-1$-th snapshot and then optimized using the t-th set of random walks.

Another strategy employing random walk approaches in dynamic graphs is to generate each random walk by considering the timing information. The selection of the link for the next transition in a random walk can follow a distribution, e.g. a probability proportional to the time that the link was added. Nguyen et al. [97] apply this strategy. Instead of representing a dynamic graph as a sequence of discrete snapshots, they define a continuous-time dynamic graph $\mathscr{G} = (\mathbb{V}, \mathbb{E}_T)$ where each edge $e_{i,j} = (v_i, v_j, t) \in \mathbb{E}_T$ is assigned a unique time $t \in \mathbb{R}^+$. In such a graph, a temporal walk is valid when it is a set of vertices connected by links with ascending timestamps. Formally, a temporal walk from node v_1 to v_k in $\mathscr{G}$ is a sequence of nodes $\{v_1, v_2, \cdots, v_k\}$ such that $e_{i,i+1} \in \mathbb{E}_T$ and $t_i \leq t_{i+1}$. This definition echoes the random walk method for static graphs but includes an additional constraint where the walk is required to respect time.

Nguyen et al. [97] determine the initial edge $e_i \in \mathbb{E}_T$ from which a temporal walk starts by following a uniform or weighted distribution. The selection of distribution is typically based on the downstream applications. For example, in the link prediction task, beginning more walks from links nearer the current timestamp can improve the predictive performance. This is because the events (edges) that occurred in the long past are prone to hold less relevance to the current state of the graph. To generate a step starting from a node v for a temporal random walk, a set of neighboring nodes centered at this node v is denoted by $\Gamma_t(v) = \{(w,t')\}$ where $e = (v,w,t') \in \mathbb{E}_T$ and $t' \geq t$. $\Gamma_t(v)$ is used for selecting the next node to which the node v transitions. The selection also can follow a strategy. For instance, considering the time interval k between the occurrence time of two edges $e^t_{u,v}$ and $e^{t+k}_{v,w}$, the link prediction task on a social network wants k to be small, which constrains the temporal walks that friends from largely different time periods are not grouped together. The procedure to generate temporal walks in [97] is summarized in Algorithm 1.

However, random walk-based approaches for dynamic graph embedding are not an ideal choice for applications where the graphs change at a fast rate. This is because performing new walks and updating the embedding model with the new walks can be quite time-consuming. In addition, most proposed methods based on random walk only consider the structure information while neglecting node attributes, which could limit the overall expressive capabilities of these methods.

Algorithm 1 Temporal Random Walk. A method from Nguyen et al. [97]

Procedure Temporal Walk($\mathscr{G} = (\mathbb{V}, \mathbb{E}_T), e_{s,r}, t, L, C$) where L is the length of a walk, C is the window size for optimization.
Initialize temporal walk $S_t = [s,r]$ where $s \in \mathbb{V}$ and $r \in \mathbb{V}$
Set $i = r$
for $p = 1$ to $min(L,C) - 1$ **do**
    $\Gamma_t(i) = \{(w,t')\}$ where $e = (i,w,t') \in \mathbb{E}_T, t' \geq t$
    if $|\Gamma_t(i)| > 0$ **then**
        Select node $j \in \mathbb{V}$ from $\Gamma_t(i)$
        Append j to S_t
        Set $t = t'$, Set $i = j$
    else
        Terminate temporal walk
    end if
end for
Output temporal walk S_t rooted at node s

2.3.3 Representation Leaning on Dynamic Graphs

Snapshots-based Methods Several graph representation learning approaches have been developed for dynamic graphs. In a family of such methods, a dynamic graph is typically

represented as a sequence of discrete snapshots $\{\mathbb{G}_1, \mathbb{G}_2, \cdots, \mathbb{G}_T\}$, where each snapshot represents the state of the dynamic graph at time t or within an observed window $(t-1, t]$. Within this context, a cornerstone of dynamic graph representation learning methods revolves around the utilization of Recurrent Neural Networks (RNNs), leveraging the successes of RNNs in sequence modeling [86].

Many snapshots-based representation learning methods have been put forth centered on depicting a dynamic graph as a series of snapshots. The fundamental concept behind these methods can be described as $\mathbf{H}_t = \text{RNN}(\mathbf{H}_{t-1}, \mathbf{Z}_t)$, where the $d-$dimensional matrix $\mathbf{H}_t \in \mathbb{R}^{|\mathbb{V}_t| \times d}$ is the representations of $|\mathbb{V}_t|$ nodes for the dynamic graph at the t-th time window. $\mathbf{H}_t$ is obtained by firstly learning the representations $\mathbf{Z}_t$ of the t-th snapshot with a static GNN model and then feeding both the $\mathbf{Z}_t$ and the representations $\mathbf{H}_{t-1}$ of the dynamic graph at the $(t-1)$-th time window into a sequence modelling model, e.g., a form of RNN [86]. $\mathbf{H}_t$ jointly captures the t-th snapshot information as well as the historical evolution pattern in the past snapshots.

An early pioneer in snapshots-based dynamic graph representation learning is the work by Seo et al. [118]. Furthermore, Manessi et al. [91] combine Long Short-Term Memory networks (LSTMs) [53] with GCNs to learn both the influences among snapshots and graph structure in each snapshot. Sankar et al. [113] propose DySAT which leverages the attention mechanism in GCN models for each $\mathbb{G}_t$ to produce the final embeddings.

This thesis uses an approach proposed by Manessi et al. [91] as an example to illustrate such snapshots-based architecture. Given a dynamic graph $\mathbb{G} = (\mathbb{V}, \mathbb{E}_T)$ represented by a sequence of snapshots $\mathbb{G} = \{\mathbb{G}_1, \mathbb{G}_2, \cdots, \mathbb{G}_T\}$, where a snapshot $\mathbb{G}_t = (\mathbb{V}_t, \mathbb{E}_t)$ can be viewed as a static graph at timestamp t and all snapshots share the same vertices, namely $\mathbb{V}_t = \mathbb{V}$. As shown in Figure 2.4, a graph convolution layer of the model takes inputs of $\mathbf{A}_t$ and $\mathbf{Z}^{(l)}$ (in the first layer $\mathbf{Z}^{(l)} = \mathbf{X}_t$, $\mathbf{X}_t \in \mathbb{R}^{|\mathbb{V}| \times d}$.). It outputs representations $\mathbf{Z}_t^{(l+1)} \in \mathbb{R}^{|\mathbb{V}| \times f}$ for $|\mathbb{V}|$ nodes by:

$$\mathbf{Z}^{(l+1)} = \mathsf{ReLU}(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{Z}^{(l)} \mathbf{W}) \tag{2.14}$$

where the adjacency matrix $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_{|\mathbb{V}|}$ is modified from $\mathbf{A}$ with nodes' self-loop $\mathbf{I}_{|\mathbb{V}|}$, $\tilde{\mathbf{D}}_{i,i} = \sum_{j=1}^{|\mathbb{V}|} \tilde{\mathbf{A}}_{i,j}$ is used to symmetrically normalise $\tilde{\mathbf{A}}$. $\mathbf{W} \in \mathbb{R}^{d \times f}$ is a parameter matrix. $\mathsf{ReLU}()$ is the rectified linear unit activation function. In this architecture, there are T copies of such operation in a layer corresponding to T snapshots. All copies are shared the same training parameter matrix $\mathbf{W}$, which enables the $\mathbf{W}$ independent of the length of the sequence of snapshots.

As shown in Figure 2.4, such snapshots-based model proposed by Manessi et al. [91] uses a separate LSTM for each node. Considering a sequence of representations $\{\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_T\}$ where $\mathbf{z}_t \in \mathbb{R}^f$ for a node v, an LSTM layer takes input as this sequence

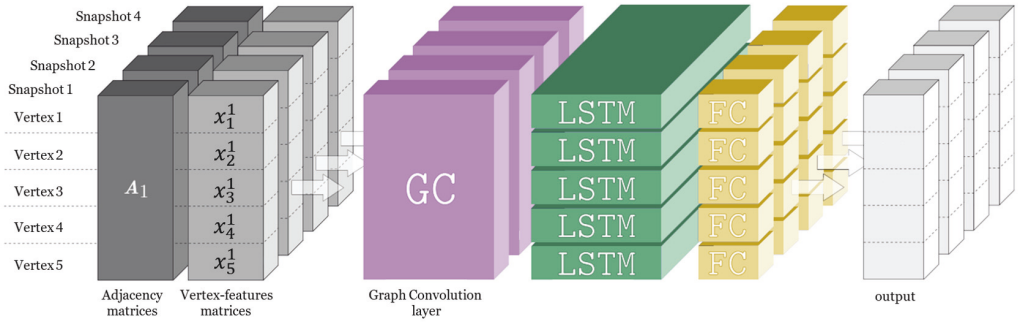


Figure 2.4 The architecture of a representation learning method for dynamic graphs from Manessi et al. [91].

and output a sequence of $\{\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_T\}$, $\mathbf{h}_t \in \mathbb{R}^k$ by:

$$
\begin{aligned}
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \\
\mathbf{c}_t &= \mathbf{j}_t \odot \tilde{\mathbf{c}}_t + \mathbf{f}_t \odot \mathbf{c}_{t-1} \\
\mathbf{o}_t &= \sigma(\mathbf{z}_t \mathbf{W}_o + \mathbf{h}_{t-1} \mathbf{U}_o + \mathbf{b}_o) \\
\mathbf{f}_t &= \sigma(\mathbf{z}_t \mathbf{W}_f + \mathbf{h}_{t-1} \mathbf{U}_f + \mathbf{b}_f) \\
\mathbf{j}_t &= \sigma(\mathbf{z}_t \mathbf{W}_j + \mathbf{h}_{t-1} \mathbf{U}_j + \mathbf{b}_j) \\
\tilde{\mathbf{c}}_t &= \sigma(\mathbf{z}_t \mathbf{W}_c + \mathbf{h}_{t-1} \mathbf{U}_c + \mathbf{b}_c)
\end{aligned}
\tag{2.15}
$$

where $\odot$ is the Hadamard product, $\sigma()$ is the sigma activation function. The $\mathbf{W}_l \in \mathbb{R}^{f \times k}$ and $\mathbf{U}_l \in \mathbb{R}^{k \times k}$ are weight matrices, $\mathbf{b}_l$ are bias vectors where $l \in \{o, f, j, c\}$. There are $|\mathbb{V}|$ (number of all nodes) copies of such LSTM. The training weights are shared among these LSTM which make this model independent of the number of graph nodes. The LSTM layer in this architecture outputs a matrix of $|\mathbb{V}| \times k \times T$ dimensions, which means that each node at timestamp t has a k dimensional representation.

Continuous Methods Instead of representing a dynamic graph as static snapshots, several innovative approaches [134, 71] learn the representations by unfolding a dynamic graph into a sequence of events that occurred in a continuous time space. This strategy allows for a comprehensive consideration of all temporal changes inherent in dynamic graphs. The fundamental concept underlying such methods involves updating node representations based on the unfolding sequence of events. Specifically, when a new link forms between nodes v and u at time t, the representation for node v, denoted as $\mathbf{z}_v$, undergoes an update based on the RNN mechanism. This update is defined as $\mathbf{z}_v = \text{RNN}(\mathbf{z}_{v,previous}, \text{CONCAT}(\mathbf{z}_u; \Delta t_v)$, where $\mathbf{z}_{v,previous}$ is the previous representation of node v before t, Δt_v is the time interval from the time of the last interaction of node v to t.

An exemplary instance of this dynamic graph unfolding approach is presented in the work by Kumar et al. [71]. The authors introduce a JODIE model to learn the dynamic

representations of users and items from a sequence of user-item interactions. In JODIE, nodes within a dynamic graph are categorized into two distinct types: users and items. Edges between these nodes correspond to user-item interactions, such as a customer making a purchase in an e-commerce platform, an editor modifying an article in Wikipedia, or an individual posting on another individual's social network page.

JODIE unfolds a dynamic graph into a sequence of temporal user-item interactions $\mathbb{S}$. An interaction is represented by $\mathbb{S}_j = (u_j, i_j, f_j, t_j)$, which is an event occurred between a user $u_j \in \mathbb{U}$ and an item $i_j \in \mathbb{I}$ at time t where f_j is a feature vector associated with this interaction, $\mathbb{U}$ and $\mathbb{I}$ are the sets for users and items, respectively. The JODIE model uses two separate RNNs to learn the dynamic representations of users and items. RNN_U is used to update user embeddings and it shares the weights among all users. RNN_I shared across all items is used to update item embeddings. When a user u interacts with item i at time t, JODIE updates the user embedding $\mathbf{u}(\mathbf{t}) \in \mathbb{R}^n$ and item embedding $\mathbf{i}(\mathbf{t}) \in \mathbb{R}^m$ using the following expressions:

$$\begin{aligned} \mathbf{u}(t) &= \text{RNN}_U(\mathbf{u}_{previous}, \mathbf{i}_{previous}, \Delta t_u, f) \\ \mathbf{i}(t) &= \text{RNN}_I(\mathbf{i}_{previous}, \mathbf{u}_{previous}, \Delta t_i, f) \end{aligned} \tag{2.16}$$

where $\mathbf{u}_{previous}$ and $\mathbf{i}_{previous}$ represent the user and item previous embeddings before the time t. Δt_u and Δt_i are the time elapsed since u's last interaction and i's last interaction, respectively. f is the interaction feature vector.

Discussion The methods based on discrete snapshots exhibit the potential to harness advancements derived from a diverse array of GNNs for static graphs. These methods, however, inherently rely on fixed time intervals to generate snapshots, potentially overlooking nuanced temporal dynamics. In contrast, continuous models possess the capacity to capture more temporal information when compared with snapshots-based methods. However, it's important to note that the complexity of continuous models increases as the increasing number of changes in a dynamic graph, which would lead to a big computational load as the graph evolves over time. Besides, existing continuous models are proposed for some specific types of dynamic graphs, thus constraining their applicability across a broader spectrum.

In light of these considerations, there emerge two noteworthy avenues for future exploration. Firstly, an exploration of methods based on discrete snapshots is warranted, capitalizing on the recent advancement achieved in the domain of neural networks. This approach promises to harmonize the benefits of static GNNs for dynamic graph analysis. Secondly, the evolution of continuous models merits attention, with the goal of devising strategies to mitigate the computational overload associated with larger and more

complex dynamic graphs. Simultaneously, there is ample room to develop continuous models that can be applied to a broader spectrum of dynamic graph types. These dual trajectories—embracing the potential of discrete snapshots-based methods in synergy with neural network advancements, and refining and diversifying the realm of continuous models—stand out as promising pathways for future research in this field.

2.3.4 Point Process Embedding Methods

In the realm of traditional statistics, the temporal point process [103] serves as an important mathematical framework for modeling a sequence of discrete events occurring asynchronously over time. Central to the concept of the temporal point process is the conditional intensity function $\lambda(t)$, which models the time of the next event occurrence given all historical events. Given the historical events before t, $\lambda(t)dt$ is the probability of an event occurring within a small window $[t, t+dt]$. The conditional density function $f(t)$ can be specified as $f(t) = \lambda(t)S(t)$, where $S(t) = \exp\left(-\int_{t_n}^{t} \lambda(\tau)d\tau\right)$ is also known as the survival function, which is the probability that no new event happens up to t since t_n.

Various formulations of $\lambda(t)$, such as Poisson process [103], Hawkes process [48], and Self-Correcting process [57], have been proposed in order to capture the underlying patterns in the temporal point sequences. Herein we turn our attention to the Hawkes process. Unlike simple models like the Poisson process which assumes that event occurrences are entirely independent, the Hawkes process models the events by considering not only the internal properties of the event self but also the impacts of the historical events occurring at the previous times. The $\lambda(t)$ for the Hawkes process [48] can be expressed as follows:

$$\lambda_i(t) = \mu_i + \sum_{t_j<t} \alpha_j \kappa(t_j - t), \tag{2.17}$$

where t_j is the time of the j-th historical event occurred before t, $\mu_i \geq 0$ is the base intensity that is a function of t but independent of the historical events. $\kappa(\cdot)$ is a kernel function that describes how much historical events excite the occurrence of an event at t.

A dynamic graph can be unfolded as a sequence of events in a continuous time space. This sequence naturally aligns itself with the concept of temporal point processes. Pioneering this alignment, Yuan et al. [178] propose a model named HTNE, which stands as one of the earliest endeavors to employ temporal point processes in the analysis of dynamic graphs.

We use HTNE to showcase the practical implementation of point processes in the dynamic graph realm. In HTNE, the event in the point process corresponds to the formation of a link between a pair of nodes in a dynamic graph. For each node, a neighborhood formation sequence is generated first. Mathematically, given a node v_x, its corresponding neighborhood formation sequence can be denoted as a sequence of events,

i.e., $\{(e_{x,nei_1},t_1),(e_{x,nei_2},t_2),\cdots,(e_{x,nei_n},t_n)\}$, where an event is an edge formation $e_{x,nei}$ with its neighboring node v_{nei} at time t. HTNE models the neighborhood formation sequences by modifying the Hawkes process into the following formulation:

$$\lambda_{e_{x,y}}(t) = \mu_{e_{x,y}} + \sum_{t_{nei}<t} \alpha_{nei,y}\kappa(t-t_{nei}) \tag{2.18}$$

where $\lambda_{e_{x,y}}(t)$ describes the intensity of an event to construct a link $e_{x,y}$ between node v_x and any other node v_y at time t, given the historical neighborhood formation sequence for vertex v_x before time t. $\lambda_{e_{x,y}}(t)dt$ is the probability of forming such edge $e_{x,y}$ within a small window $[t,t+dt]$. $\mu_{e_{x,y}}$ is a baseline intensity of forming an edge $e_{x,y}$ at any time, which is independent of the historical events. $\alpha_{nei,y}$ represents the degree to how much a historical neighboring node v_{nei} excites the formation of this edge $e_{x,y}$, where node v_{nei} was linked to node v_x before the time t. The kernel function $\kappa(t-t_{nei})$ describes the time decay pattern. HTNE uses an exponential function as the kernel function, i.e., $\kappa(t-t_{nei}) = \exp(\delta_x(t-t_{nei}))$, where δ_x is a learnable parameter dependent of node v_x. δ_x describes that for the node v_x, its different historical neighboring node can influence its current neighbor node v_y interaction with different intensity.

The $\mu_{e_{x,y}}, \alpha_{nei,y}$ are the parameters that can be learned by optimizing the Hawkes process. HTNE represents $\mu_{e_{x,y}}$ by $\mu_{e_{x,y}} = -dist(\mathbf{e}_x,\mathbf{e}_y)$, where $\mathbf{e}_i$ is the the embedding for each node v_i. Intuitively, this reveals the natural affinity of nodes in a graph, i.e., the node embedding $\mathbf{e}_x$ for node v_x wants neighboring nodes to have similar embeddings and independent nodes to have distant embedding vectors. For example, when node v_x is connected with node v_y, the $dist(\mathbf{e}_x,\mathbf{e}_y)$ should be learned as small and thus this negative distance $\mu_{e_{x,y}}$ is large so that Hawkes model gives high probability of the event forming a link between the two nodes.

Similarly, $\alpha_{nei,y}$ is written as $\alpha_{nei,y} = w_{nei,x}(-dist(\mathbf{e}_{nei},\mathbf{e}_y))$ where the weight $w_{nei,x}$ is

$$w_{nei,x} = \frac{\exp(-dist(\mathbf{e}_x,\mathbf{e}_{nei}))}{\sum_{\mathbf{e}_i\in\mathbb{H}_x(t)}\exp(-dist(\mathbf{e}_x,\mathbf{e}_i))}, \tag{2.19}$$

where $\mathbb{H}_x(t)$ is a set of embeddings of nodes which are the neighbours of node v_x before time t. Intuitively, the weight $w_{nei,x}$ describes a fact: the affinity between the historical vertex v_{nei} and the vertex v_y should be influenced by the vertex v_x. In other words, it describes the different influences of the different historical neighboring node v_{nei} on the formation of the edge $e_{x,y}$. For example, in a co-authorship network, some authors have fixed co-researchers over time. As a result, the historical co-authors have larger impacts on predicting the co-authors in the next publication. Whereas some researchers transfer their research interests from one area to another different one. Their co-authors may change frequently over time, such that they have different affinities with the historical co-authors.

Hawkes process is typically optimized by maximum likelihood estimation which is based on the assumption that the observed data is most probable. In HTNE, the Hawkes process should give a higher probability for the case when v_y is a neighboring node of node v_x but a lower probability when v_y is not the neighbor of node v_x.

Letting $\mathbb{H}_x(t)$ denote the historical part of the neighborhood formation sequence for vertex v_x until time t, the probability of link formation between node v_x and another node v_y at time t can be inferred by the conditional intensity:

$$p(e_{x,y}, \mathbb{H}_x(t)) = \frac{\lambda_{e_{x,y}}(t)}{\sum_{y'} \lambda_{e_{x,y'}}(t)} \tag{2.20}$$

The log-likelihood of neighborhood formation sequences for all nodes in the graph can be obtained with this probability. However, the Equation 2.20 requires to calculate $\lambda_{e_{x,y'}}(t)$ over the entire set of nodes for each event, i.e., edge formation, which is computationally expensive when the graph is large.

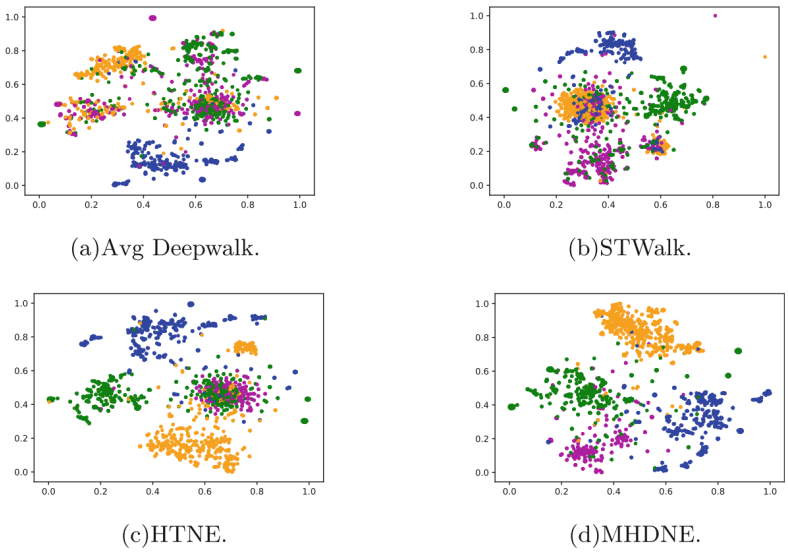
To address this issue, HTNE uses the Negative Sampling method [95] to approximately optimize the Hawkes process for each edge formation. Thus the objective function can be written as follows:

$$loss = \log\sigma(\lambda_{e_{x,y}}(t)) + \sum_{k=1}^{K} -\log\sigma(\lambda_{e_{x,k}}(t)) \tag{2.21}$$

where y refers to the node v_y that is in the neighborhood formation sequence of node v_x, while k refers to negative node v_k which have not present in this neighborhood formation sequence. The negative nodes for node v_x are sampled by extracting nodes from all nodes of the graph but not in the neighborhood formation sequence of node v_x. The $\sigma(x) = 1/(1+\exp(-x))$ is the sigmoid activation function. It can be seen that the negative sampling loss computes the conditional intensity for only a small percentage of nodes rather than all nodes in the graph.

In HTNE, optimizing the Hawkes process, where events are the link formations in neighborhood formation sequences, is to modify the parameter $\mu_{e_{x,y}}, \alpha_{nei,y}$ in the Equation 2.18. The $\mu_{e_{x,y}}, \alpha_{nei,y}$ are represented by the embeddings of corresponding nodes. As a result, minimizing the loss function in Equation 2.21 means to optimize the embeddings for nodes. The negative sampling loss function encourages nodes connected with each other to have similar embedding, while those far apart have separated ones.

Ying et al. [155] propose a model MHDNE which is a subsequent work of HTNE. MHDNE considers not only the influence of historical neighbor nodes on the current node but also the impact of the common neighbors between two nodes. MHDNE obtains better performance than HTNE. Figure 2.5 that is extracted from the work of MHDNE [155], shows a t-SNE visualization of the embeddings of 2500 authors from four research fields



(a)Avg Deepwalk. (b)STWalk.

(c)HTNE. (d)MHDNE.

Figure 2.5 Visualization of four models. Image extracted from MHDNE [155].

(colors). Avg Deepwalk conducts Deepwalk [104] algorithm over a set of snapshots to obtain vertex representations at different times. STWalk [100] performs space-walk and time-walk on the graph which can extract the spatio-temporal behavior of vertexes. As shown in Fig. 2.5, Avg Deepwalk algorithm can only cluster one class of authors to an independent area, and the researchers in the rest three classes are mixed. STWalk maps the purple dots and green dots to scattered positions, failing to extract the properties of such kinds of authors. HTNE [178] can map the authors in the three fields to different clusters, but map some researchers in these three classes into the "purple" class. MHDNE shows better results than the other three methods, which can map the authors to different communities with clear margins.

The embedding methods based on the point process have exhibited a distinct advantage for event time prediction [178, 155], attributed to the continuous nature of the point process. However, recent studies [56] have revealed shortcomings in the performance of these methods, prompting the need for refinement. Furthermore, a critical limitation of existing point process-based embedding models lies in their oversight of node attributes. To this end, a promising avenue for future exploration lies in the realm of dynamic graph embedding, entailing the seamless integration of node attributes alongside point processes. This innovative fusion holds considerable potential for advancing predictive accuracy and enhancing the practical applicability of such models.

2.4 Summary

The aforementioned literature demonstrates the incredible progress in the field of GNNs and several notable methods in graph representation learning, highlighting key advancements

in attention mechanisms, sampling techniques, specialized architectures for diverse graph structures, etc. Expanding on these advancements, our research encompasses three distinct contributions presented in the subsequent chapters. Firstly, we propose a general and scalable GNN framework that demonstrates remarkable performance. Secondly, we present an interpretative analysis of GNNs in the context of link prediction. Lastly, we study the effectiveness of different GNN techniques in addressing challenges encountered in real-world graph applications. These innovations collectively enhance generalization capabilities, improve scalability for handling large graphs, and increase the interpretability of model predictions, thereby establishing our research works as significant contributions to the realm of graph research.

Chapter 3

VAGNN: A General and Scalable GNN Framework

This chapter introduces a general GNN framework called Virtual Adjacency Graph Neural Network (VAGNN), motivated by the commonality analysis of various GNN architectures. Unlike conventional GNNs that employ fixed configurations to construct neighborhoods for information aggregation, VAGNN leverages a virtual adjacency matrix to optimize neighborhoods by selectively excluding 1-hop neighbors while incorporating high-hop local neighbors and global nodes. Additionally, VAGNN allows for the selection of different attention mechanisms for aggregation and the incorporation of supplementary information into attention weights. The linear computational complexity of VAGNN makes it scalable for handling large graphs. Experimental evaluations on diverse real-world datasets validate the generalization and scalability capabilities of VAGNN. Parameter sensitivity analysis also reveals the importance of carefully selecting and balancing the inclusion of local and global information in VAGNN. This work lays the foundation for further exploration in developing more efficient techniques for virtual adjacency matrix construction and weighted aggregation functions, opening up possibilities for the design of more robust GNNs in the future.

3.1 Introduction

Over the past few years, a considerable array of GNN models has emerged [148, 52]. The majority of these models adhere to the neighborhood information aggregation algorithm [150] (also known as the message-passing mechanism [40]). This algorithm can be expressed by Eq. 3.1[1], where the representation of a node v_i is iteratively updated by

[1]For simplicity, we omit the residual connections, activation functions, etc.

aggregating representations of its neighboring nodes and the node itself (i.e., $\mathbb{N}(v_i) \cup \{v_i\}$).

$$\mathbf{h}_i^{(l+1)} = \text{AGGREGATE}^{(l)}\left(\left\{\mathbf{h}_j^{(l)} \mid v_j \in \mathbb{N}(v_i) \cup \{v_i\}\right\}\right) \tag{3.1}$$

Various aggregation-based GNNs have been proposed [52], with notable innovations primarily centered around two key aspects: the construction of the neighborhood set $\mathbb{N}(\cdot)$ and the development of the function AGGREGATE$(\cdot)$. As shown in Eq. 3.1, earlier GNNs [64, 137] typically construct the set $\mathbb{N}(v_i)$ with the direct utilization of all 1-hop neighbors of the node v_i. Alternatively, some models [46, 21] construct $\mathbb{N}(v_i)$ by selecting a subset of 1-hop neighbors of v_i to improve computational efficiency. Furthermore, certain models [1, 77, 35] incorporate high-order neighboring nodes of v_i into $\mathbb{N}(v_i)$, resulting in high-hop GNNs. Drawing inspiration from the Transformer model in Natural Language Processing (NLP) [135], $\mathbb{N}(v_i)$ is created to encompass all nodes in a graph in several graph Transformers [159, 156, 19].

In terms of the design of the function AGGREGATE$(\cdot)$, the simplest approaches [64, 46, 151] involve set-based MEAN- or MAX-pooling over the set of node representations (i.e., the set of $\left\{\mathbf{h}_j^{(l)} \mid j \in \mathbb{N}(i) \cup \{i\}\right\}$ in Eq. 3.1). Subsequently, attention mechanisms were integrated into the AGGREGATE$(\cdot)$ [137, 132]. In the graph Transformer, supplementary information, such as shortest-path distance between nodes and edge features, is encoded into additional weights alongside attention weight to collectively govern the neighborhood aggregation [156, 19, 102].

The analysis of diverse GNN architectures motivates us to propose a more general GNN framework called Virtual Adjacency GNN (VAGNN). We introduce the term "virtual adjacency matrix" for GNNs to differentiate it from the actual adjacency matrix. In graph theory, the adjacency matrix describes the connections between nodes. In the case of a graph with N nodes, the adjacency matrix is an $N \times N$ square matrix, where the i,j-th entry of the matrix is assigned a value of 1 if node v_j is a 1-hop neighbor of node v_i, and 0 otherwise. In GNNs like GCN [64] where $\mathbb{N}(\cdot)$ only comprises all 1-hop neighbors of a node, the matrix representing all nodes' $\mathbb{N}(\cdot)$s aligns precisely with the adjacency matrix. However, for many GNNs [46, 21] in which the construction of $\mathbb{N}(\cdot)$ involves using partial 1-hop neighbors or higher-hop nodes of a node, the corresponding matrix is no longer being the actual adjacency matrix. Therefore, it is appropriate to refer to the matrix governing the neighborhood aggregation in GNNs as a virtual adjacency matrix.

Our proposed VAGNN is a general and scalable GNN framework. Firstly, with the concept of the virtual adjacency matrix, we can construct the neighborhood set $\mathbb{N}(\cdot)$ from a general perspective. Unlike most existing GNNs that employ fixed methods for constructing the $\mathbb{N}(\cdot)$, VAGNN allows for the optimization of the $\mathbb{N}(\cdot)$ by customizing the removal of 1-hop neighbors and the incorporation of high-hop local neighbors and global nodes. Moreover, VAGNN offers the capability to choose from various attention mechanisms for

the aggregation function, and provides the option to incorporate supplementary information for controlling the aggregation. Theoretically, VAGNN can be transformed into a majority of existing GNNs and graph Transformers[137, 77, 35, 156]. Also, it holds the potential to achieve superior performance by optimizing the virtual adjacency matrix and aggregation methods tailored specifically for the given graph dataset.

The virtual adjacency matrix in VAGNN acts as a mask that determines whether the nodes require attention calculation and information aggregation or not. When implementing VAGNN based on sparse matrix operations, its computational complexity exhibits a linear relationship with the number of node pairs used for neighborhood aggregation (approximately equal to the sum of the sizes of all nodes' $\mathbb{N}(\cdot)$s). This linear complexity demonstrates the scalability of VAGNN, rendering it well-suited for large graphs [54]. Especially, by selectively choosing global nodes, VAGNN effectively overcomes the scalability challenges associated with utilizing global nodes in graph Transformers, reducing the storage complexity from $O(N^2)$ or even $O(N^3)$ to N, with N representing the total number of nodes in a graph [159, 156, 102].

We conduct extensive experiments on nine diverse real-world datasets sourced from the Open Graph Benchmark (OGB) [54]. The experimental results demonstrate the remarkable generalization and scalability capabilities of the proposed VAGNN. We also investigate the sensitivity of parameters involved in the construction of the virtual adjacency matrix. Our observations reveal that removing a small number of 1-hop neighboring nodes and incorporating a limited number of global nodes yield favorable outcomes, while an excessive utilization of global nodes leads to performance degradation. These findings highlight the importance of carefully selecting and balancing the inclusion of local and global information in GNNs. Furthermore, we explore the performance of GNN models equipped with different attention mechanisms [137, 132, 156]. Our results show that the self-attention mechanism consistently outperforms others. Lastly, superior results are achieved by VAGNN when employing the combination of optimal parameters.

Our comprehensive analysis and empirical investigation demonstrate the potential of VAGNN as an effective approach for real-world graph-related tasks [54]. The obtained outcomes further emphasize the importance of carefully configuring the virtual adjacency matrix to maximize performance. Future research can focus on further optimizing the virtual adjacency matrix construction methods and weighted aggregation functions for specific graph datasets to unlock even greater potential in graph representation learning.

3.2 Preliminary

Most modern GNNs follow the neighborhood information aggregation algorithm, as expressed by Eq. 3.1. This equation describes the update rule of the representation of an

individual node in GNNs. In fact, the aggregation-based GNNs can be formalized based on matrix multiplication as below:

$$\mathbf{H}^{(l+1)} = \sigma\left(\overset{*}{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right), \tag{3.2}$$

where $\mathbf{H}^{(l)} \in \mathbb{R}^{N\times d}$ and $\mathbf{H}^{(l+1)} \in \mathbb{R}^{N\times d'}$ are the input and output node representations in the l-th layer of a GNN. The i-th row of $\mathbf{H}$ is denoted as $\mathbf{h}_i \in \mathbb{R}^d$ which is a d-dimensional representation of the node v_i. $\mathbf{H}^{(0)} = \mathbf{X}$ is the input feature matrix. $\mathbf{W}^{(l)} \in \mathbb{R}^{d\times d'}$ is a trainable weight matrix. $\sigma(\cdot)$ is an activation function. $\overset{*}{\mathbf{A}}$ is a matrix governing the information propagation between nodes. For the plain GNNs like GCN [64], $\overset{*}{\mathbf{A}}$ is a normalized version of the adjacency matrix $\mathbf{A}$. For high-hop GNNs[77, 35], $\overset{*}{\mathbf{A}}$ would contain the connection information between every node and its high-hop neighbors.

3.3 The Proposed VAGNN

In this section, we introduce our VAGNN. Without loss of generality, we demonstrate this work on homogeneous graphs [52]. To enhance the readability of the subsequent equations, we provide a concise summary of the notations in Table 3.1.

Table 3.1 A brief summary of the notations used in VAGNN

Notation	Description
l, L	the l-th layer and total layers L
m, M	the m-th head and total heads M in multiple-head attention module $\text{MultiHead}(\cdot)$
$d, d', d_{\text{O}}, d_{\text{A}}$	the dimension of the representation of a node
$\mathbf{A}$	the actual adjacency matrix
$\overset{*}{\mathbf{A}}$	the matrix for information propagation
$\overset{\vee}{\mathbf{A}}$	the virtual adjacency matrix
$\overset{\wedge}{\mathbf{A}}$	normalized version of the virtual adjacency matrix
$\hat{a}_{i,j}^{(l,m)}$	the normalized weight between nodes v_i, v_j
$\bar{a}_{i,j}^{(l,m)}$	the weight for information aggregation between v_i, v_j
$\check{a}_{i,j}$	the i,j-th entry of the virtual adjacency matrix $\overset{\vee}{\mathbf{A}}$

We begin by providing an overview of the architecture of a VAGNN layer. Formally, the l-th layer of VAGNN can be described by the equation below:

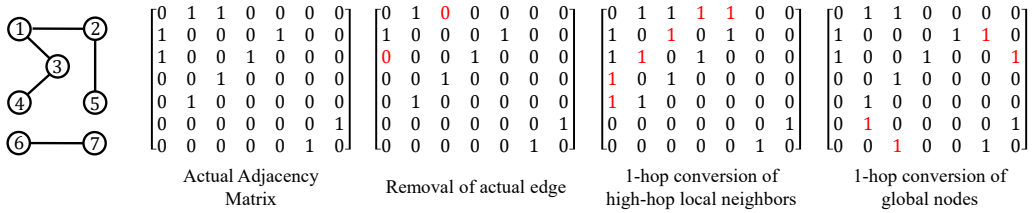


Figure 3.1 An illustration of the methods constructing a virtual adjacency matrix on an undirected graph.

$$\mathbf{H}^{(l+1)} = \text{MultiHead}\left(\mathbf{H}^{(l)}\right)\mathbf{W}_{\text{O}}^{(l)}, \tag{3.3}$$

where

$$\begin{aligned}\text{MultiHead}\left(\mathbf{H}^{(l)}\right) = \text{COMBINE}\Big(\Big\{&\mathbf{H}^{(l)}\mathbf{W}^{(l,0)},\\ &\hat{\mathbf{A}}^{(l,m)}\mathbf{H}^{(l)}\mathbf{W}^{(l,m)} : \forall m = 1,\cdots,M\Big\}\Big).\end{aligned} \tag{3.4}$$

MultiHead $\left(\mathbf{H}^{(l)}\right)$ is a multi-head attention module where M is a hyper-parameter defining the number of attention heads. Multiple head attention in GNNs[137, 156] is an extension of the attention mechanism that is commonly used in sequence-based models such as language Transformers [135]. The MultiHead $\left(\mathbf{H}^{(l)}\right)$ allows an individual layer of VAGNN to learn different patterns and relationships within the representations $\mathbf{H}^{(l)}$ through different attention heads simultaneously, improving the model's expressive power. The module involves three main steps: computing attention coefficients (i.e., $\hat{\mathbf{A}}^{(l,m)} \in \mathbb{R}^{N\times N}$ for the m-th head), aggregating information (i.e., the matrix multiplication $\hat{\mathbf{A}}^{(l,m)}\mathbf{H}^{(l)}\mathbf{W}^{(l,m)}$), and combining representations performed by the function COMBINE$(\cdot)$. The function COMBINE$(\cdot)$ can be Concatenation, MEAN, etc. The $\mathbf{W}^{(l,m)} \in \mathbb{R}^{d\times d_{\text{O}}}$ is a trainable weight matrix. The weight matrix $\mathbf{W}_{\text{O}}^{(l)}$ satisfies $\mathbf{W}_{\text{O}}^{(l)} \in \mathbb{R}^{d_{\text{O}}(M+1)\times d'}$ or $\mathbf{W}_{\text{O}}^{(l)} \in \mathbb{R}^{d_{\text{O}}\times d'}$ if COMBINE$(\cdot)$ is Concatenation or MEAN, respectively. The hyper-parameters d, d', and d_A define the dimensions of the representations of each node in different learning stages.

In Eq. 3.4, the $\mathbf{H}^{(l)}\mathbf{W}_{\text{A}}^{(l,0)}$ is a residual connection from $\mathbf{H}^{(l)}$ towards $\mathbf{H}^{(l+1)}$. The $\hat{\mathbf{A}}^{(l,m)}\mathbf{H}^{(l)}\mathbf{W}^{(l,m)}$ is the neighborhood information aggregation in the m-th attention head of the l-th layer of VAGNN, where $\hat{\mathbf{A}}^{(l,m)} \in \mathbb{R}^{N\times N}$ stores the normalized weights. The i,j-th entry of $\hat{\mathbf{A}}^{(l,m)}$ (i.e., $\hat{a}_{i,j}^{(l,m)}$) is obtained by normalizing $\bar{a}_{i,j}^{(l,m)}$ using the entries in the i-th row of $\bar{\mathbf{A}}^{(l,m)}$ with the softmax function:

$$\hat{a}_{i,j}^{(l,m)} = \text{softmax}_j\left(\bar{a}_{i,j}^{(l,m)}\right) = \frac{\exp\left(\bar{a}_{i,j}^{(l,m)}\right)}{\sum_{k=0}^{N-1}\exp\left(\bar{a}_{i,k}^{(l,m)}\right)}. \tag{3.5}$$

The $\bar{a}_{i,j}^{(l,m)}$, as the i,j-th entry of $\bar{\mathbf{A}}^{(l,m)} \in \mathbb{R}^{N\times N}$, is the weight for information aggregation between node v_i and v_j. It is computed by the equations below:

$$\bar{a}_{i,j}^{(l,m)} = \begin{cases} \text{ATTN}\left(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}\right) + \text{EXT}\left(v_i, v_j\right), & \text{if } \check{a}_{i,j} = 1 \\ 0, & \text{if } \check{a}_{i,j} = 0 \end{cases}, \tag{3.6}$$

where the function $\text{ATTN}\left(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}\right) \to \mathbb{R}$ calculates the attention coefficient between node v_i and v_j using their corresponding representation $\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}$. The function $\text{EXT}\left(v_i, v_j\right) \to \mathbb{R}$ introduces an extra weight between node v_i and v_j, enabling the incorporation of information that is challenging to be acquired solely by the attention coefficient. For instance, this function can incorporate structural information such as the shortest-path distance between node v_i and v_j [156]. The $\check{a}_{i,j}$ is the i,j-th entity of the matrix $\check{\mathbf{A}}$. Herein $\check{\mathbf{A}}$ is our proposed *virtual adjacency matrix* for VAGNN. In Eq. 3.6, we can see that $\check{\mathbf{A}}$ acts as a mask, determining which node pairs' weights need to be computed.

Eq. 3.6 presents the fundamental elements underlying the proposed VAGNN. This equation summarizes and generalizes the common characteristics of GNNs and graph Transformers [137, 77, 156], endowing VAGNN with powerful expressiveness. In the subsequent sections, we will provide a comprehensive exposition of the key components of this equation.

3.3.1 Attention Methodologies

The attention mechanism enables the GNN model to learn adaptive importance weights between two nodes, allowing every node to differentiate between neighbors during the process of aggregating neighborhood information in the GNN.

As shown in Eq. 3.6, the $\text{ATTN}\left(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}\right)$ is the attention function used to calculate the attention coefficient between nodes v_i, v_j based on the representations of those two nodes. We summarize several main methods for the function $\text{ATTN}(\cdot,\cdot)$ as follows:

1. Cosine similarity-based attention [132]. This is the simplest method for computing attention weights. It is typically computed as:

$$\text{ATTN}\left(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}\right) = \beta^{(l,m)} \cos\left(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}\right), \tag{3.7}$$

   where $\beta^{(l,m)} \in \mathbb{R}$ is a learnable parameter for the m-th attention head in the l-th layer, $\cos(x,y) = xy^\top/|x||y|$, and $|x|$ is the Euclidean norm of the vector x. $\cos(\cdot,\cdot)$ calculates the cosine of the angle between two vectors, tending to learn higher attention weights for more relevant representation vectors.

2. GAT-type attention [136, 15]. In GAT [136], $\mathrm{ATTN}(\cdot,\cdot)$ can be formalized as:

$$\mathrm{ATTN}\left(\mathbf{h}_i^{(l)},\mathbf{h}_j^{(l)}\right)=\sigma\Bigg(\mathbf{q}^{(l,m)}\cdot\Big(\mathrm{CAT}\left(\mathbf{h}_i^{(l)}\mathbf{W}_{\mathrm{Q}}^{(l,m)},\mathbf{h}_j^{(l)}\mathbf{W}_{\mathrm{Q}}^{(l,m)}\right)\Big)^{\top}\Bigg), \tag{3.8}$$

where $\sigma(\cdot)$ is an activation function, the $\cdot^{\top}$ represents transposition, $\mathrm{CAT}(\cdots)$ is the concatenation operation, $\mathbf{W}_{\mathrm{Q}}^{(l,m)}\in\mathbb{R}^{d\times d_{\mathrm{A}}}$ is a trainable weight matrix shared across all nodes, and $\mathbf{q}^{(l,m)}\in\mathbb{R}^{2d_{\mathrm{A}}}$ is a weight vector. Note that $\mathbf{W}_{\mathrm{Q}}^{(l,m)}$ transforms the representations of nodes from the dimension d into d_{A}, which offers the advantage of optimizing computer memory usage through assigning a small value to the hyper-parameter d_{A} when computing attention on large graphs.

   Compared to cosine similarity-based attention, GAT-type attention employs a shared linear transformation to hidden states across all nodes in the graph, which could improve the expressive power of the model.

3. Self-attention mechanism [167, 159, 156]. Self-attention has demonstrated great success in Transformers in NLP and computer vision[135, 30]. Recent efforts have been made to extend it to the domain of GNNs. For self-attention mechanism, $\mathrm{ATTN}(\cdot,\cdot)$ can be expressed as:

$$\mathrm{ATTN}\left(\mathbf{h}_i^{(l)},\mathbf{h}_j^{(l)}\right)=\frac{\left(\mathbf{h}_i^{(l)}\mathbf{W}_{\mathrm{Q}}^{(l,m)}\right)\cdot\left(\mathbf{h}_j^{(l)}\mathbf{W}_{\mathrm{K}}^{(l,m)}\right)^{\top}}{\sqrt{d_{\mathrm{A}}}}, \tag{3.9}$$

   where $\mathbf{W}_{\mathrm{Q}}^{(l,m)}\in\mathbb{R}^{d\times d_{\mathrm{A}}}$ and $\mathbf{W}_{\mathrm{K}}^{(l,m)}\in\mathbb{R}^{d\times d_{\mathrm{A}}}$ are trainable weight matrices shared across all nodes for the m-th attention head in the l-th layer, $\frac{1}{\sqrt{d_{\mathrm{A}}}}$ is a scaling factor [135, 156].

   As shown in Eq. 3.8 and Eq. 3.9, both the GAT-type attention and self-attention mechanism apply linear transformations to the hidden representations of two corresponding nodes. Nevertheless, these two attention methods exhibit a distinction. In the GAT-type attention, the transformed results are concatenated and subsequently subjected to a dot product operation with a weight vector. In contrast, the self-attention mechanism directly performs a dot-product operation on the two transformed representation vectors. In terms of computation resource utilization, the self-attention mechanism would be more space-efficient than the GAT-type attention method.

3.3.2 The Extra Weight

In Equation 3.6, we can see that the learning of the attention weight between two nodes v_i, v_j primarily relies on the node-wise representations, i.e., h_i and h_j. However, when characterizing the relationship between two nodes in graph data, besides the reliance on node representations, there exist other factors, such as the shortest-path distance between two nodes, whether they have an edge, the weight of the edge, and the number of common neighbors they share. To effectively capture these extra pieces of information, we introduce a function $\mathrm{EXT}\left(v_i, v_j\right)$ and add it to the attention weight, thus providing a better description of the relationship between the two nodes. The incorporation of such additional weight can be seen as analogous to the utilization of positional encoding in NLP Transformer models [135], where positional information between two tokens is combined with the self-attention weight.

$\mathrm{EXT}\left(v_i, v_j\right)$ serves as a valuable complement to the attention weight, introducing an explicit component in the GNN design. We provide a specific form of $\mathrm{EXT}\left(v_i, v_j\right)$ as follows:

$$\mathrm{EXT}\left(v_i, v_j\right) = \alpha^{(l,m)}\mathrm{SPD}\left(v_i, v_j\right) + \cdots + \mathrm{EW}\left(v_i, v_j\right), \tag{3.10}$$

where $\mathrm{SPD}\left(v_i, v_j\right)$ calculate the shortest-path distance between two nodes v_i, v_j, $\alpha^{(l,m)} \in \mathbb{R}$ is a trainable weight for the function $\mathrm{SPD}\left(\cdot,\cdot\right)$, $\mathrm{EW}\left(v_i, v_j\right)$ can be a function to aggregate the features of edges between v_i, v_j. In fact, previous works [159, 156, 19, 102] have explored incorporating information independent of node features to the attention weight. For example, Graphormer [156] proposes a method of encoding edge features. In VAGNN, that method can be formalized as $\mathrm{EW}\left(v_i, v_j\right) = \frac{1}{E}\sum_{e=1}^{E} \mathbf{x}_e(\mathbf{w}_e^{(l,m)})^\top$, where E is the number of edges in (one of) the shortest path between v_i, v_j, the edges is indexed from v_i to v_j, $\mathbf{x}_e$ is a feature vector of the e-th edge, and $\mathbf{w}_e^{(l,m)}$ is the e-th trainable weight vector for $\mathrm{EW}\left(\cdot,\cdot\right)$.

The function $\mathrm{EXT}\left(\cdot,\cdot\right)$ provides significant flexibility for the architectural design of the proposed VAGNN. It should be noted that the main focus of this work is to demonstrate the generalization capabilities of VAGNN. We leave the study into the methods of $\mathrm{EXT}\left(\cdot,\cdot\right)$ for future research endeavors.

3.3.3 Virtual Adjacency Matrix

In this work, we investigate various existing GNNs and identify a common characteristic among these models. Specifically, we observe that many GNNs modify the original adjacency matrix for neighborhood information aggregation, with the purpose of improving model expressiveness or computational efficiency. For example, GraphSAGE [46] removes some of the actual edges (i.e., 1-hop neighbors) to enhance the computational efficiency of GNNs, where a fixed-size set of 1-hop neighbors is uniformly sampled from the full neighborhood sets. High-hop GNNs, such as DEGNN [77], MixHop [1], and KPGNN

[35], incorporate higher-hop neighboring nodes along with 1-hop nodes for neighborhood aggregation. Graph Transformers [32, 159, 156, 19] employ the self-attention mechanism that takes into account all nodes, where the matrix for the neighborhood aggregation is equivalent to $\{1\}^{N\times N}$. In graph theory, the adjacency matrix describes the actual connections between nodes in a graph. Nevertheless, these modified adjacency matrices employed for neighborhood aggregation in GNNs have lost that original functionality. Hence, we propose the term virtual adjacency matrix as a more appropriate descriptor for the modified matrix.

The virtual adjacency matrix is an important concept in our proposed VAGNN. As shown in Eq. 3.6, the virtual adjacency matrix serves as a mask that controls which node pairs can participate in the neighborhood information aggregation in the GNN. We study the methods for constructing the virtual adjacency matrix and categorize them into three main classes outlined below:

1. Removal of actual edges. This class of methods involves converting certain 1-hop neighbors into non-1-hop neighbors.

2. 1-hop conversion of high-hop local neighboring nodes. Here, high-hop local neighbors are transformed into 1-hop neighbors.

3. 1-hop conversion of global nodes. This class of methods focuses on converting global nodes into 1-hop neighbors.

The latter two classes of methods involve the addition of virtual edges between nodes, enabling high-hop neighbors and global nodes directly participate in the aggregation process of the central node. It is important to note that high-hop neighboring nodes and global nodes are different. High-hop neighbors are not directly connected to the central node, but they have small shortest-path distances to that node. In contrast, global nodes may have significantly larger shortest-path distances or even no path connection to the central node.

Diverse methods can be developed for the three aforementioned classes of methods for the virtual adjacency matrix construction. For example, as shown in Fig. 3.1, regarding the removal of actual edges, potential approaches could be random edge removal or edge removal based on weight. In the case of 1-hop neighbor conversion, the methods could include transforming all 2-hop local neighbors, selecting local nodes along a random path walk starting from the target node, converting partial global nodes, etc. Note that our work primarily centers around introducing VAGNN and its generalization capability. We defer the investigation of specific construction methods to future research. Furthermore, in the subsequent experimental section, we will evaluate the performance of several fundamental construction methods.

3.3.4 The Expressiveness of VAGNN

Compared to previous GNNs, the proposed VAGNN allows us to approach and design the GNN models from a more general perspective. Unlike conventional GNN models, VAGNN is not restricted to a specific architecture. Instead, it provides a general framework that is capable of being transformed into various existing GNNs and graph Transformers [64, 137, 156]. This generalization is achieved through customizable options in VAGNN, including designing the functions of $\mathrm{ATTN}(\cdot,\cdot)$ and $\mathrm{EXT}(\cdot,\cdot)$, and constructing the virtual adjacency matrix.

Specifically, when the virtual adjacency matrix is the original adjacency, VAGNN without any attention mechanism becomes pioneering GNN models like GCN [64]. Building upon this, the introduction of a residual connection gives rise to JKGNN [151], GCNII [22] and DeeperGCN [75], while the inclusion of an attention mechanism leads to the development of GAT [137], AGNN [132], and similar models. If the virtual adjacency matrix is constructed by removing certain edges from the original adjacency, it takes the form of GraphSAGE [46], FastGCN [21], etc. Alternatively, If the virtual adjacency matrix incorporates high-hop neighbors, it resembles DEGNN [77], MixHop [1] and KPGNN [35]. Lastly, if the virtual adjacency matrix consists entirely of *one*s, along with the self-attention mechanism and additional weights based on structural information, VAGNN is transformed into the architecture of Graph Transformers [32, 159, 156, 19].

In addition to its ability to be configured as existing GNNs and graph Transformers, VAGNN offers valuable assistance in designing novel and potentially more effective GNNs. Firstly, existing GNNs exhibit certain deficiencies. Almost all previous GNNs primarily concentrate on local neighbors within a certain hop, neglecting the incorporation of global nodes. On the other hand, graph transformers [167, 32, 156, 102] take into account all nodes in the process of neighborhood information aggregation, where each node attends to every other node during the self-attention computation. However, this design of graph transformers comes with several limitations. Firstly, the inclusion of numerous global nodes would excessively dilute the information from local neighbors, leading to the over-dilution issue [69]. Secondly, this setting makes the graph Transformer computationally expensive or even infeasible for large graphs [156]. Moreover, it can be likened to a language Transformer [135] taking the entire dictionary as input instead of a single sentence, which is unreasonable.

In contrast, VAGNN can address the aforementioned issues by constructing a virtual adjacency matrix that incorporates partial global nodes. This approach brings several benefits. Firstly, it enables a balance between the contributions of information from local and global nodes by controlling the proportion of global nodes used. Moreover, as opposed to traditional graph Transformers, the utilization of a virtual adjacency matrix with partial global nodes allows VAGNN to be implemented based on sparse matrix operations. This

can reduce the storage complexity from $O(N^2)$ and $O(N^3)$ of the dense matrix addition and multiplication (N is the number of total nodes) to a linear scale with the number of selected node pairs, thereby enabling VAGNN to exhibit excellent computational scalability even on large graphs. Additionally, by simultaneously employing three classes of methods for the virtual adjacency matrix construction (as discussed in Section 3.3.3) and optimizing the proportions of different nodes involved in those methods, we have the potential to achieve a superior model performance.

3.4 Experiments

This section presents the experimental evaluation of our proposed VAGNN model. First of all, it should be noted that the main objective of this work is to introduce the VAGNN model and demonstrate its powerful expressiveness and generalization capabilities, rather than pursuing state-of-the-art model performance. Therefore, we design the experiments to primarily serve the following purposes:

- Examination of generalization capabilities. We compare the performance of VAGNN with several representative GNNs and graph Transformers by configuring VAGNN to be transformed into these models.
- Sensitivity study on parameters of the virtual adjacency matrix construction. We investigate the parameter sensitivity of the methods for constructing the virtual adjacency matrix. Specifically, we focus on examining the influence of different proportions of 1-hop nodes, high-hop nodes, and global nodes on the model's performance.
- Study on attention methods. We assess the performance of the models with different attention methods.
- Exploration of the superior model performance. With the optimized parameters, we explore the potential of the most optimal VAGNN performance.

3.4.1 Datasets

All datasets used in our experiments are selected from the Open Graph Benchmark (OGB) [54]. We conduct experiments on three prominent graph tasks, i.e., node classification, link prediction, and graph classification. Accordingly, the datasets are ogbn-products, ogbn-proteins, and ogbn-arxiv for node classification; ogbl-ppa, ogbl-collab, and ogbl-ddi for link prediction; and ogbg-molhiv, ogbg-molpcba, and ogbg-code2 for graph classification.

OGB provides official evaluation protocols [54] for all benchmark datasets. We strictly follow these protocols for the training/validation/test data split and the evaluation metrics.

Specifically, For ogbg-molpcba, which comes with multiple classification tasks, Average Precision (AP) is used as the evaluation metric. For the datasets used in link prediction, the evaluation metric is Hits@k, which measures the proportion of positive samples that appear in the top-k positions in a ranked list of all (positive and negative) samples. We refer to [54] for further details about OGB datasets. All our experimental results are reported on the test set, with mean and standard deviation computed across 5 trials.

Node Classification Datasets

ogbn-products The ogbn-products is an undirected and unweighted graph that serves as a representation of the Amazon product co-purchasing network. Each node within the graph corresponds to a product available for purchase on Amazon, and edges between two products indicate that the products have been purchased together. The preprocessing methodology follows the work of [25] in terms of extracting node features and target categories. This process involves the generation of node features through the derivation of bag-of-words features from the product descriptions, followed by a subsequent application of Principal Component Analysis to condense the dimensions to a size of 100. The predictive objective revolves around categorizing products within a multi-class classification framework, where the 47 highest-level categories are employed as the target labels.

Dataset splitting of the dataset adheres to a more intricate and practical approach. It incorporates the notion of sales ranking, thereby engendering a dataset partition that mirrors real-world dynamics. The sales ranking, indicative of popularity, serves as the criterion for sorting products, and subsequently, the uppermost 8% of products are designated for training purposes. The subsequent 2% of products in the ranking hierarchy are allocated for validation, while the residual products constitute the testing set. This sophisticated partitioning procedure faithfully emulates real-world scenarios wherein labels are primarily ascribed to pivotal nodes within the network, subsequently enabling machine learning models to extrapolate predictions to nodes of relatively lower importance.

ogbn-proteins The ogbn-proteins is a comprehensive undirected, weighted, and type-enriched graph, with nodes symbolizing distinct proteins and edges denoting diverse biological interactions among proteins, such as physical interactions, co-expression, or homology [129, 26]. Each edge is associated with an 8-dimensional feature vector, wherein each dimension embodies the estimated confidence level of a specific association type, bounded between 0 and 1. Notably, higher values signify heightened confidence in the association. The protein entities originate from a spectrum of 8 distinct species.

The primary predictive task revolves around the detection of protein functions, orchestrated as a multi-label binary classification challenge encompassing a total of 112 distinctive labels. The assessment metric involves calculating the average ROC-AUC

scores across these 112 tasks. To ensure rigorous evaluation and robust model performance, the dataset undergoes strategic partitioning, wherein the protein nodes are segregated into training, validation, and test sets based on their respective species of origin. This partitioning strategy facilitates an insightful evaluation of model generalization across various species contexts.

ogbn-arxiv The ogbn-arxiv is a directed graph that captures the intricate web of citations among Computer Science (CS) arXiv papers as indexed by MAG [141]. Each node within this graph signifies an arXiv paper, while the directed edges depict the interconnections where one paper references another. In an endeavor to enhance the information within each node, a 128-dimensional feature vector is crafted by amalgamating embeddings derived from the titles and abstracts of the respective papers. The constituent word embeddings are artfully computed through the application of the skip-gram model [95] across the expansive landscape of the MAG corpus. Furthermore, MAG paper IDs are also mapped into the raw texts of titles and abstracts. This dataset further augments the richness node features by associating each paper with its year of publication.

The prediction task is to predict the 40 distinct subject areas to which arXiv CS papers belong, such as cs.AI, cs.LG, and cs.OS, which are manually assigned by the authors and the moderators of arXiv. Given the exponential surge in scientific publications over the past century, with the volume doubling every 12 years, the automation of categorizing the thematic essence of each publication stands as a pragmatic necessity. This endeavor formally translates into predicting the primary categories to which the arXiv papers belong, thus constituting a complex 40-class classification challenge.

In a bid to reflect the real-world dynamics, a pragmatic data partitioning strategy is undertaken based on the publication timelines of the papers. This configuration mirrors the customary practice where Machine Learning models are nurtured on extant papers and subsequently leveraged to forecast the thematic domains of freshly-minted publications. Such a configuration seamlessly lends itself to real-world applications, particularly in assisting arXiv moderators. To be precise, the proposition advocates for training on papers published up until 2017, validation on those unveiled in 2018, and culminating with testing on the corpus brought forth since the advent of 2019.

Link Prediction Datasets

In this work, we do not evaluate the performance of link prediction methods on previous small graph datasets such as Cora, Citeseer, Pubmed [177]. This is mainly because the different split proportions of train, validation, and test sets on these datasets would lead to different comparison results [33]. We refer readers to [33] for more details about the

Table 3.2 Statistics of OGB link prediction datasets used in our experiments.

Dataset	#Nodes	#Edges	#Degree
ogbl-ddi	4,267	1,334,889	500
ogbl-collab	235,868	1,285,465	8
ogbl-ppa	576,289	30,326,273	73
ogbl-citation2	2,927,963	30,561,187	21

issues of these traditional datasets. As an alternative, we conduct experiments on four link prediction datasets from OGB [54]. The statistics of datasets are summarized in Table 3.2.

ogbl-ddi The ogbl-ddi is a homogeneous, unweighted, undirected graph built based on a drug-drug interaction network [144]. In this graph, a node represents a drug and an edge describes an interaction between two drugs. The nodes in this graph do not have any features. OGB officially splits edges into train, validations, and test sets according to what proteins those drugs target in the body. The test set is composed of drugs that predominantly bind to different proteins from drugs in the train and validation sets, thereby evaluating the generalization capacity of the model for practical link prediction.

ogbl-collab The ogbl-collab is an undirected graph extracted based on a collaboration network between authors from MAG [141]. The nodes represent authors and the edges indicate the collaboration between authors. Every node has 128-dimensional features obtained by averaging the word embeddings of papers that are published by the authors [54]. Each edge comes with an attribute, i.e., the *year* when the co-authored paper is published. The task is to predict future author collaborations. The data is split according to time, aiming to simulate a realistic application in collaboration recommendation.

ogbl-ppa The ogbl-ppa is an undirected, unweighted graph. Nodes represent proteins from 58 different species, and edges are biologically meaningful associations between proteins [129]. The node features are obtained based on the species. OGB splits the data according to the type of protein associations, which meets the practical needs.

Graph Classification Datasets

ogbg-molhiv and ogbg-molpcba The ogbg-molhiv and ogbg-molpcba datasets are two molecular property prediction datasets derived from the MoleculeNet with different sizes: ogbg-molhiv (small) and ogbg-molpcba (medium). RDKit [72] is employed for the pre-processing of all molecules. Each graph in these datasets represents a unique molecule,

wherein atoms constitute the nodes and chemical bonds represent the edges. Notably, the node features encompass nine dimensions, including attributes like atomic number, chirality, formal charge, and ring membership.

The predictive objective centers around accurate forecasting of molecular properties, framed as binary labels. An illustrative instance is whether a molecule obstructs HIV virus replication. ROC-AUC is employed for evaluating ogbg-molhiv, while the evaluation metric for ogbg-molpcba integrates the Average Precision (AP) across tasks, attributed to the highly imbalanced class distribution (only 1.4% positive data). In order to partition the datasets, the scaffold splitting technique is adopted. By categorizing molecules based on their two-dimensional structural frameworks, this method ensures that structurally distinct molecules are distributed into diverse subsets. This approach augments the authenticity of model performance estimations, aligning with prospective experimental scenarios [149].

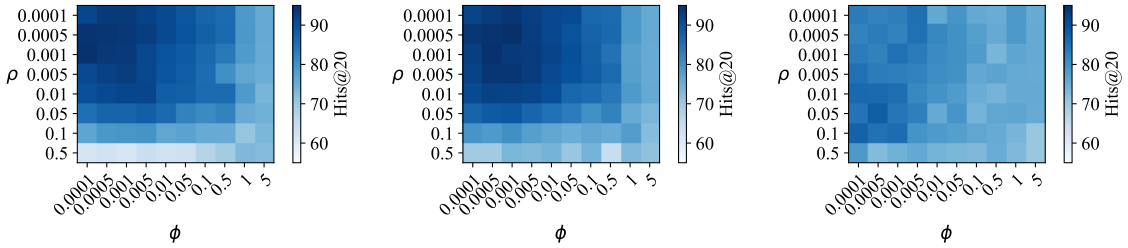
ogbg-code2 The ogbg-code2 dataset constitutes a comprehensive assemblage of Abstract Syntax Trees (ASTs), drawn from a vast repository of approximately 450,000 Python method definitions. These methods are culled from a diverse array of 13,587 repositories spanning the GitHub spectrum's most renowned projects. The wellspring of our Python method compendium emerges from GitHub's CodeSearchNet initiative - a compendium of datasets and benchmarks tailored for machine learning-based code retrieval. Notably, ogbg-code2 comes with additional features including AST edges, AST nodes, and tokenized method names. Through this augmentation, ogbg-code2 empowers us to encapsulate source code in its intrinsic graph architecture, transcending the confines of its token-based sequence representation.

The primary predictive task entails predicting the sub-tokens that collectively constitute the method name. The evaluation adheres to the methodology outlined in [5, 6], employing the F1 score to evaluate the alignment between predicted sub-tokens and ground-truth sub-tokens. Dataset splitting follows a prudent project-oriented approach [4], whereby the training set's ASTs originate from GitHub projects unrepresented in both the validation and test sets. This design mirrors the pragmatic workflow of training a model on an extensive repository of source code and subsequently deploying it to predict method names within a distinct codebase. This project-based partitioning scheme serves as a robust litmus test for the model's capacity to apprehend the nuances of code semantics, eschewing reliance on the specificities of training projects, such as idiosyncratic naming conventions and coding styles unique to particular developers.

3.4.2 Implementation

We implement our algorithm based on PyTorch, PyTorch Geometric [36], and OGB [54]. The model is trained with Adam optimizer [63] with the learning rate decayed using the

ExponentialLR method [79]. All experiments are conducted on a Linux machine with two 32-core CPUs, 512G RAM, and two NVIDIA A100 (40G) GPUs. Code is available at *https://github.com/2023researchup/GNNHE*.



(a) 1-hop (b) 2-hop (c) 3-hop

Figure 3.2 The results of the parameter sensitivity study on the virtual adjacency matrix construction. The three plots (a) (b) (c) correspond to three different configurations for the 1-hop conversion of high-hop local neighboring nodes. For example, in plot (a), the term "1-hop" represents the virtual adjacency matrix that uses the local neighboring nodes within 1 hop (i.e., only the actual 1-hop nodes). In plot (b), the term "2-hop" refers to the conversion of local neighboring nodes within 2 hops into 1-hop neighbors in the virtual adjacency matrix. The colors of grid cells in the plots indicate the performance (Hits@20) of the proposed VAGNN on the *ogbl-ddi* dataset.

3.4.3 Examination of VAGNN Generalization Capabilities

This work introduces a general GNN framework, i.e., VAGNN, with the purpose of bridging the current gaps among various architectures of GNNs and graph Transformers [64, 137, 156]. The attainability of this purpose stems from the commonality of all these models belonging to the family of the neighborhood information aggregation algorithm.

In order to experimentally examine the generalization capability of VAGNN, we carefully choose several representative GNN models, including:

1. GCN [64]. It uses all 1-hop nodes for neighborhood information aggregation without any attention weights.

2. GraphSAGE [46]. It is similar to GCN, but it differs in the selection of the neighborhood set. Instead of utilizing all 1-hop neighbors as done in GCN, GraphSAGE employs a fixed-size neighborhood set sampled uniformly from the 1-hop neighbors.

3. DEGNN [77]. It is a high-hop GNN framework where the neighboring nodes attending the central node's information aggregation are selected based on the shortest-path distance between the neighbors and the central node. The distance in DEGNN is a hyper-parameter. In this work, we specifically set this hyper-parameter as 2, which corresponds to a 2-hop GNN configuration.

Table 3.3 Results of node classification.

	ogbn-products (Accuracy %)	**ogbn-proteins (AUC %)**	**ogbn-arxiv (Accuracy %)**
GCN [64]	75.64 ± 0.21	76.53 ± 0.35	71.74 ± 0.29
VAGNN(GCN)	75.77 ± 0.19	77.31 ± 0.41	71.38 ± 0.31
GraphSAGE [46]	78.29 ± 0.16	77.68 ± 0.20	71.49 ± 0.07
VAGNN(SAGE)	78.03 ± 0.35	78.04 ± 0.37	71.25 ± 0.18
DEGNN [77]	80.14 ± 0.64	85.01 ± 0.72	71.62 ± 0.17
VAGNN(DE)	80.37 ± 0.91	85.17 ± 0.65	71.93 ± 0.21
AGNN [132]	78.84 ± 0.35	79.11 ± 0.35	71.81 ± 0.37
VAGNN(AGNN)	78.92 ± 0.43	79.02 ± 0.34	71.74 ± 0.57
GAT [137]	79.23 ± 0.78	79.35 ± 0.44	71.97 ± 0.35
VAGNN(GAT)	79.19 ± 0.59	79.41 ± 0.38	71.86 ± 0.25
Graphormer [156]	–	–	–
VAGNN(Trans)	81.14 ± 0.67	80.73 ± 0.51	72.14 ± 0.45
VAGNN(mix+Cos)	81.53 ± 0.57	86.17 ± 0.28	71.64 ± 0.29
VAGNN(mix+GAT)	82.91 ± 0.62	87.29 ± 0.52	72.06 ± 0.77
VAGNN(mix+Trans)	84.26 ± 0.79	88.71 ± 0.69	72.85 ± 0.63

Table 3.4 Results of link prediction

	ogbl-ppa (Hits@100 %)	**ogbl-collab (Hits@50 %)**	**ogbl-ddi (Hits@20 %)**
GCN [64]	54.19 ± 1.21	49.35 ± 0.47	88.42 ± 1.34
VAGNN(GCN)	54.23 ± 1.17	49.19 ± 0.53	88.52 ± 2.83
GraphSAGE [46]	49.07 ± 2.18	48.10 ± 0.81	87.37 ± 3.68
VAGNN(SAGE)	50.15 ± 2.73	48.39 ± 0.75	88.09 ± 4.91
DEGNN [77]	51.33 ± 0.87	50.29 ± 0.27	89.94 ± 5.23
VAGNN(DE)	51.61 ± 1.31	50.29 ± 0.32	89.17 ± 3.94
AGNN [132]	54.58 ± 2.56	49.24 ± 0.48	89.61 ± 3.54
VAGNN(AGNN)	54.37 ± 2.10	49.17 ± 0.26	90.02 ± 3.84
GAT [137]	56.27 ± 3.25	49.17 ± 0.55	91.21 ± 5.98
VAGNN(GAT)	55.98 ± 2.79	49.33 ± 0.19	90.79 ± 7.12
Graphormer [156]	–	–	–
VAGNN(Trans)	53.29 ± 4.41	50.05 ± 0.72	92.31 ± 6.25
VAGNN(mix+Cos)	55.37 ± 2.46	50.91 ± 0.37	91.42 ± 2.57
VAGNN(mix+GAT)	57.68 ± 3.28	52.62 ± 0.73	92.68 ± 5.51
VAGNN(mix+Trans)	59.73 ± 3.35	53.29 ± 0.65	94.07 ± 3.66

Table 3.5 Results of graph classification

	ogbg-molhiv (AUC %)	**ogbg-molpcba (AP %)**	**ogbg-code2 (F1 Score %)**
GCN [64]	76.06 ± 0.97	23.96 ± 0.41	15.07 ± 0.18
VAGNN(GCN)	76.25 ± 0.64	23.72 ± 0.70	15.43 ± 0.35
GraphSAGE [46]	76.35 ± 1.32	25.05 ± 0.17	15.13 ± 0.39
VAGNN(SAGE)	76.18 ± 1.09	25.17 ± 0.31	15.36 ± 0.25
DEGNN [77]	79.84 ± 1.25	28.15 ± 0.41	15.81 ± 0.21
VAGNN(DE)	79.62 ± 1.47	28.26 ± 0.63	15.78 ± 0.25
AGNN [132]	76.36 ± 1.13	26.84 ± 0.22	15.37 ± 0.15
VAGNN(AGNN)	76.41 ± 1.05	27.08 ± 0.31	15.26 ± 0.12
GAT [137]	76.81 ± 1.51	27.03 ± 0.29	15.69 ± 0.10
VAGNN(GAT)	76.83 ± 1.29	27.14 ± 0.33	15.57 ± 0.21
Graphormer [156]	80.51 ± 0.53	31.39 ± 0.32	17.52 ± 0.14
VAGNN(Trans)	80.95 ± 1.07	31.58 ± 0.51	17.71 ± 0.23
VAGNN(mix+Cos)	78.91 ± 1.47	29.12 ± 0.27	16.14 ± 0.21
VAGNN(mix+GAT)	79.49 ± 1.62	29.88 ± 0.54	17.92 ± 0.23
VAGNN(mix+Trans)	80.94 ± 1.14	31.72 ± 0.48	18.51 ± 0.19

4. AGNN [132]. It is a 1-hop GNN and employs a cosine similarity attention mechanism as shown in Eq. 3.7.

5. GAT [137]. It is a 1-hop GNN that incorporates an attention mechanism as shown in Eq. 3.8.

6. Graphormer [156]. It follows the basic architecture of NLP Transformer [135] and employs self-attention on all nodes.

We use the term VAGNN(GCN) to denote the VAGNN that has been configured exactly according to the GCN. Similarly, other VAGNN variations, such as VAGNN(SAGE), follow the same nomenclature convention. Note that VAGNN(Trans) employs the same attention mechanism as Graphormer. However, we utilize partial global nodes for each node in VAGNN(Trans) when using all nodes to calculate the self-attention is challenging on some datasets.

Tables 3.3, 3.4, and 3.5 present the results of node classification, link prediction, and graph classification, respectively. The results of the VAGNN variants closely align with those of the corresponding GNN models across most datasets, demonstrating the effectiveness of the VAGNN in generalizing various GNN architectures within a single framework. These results also confirm that the majority of GNNs and graph Transformers are built and operated within the form of neighborhood information aggregation (Equation 1).

3.4.4 Parameter Sensitivity Study on Virtual Adjacency Matrix

The virtual adjacency matrix serves as a core component of VAGNN, governing the participation of nodes in neighborhood information aggregation. As outlined in Section 3.3.3, the methods used to construct the virtual adjacency matrix can be categorized into three classes, which correspond to the control over the proportions of three types of nodes involved in the aggregation, i.e., actual 1-hop neighboring nodes, high-hop local neighbors, and global nodes. In this parameter sensitivity analysis, our objective is to investigate how different proportions of these three classes of nodes affect the performance of the model. To effectively demonstrate this, we utilize three commonly-used methods with concise parameters. Specifically, as shown in Fig. 3.1, for the removal of 1-hop neighbors, we randomly eliminate $\rho|\mathbb{E}|$ actual edges (equivalent to converting 1-hop neighbors to non-1-hop nodes). Herein, $|\mathbb{E}|$ represents the total number of actual edges. Moreover, regarding the 1-hop conversion of high-hop local neighbors, we explore three settings: using the original 1-hop neighbors, converting all local neighboring nodes within 2 hops into 1-hop neighbors, and converting those within 3 hops into 1-hop. Lastly, for the 1-hop conversion of global nodes, we randomly select $\phi|\mathbb{E}|$ global nodes.

We use the dataset *obgl-ddi* to investigate the sensitivity of parameters. The results are presented in Fig. 3.2. Regarding the parameter ρ, it can be observed that VAGNN attains its optimal performance at relatively small values of ρ, with the majority of peaks occurring around 0.0005 of ρ. As ρ increases, the performance of VAGNN gradually declines and becomes particularly poor when $\rho \geq 0.1$. The lowest performance of VAGNN is observed when $\rho = 1$, which can be explained by the fact that setting $\rho = 1$ results in the removal of all 1-hop neighboring nodes. These results emphasize the critical significance of the information contained in 1-hop neighboring nodes in the GNN modeling.

In the context of the 1-hop conversion of high-hop local neighboring nodes, the results on the 1-hop and 2-hop (Fig. 3.2a and Fig. 3.2b) are significantly better than the results on the 3-hop (Fig. 3.2c). Remarkably, the most favorable results are observed on the 1-hop conversion of 2-hop nodes (Fig. 3.2b), suggesting that the incorporation of high-hop nodes could effectively improve the performance of the VAGNN model.

The results on the 1-hop conversion of 3-hop neighbors exhibit minimal variance. This result can be attributed to the characteristics of the ogbl-ddi dataset. Specifically, the ogbl-ddi is a graph with 4267 nodes and an average node degree of around 500. We observe that for every node in ogbl-ddi, its neighbors within 3 hops would approximately account for 95 percent of the entire graph's nodes. Consequently, the virtual adjacency matrix based on the 1-hop conversion of 3-hop neighbors on the ogbl-ddi dataset is similar to the standard graph Transformer, where all nodes are considered as 1-hop neighbors in the virtual adjacency matrix.

For the parameter ϕ, Fig. 3.2 demonstrates that smaller values of ϕ generally yield superior performance, while larger values of ϕ lead to a noticeable decline in the results. Moreover, when $\phi \geq 1$, the model exhibits poor and similar outcomes. This could be attributed to the dense nature of the graph in the ogbl-ddi dataset (4267 nodes and a degree of around 500). When $\phi \geq 1$, nearly all global nodes for each node would be converted into its 1-hop neighbors in the virtual adjacency matrix. In many real-world graphs, the number of global nodes is often far larger than that of local neighboring nodes for a node. Hence, the meticulous selection of an appropriate number of global nodes for the virtual adjacency matrix holds crucial importance.

We also extend the parameter study to the remaining datasets used in this work. The observed parameter sensitivity on these datasets exhibits a consistent pattern that aligns with the results presented on the ogbl-ddi dataset.

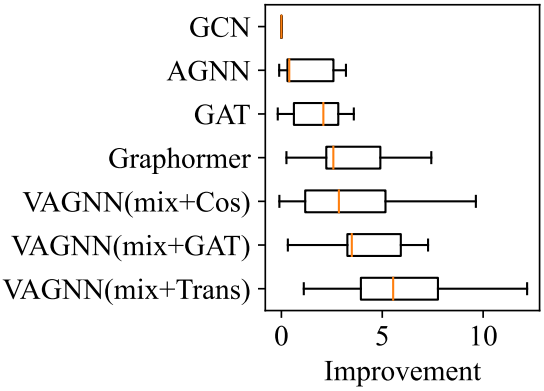


Figure 3.3 The performance improvement of attention-based GNN models over the baseline GCN [64]. Boxplots present the statistical analysis based on the results obtained from the nine datasets used in this work.

3.4.5 Attention Methods Assessment

With the goal of guiding the selection of the attention mechanisms for our proposed VAGNN, we conduct a study on the empirical performance of the attention-based GNN models, including three attention mechanisms: cosine similarity-based, GAT-type, and self-attention. We take the results of GCN [64] as the baseline and statistically analyze the improvements of the attention-based models over this baseline across nine datasets. The analytical results are presented in Fig. 3.3. It can be seen that the self-attention-based models (Graphormer [156] and VAGNN(mix+Trans)) demonstrate more enhancement, followed by the GAT-type ones [137], while the models with cosine similarity-based attention, including AGNN [132] and VAGNN (mix+Cos), show relatively minimal improvements.

As discussed in Section 3.3.1, it is worth noting that the GAT-type attention mechanism [137] theoretically requires more computational resources than the self-attention mecha-

nism [159, 156, 19]. In our experiments, to avoid exceeding the computational capacity of the machine, the dimensions of the representations used for computing GAT-type attention often must be set smaller than those used for the self-attention mechanism, which would partially limit the expressive capability of GAT-type models and subsequently impact their performance. Overall, all these results suggest that the self-attention mechanism may be a more favorable choice for the GNN design compared to the other two mechanisms.

3.4.6 Exploration of the Superior Model Performance

To attain the optimal performance of VAGNN, we carefully select the most effective combination of parameters for constructing the virtual adjacency matrix. We also employ the extra weight as shown in Eq.3.10 where we follow the methods in [156]. Together with different attention mechanisms, we denote such optimal VAGNN models as VAGNN(mix+Cos), VAGNN(mix+GAT), or VAGNN(mix+Trans).

In Tables 3.3, 3.4, and 3.5, it can be found that our VAGNN(mix+Trans) consistently achieves superior performance across most datasets. Moreover, by combining carefully-selected parameters for the virtual adjacency matrix and the attention mechanism, VAGNN surpasses the results obtained by previous GNN models that solely rely on a fixed "virtual" adjacency matrix (e.g., GAT [137] with a 1-hop matrix). These outcomes serve as compelling evidence for the significance of our optimizable virtual adjacency matrix.

VAGNN provides a flexible mechanism for customizing the proportions of diverse types of nodes for neighborhood aggregation in GNNs. The utilization of VAGNN in achieving superior outcomes yields insightful findings that are highly beneficial for researchers and practitioners, providing them with valuable knowledge and practical guidance for effectively implementing these techniques in various scenarios. VAGNN also opens up avenues for future exploration, particularly in the development of more efficient approaches for constructing the virtual adjacency matrix. This line of research holds substantial potential for advancing the design of more robust GNNs in forthcoming studies.

3.5 Summary

In this chapter, we present VAGNN, a general and scalable GNN framework. With an optimizable virtual adjacency matrix, VAGNN offers the capability to customize the neighborhood for aggregation, allowing for the selective exclusion of 1-hop neighbors and the incorporation of high-hop local neighbors and global nodes. We also describe the flexibility of VAGNN in terms of selecting attention mechanisms for information aggregation and incorporating supplementary information into the attention weight. By customizing the construction of the virtual adjacency matrix and utilizing various attention mechanisms, VAGNN can be transformed into most existing GNNs and transformers, demonstrating the

generalization of VAGNN. Furthermore, we discuss the linear computational complexity of VAGNN, which enables its scalability for handling large graphs. Additionally, we perform a parameter sensitivity analysis, highlighting the importance of carefully selecting and balancing the inclusion of local and global information in GNNs. VAGNN provides an effective tool for researchers and practitioners in the field of GNNs. It paves the way for designing more efficient techniques for virtual adjacency matrix construction and weighted aggregation functions, which will contribute to the development of more effective GNN models in the future.

Chapter 4

An Interpretive Analysis of GNNs in Link Prediction

This chapter investigates the link prediction capability of GNNs. Our research reveals that GNNs lack the model ability to learn structural information related to the number of common neighbors between two nodes, primarily due to the nature of set-based pooling of the neighborhood aggregation scheme. Furthermore, our extensive experimental analysis indicates that when the training is supervised by positive and negative link samples, trainable node embeddings can improve the performance of the GNN-based link prediction model. Importantly, we observe that the denser the graph, the greater the improvement. We attribute this to the characteristics of node embeddings, where the state of each link sample could be encoded into the embeddings of nodes that are involved in the neighborhood aggregation of the two nodes in that link sample. In denser graphs, every node could have more opportunities to attend the neighborhood aggregation of other nodes and encode states of more link samples to its embedding, thus learning better node embeddings for link prediction. Lastly, we demonstrate that the insights gained from our research carry important implications in identifying the limitations of existing methods, which could guide future research efforts toward the development of more robust link prediction algorithms.

4.1 Introduction

GNNs have demonstrated powerful expressiveness in graph representation learning [173]. However, what structural information can be learned via GNNs remains an open question. In the literature, this question was predominantly studied in terms of graph-level, e.g., graph isomorphism [150, 24, 39, 13] by comparing GNNs and Weisfeiler-Lehman test [143]. Few papers have investigated this open question in terms of structural information specific to two nodes. A prominent task relying on such information is link prediction. Existing GNN-based link prediction works [168, 124, 142, 171, 44, 145] primarily pay

their attention to the model design. Nevertheless, rare of them touch on the core question: Do their models really learn pair-specific structural information for link prediction?

For example, SEAL [168] and its successors [77, 131, 154] are a family of link prediction methods that attempt to use GNNs to learn pair-specific structural information represented by traditional link heuristics such as Common Neighbors, Katz index [62], and SimRank [60]. SEAL has proven that most link heuristics between two nodes can be computed approximately within an enclosing subgraph specifically constructed for the two nodes [168]. SEAL-type methods also assign a labeling vector as additional features to each node in such subgraph based on the relationship of each node to the target two nodes [170]. Then these methods perform link prediction by using GNNs to classify such enclosing subgraphs, with the expectation that GNNs can learn the structural information equivalent to link heuristics from the enclosing subgraph. However, few of these works have examined whether this expectation holds true, and our research indicates a negative answer to this expectation.

In this work, we present analytical and empirical investigations into the link prediction capability of GNNs, with a focus on a fundamental question[1]: *whether GNNs can effectively learn the pair-specific structural information related to the number of common neighbors for link prediction*? and by exploring our experimental observations: *experimentally, node embeddings can improve the performance of GNN-based link prediction models, and the denser the graph, the more the improvement.* We consider the latter exploration to be one of our primary contributions because 1) existing link prediction works rarely reveal and delve into these observations and 2) these empirical findings come with significant practical implications for selecting appropriate link prediction methods based on graph density. Without sacrificing the generality of our research, we focus our work on the context of link prediction.

Prior to presenting our research, we hereby declare that the primary objective of this research work is to investigate fundamental problems in GNNs for the task of link prediction. Therefore, the GNN algorithms employed in this work are basic and general. It is important to note that this work does not seek to propose innovative model architectures or suggest novel applications for GNNs. Instead, we focus on analyzing the existing models, identifying their strengths and weaknesses, and providing insights into link prediction for potential improvement. We believe that an in-depth understanding of fundamental algorithms is imperative for developing effective and efficient GNN models in the future.

First, modern GNNs follow a neighborhood aggregation scheme, where each node's representation is recursively updated by aggregating the representations of that node and

[1]There exists multiple types of structural information specific for two nodes in a graph. In this work, we do not attempt to study all of them and measure which ones can be learned or not via GNNs. This is because it requires the interpretation of the GNN which remains an open question and falls outside the scope of this research. Instead, our primary objective is to study the limitation of GNNs in learning an important type of structural information that highly relies on the number of common neighbors.

its neighbors [27]. The learned representations are node-wise. It has been recognized that each node's representation can hardly capture information related to the number of its neighbors due to the nature of the set-based pooling of the aggregation scheme which inherently ignores the size of the neighborhood set of each node [150, 170].

A general strategy for applying node-wise representations learned by GNNs to downstream multiple-node tasks (e.g., link prediction, graph classification, etc.) is to combine the representations of the nodes involved in these tasks. For link prediction, we find that the combination of two nodes' representations essentially lacks the ability to capture information related to the number of common neighbors. This is mainly because node-wise representations learned by GNNs inherently lack information about the number of neighbors of each node, and most operations of combining two nodes' representations (e.g., concatenation, Hadamard production, etc.) also do not contain any behaviors of counting how many common neighbors between two nodes.

To empirically verify this, we adopt an approach of incorporating traditional link heuristics (e.g., Common Neighbors) into the GNN and examine the link prediction performance. The approach yields results either superior or comparable to those obtained by using only GNNs, experimentally supporting our analysis. Notably, this approach helps us achieve state-of-the-art on two Open Graph Benchmark (OGB) [54] datasets, i.e., ogbl-collab and ogbl-citation2.

In our experiments, we find that trainable node embeddings can enhance the performance of GNN-based link prediction models, and the denser the graph, the stronger the enhancement. In particular, by only utilizing node embeddings in GCN [64] or GAT [136], we are able to surpass all previous best results on two dense graphs ogbl-ddi and ogbl-ppa, achieving 96.21, and 63.51, respectively.

Our explanation for these experimental observations is as follows. Compared to the model weights of a GNN that are shared across all nodes [64, 27], each trainable node embedding is unique to its respective node. This characteristic of node embeddings can benefit the model. When the training is supervised by positive and negative link samples (i.e., two nodes are not linked), the link state of two nodes in every link sample could be encoded into the node embeddings of that two nodes and their neighboring nodes with the help of the neighborhood aggregation algorithm of the GNN. This would enable each node embedding to remember the relationships of that node to other nodes, allowing the model to know better which two nodes are more likely to be linked or not. Moreover, in the neighborhood aggregation of the GNN, the denser graphs would allow each node to see more other nodes, leading to better learning of node embeddings for link prediction.

This study provides deeper insights into the expressiveness of GNNs in link prediction. These insights can help identify and interpret the limitations of existing link prediction methods, potentially directing the search for more robust algorithms. To demonstrate this, we present two analytical case studies: first, we show that SEAL-type methods [168, 131]

Table 4.1 NCN-dependent link heuristics between nodes v, u.

Heuristic	Definition
$\text{CN}_{v,u}$	$\lvert\Gamma_v \cap \Gamma_u\rvert$
$\text{JA}_{v,u}$ [59]	$\frac{\lvert\Gamma_v \cap \Gamma_u\rvert}{\lvert\Gamma_v \cup \Gamma_u\rvert}$
$\text{AA}_{v,u}$ [2]	$\sum_{z \in \Gamma_v \cap \Gamma_u} \frac{1}{\log\lvert\Gamma_z\rvert}$
$\text{RA}_{v,u}$ [174]	$\sum_{z \in \Gamma_v \cap \Gamma_u} \frac{1}{\lvert\Gamma_z\rvert}$
SPD	Shortest Path Distance
$\text{Katz}_{v,u}$ [62]	$\sum_{l=1}^{\infty} \beta^l \lvert\{\text{path}_{v,u}^{(l)}\}\rvert$

are unable to effectively learn information related to the number of common neighbors. Second, we find that NBFNet [177] lacks the algorithmic capability to train powerful node embeddings for link prediction. In addition, we empirically compare the performance of multiple types of link prediction methods on OGB datasets. The results can be explained with our insights, further underlining the significance of our findings.

4.2 Preliminaries

Without loss of generality, we demonstrate our work on homogeneous graphs [52].

4.2.1 Problem Definition

Link prediction is a node-pair-specific problem, aiming to estimate the likelihood $\hat{y}_{v,u}$ of the existence of an unknown edge $\mathcal{E}_{v,u} \notin \mathbb{E}$ between two nodes $v, u \in \mathbb{V}$. Herein we refer to v, u as *two target nodes* in the *candidate link* $\mathcal{E}_{v,u}$.

4.2.2 NCN-dependent Structural Information

In this research, we distinguish an important type of node-pair-specific structural information, i.e., *NCN-dependent* structural information from others, where *NCN* refers to the *N*umber of *C*ommon *N*eighbors between two nodes.

First of all, we use several traditional link heuristics[2] to instantiate NCN-dependent structural information. These heuristics include Common Neighbors (CN), Jaccard (JA) [59], AdamicAdar (AA) [2], Resource Allocation (RA) [174], Katz index [62], etc. Table 4.1 lists the definitions of these heuristics. Common Neighbors (CN) is defined as the size of the intersection of the first-order neighborhood sets of two nodes. Jaccard (JA)

[2]In this work, we do not pursue collecting and reviewing all link heuristics. Instead, we only use several of them for the convenience of presenting our study on GNNs.

coefficient [59] normalizes the CN by the size of the union of the two nodes' neighborhood sets. AdamicAdar (AA) [2] and Resource Allocation (RA) [174] both suppress the contribution of nodes by penalizing each node with its degree. SPD is the shortest path distance between two nodes. Katz index [62] weighted counts all paths between two nodes, where $|\{\text{path}_{v,u}^{(l)}\}|$ is the number of all paths between node v and u with the length of l, and β is a damping factor satisfying $0 < \beta < 1$.

We can see from Table 4.1 that the first four heuristics highly rely on the number of common neighbors between two nodes v, u (i.e., $|\Gamma_v \cap \Gamma_u|$). Theoretically, Shortest Path Distance (SPD) and Katz index [62] also capture NCN-dependent structural information. For example, $\text{SPD}_{v,u} = 3$ means that $\text{CN}_{v,u} = 0$ and at least one node in Γ_u has at least one common neighbor with v. For $l = 4$ in Katz, $|\{\text{path}_{v,u}^{(4)}\}|$ can be computed by $|\{\text{path}_{v,u}^{(4)}\}| = \sum_{a\in\Gamma_v, b\in\Gamma_u} \text{CN}_{a,b}$.

Algorithm 2 SimRank [60]

1: **Input:** Graph $\mathscr{G} = (\mathbb{V}, \mathbb{E})$ ($|\mathbb{V}| = N$), decay factor C ($0 < C < 1$), iterations K
2: **Output:** Similarity $\mathbf{S} = (s_{i,j}) \in \mathbb{R}^{N\times N}$
3: **Initialize:** $s_{i,j}^{(0)} = 1$ if $i = j$, otherwise 0
4: **for** $m = 1$ **to** K **do**
5: $\quad s_{ij}^{(m)} = \frac{C}{|\Gamma_i||\Gamma_j|} \sum_{b=1}^{|\Gamma_j|} \sum_{a=1}^{|\Gamma_i|} s_{\Gamma_i(a)\Gamma_j(b)}^{(m-1)}$, where $\Gamma_i(a)$ is the a-th node in Γ_i
6: **end for**

Besides, we introduce SimRank [60], a link heuristic with limited capability to extract NCN-dependent structural information. As shown in Algorithm 2, SimRank recursively refines similarity scores between every two nodes by considering the neighboring nodes of the two nodes, where the number of common neighbors is ignored essentially. Specifically, we can see from the core function in Line 5 of Algorithm 2 that the similarity score $s_{i,j}^{(m)}$ between node i, j in the m-th iteration is obtained by averaging the similarity scores between all neighbors of i and j from the $(m-1)$-th iteration, where the information about how many common neighbors between i, j can hardly be encoded into $s_{i,j}^{(m)}$. We will demonstrate that the SimRank learning style exhibits certain similarities to the GNN.

4.2.3 GNNs in Learning the Number of Neighbors

Modern GNNs follow a neighborhood information aggregation algorithm where the representation of each node in a graph is iteratively updated by aggregating the representations of its neighbors and its own [27]. Formally, the representation of a node i updated by the

l-th layer of a GNN is

$$\begin{aligned}\hat{\mathbf{h}}_i^{(l)} &= \mathrm{AGG}^{(l)}\left(\left\{\mathbf{h}_j^{(l-1)} \mid \forall j \in \Gamma_i \cup \{i\}\right\}\right),\\ \mathbf{h}_i^{(l)} &= \hat{\mathbf{h}}_i^{(l)}\mathbf{W}^{(l)},\end{aligned} \tag{4.1}$$

where $\mathbf{h}_i^{(0)}$ is initialized with the feature vector of node i, $\mathrm{AGG}^{(l)}(\cdot)$ is instantiated as a set-based pooling operation such as MAX, MEAN, or attention-based SUM [47, 136], $\mathbf{W}^{(l)}$ is a weight matrix for the l-th GNN layer, which is shared across all nodes and used for representation transformation (i.e., if $\hat{\mathbf{h}}_v^{(l)} \in \mathbb{R}^f, \mathbf{W}^{(l)} \in \mathbb{R}^{f \times f'}$, then $\mathbf{h}_v^{(l)} \in \mathbb{R}^{f'}$).

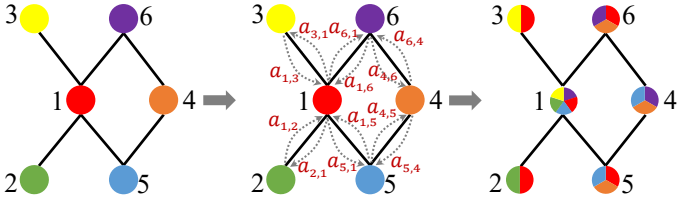


Figure 4.1 An illustration of neighborhood information propagation and aggregation in GNNs, where $a_{i,j}$ can be an edge weight or attention weight from node j to i.

Fig. 4.1 illustrates the neighborhood information propagation and aggregation process in GNNs. In Fig. 4.1, $\hat{\mathbf{h}}_i^{(l)}$ in Equation 4.1 can be computed by

$$\hat{\mathbf{h}}_i^{(l)} = \sum_{j \in \Gamma_i \cup \{i\}} \frac{a_{i,j}^{(l)}}{\sum_{j \in \Gamma_i \cup \{i\}} a_{i,j}^{(l)}} \mathbf{h}_j^{(l-1)}, \tag{4.2}$$

where $a_{i,j}^{(l)}$ is the weight for the message (i.e., $\mathbf{h}_j^{(l-1)}$) from node j to i. For MEAN-pooling in GNNs like GCN [47], it can be $a_{i,j}^{(l)} = 1, \forall \mathscr{E}_{i,j} \in \mathbb{E}$. For attention-based SUM in GNNs like GAT [136], $a_{i,j}^{(l)}$ is an attention coefficient that is computed dynamically based on $\mathbf{h}_i^{(l-1)}$ and $\mathbf{h}_j^{(l-1)}$ during the training process.

Theorem 1. *The node representations learned by aggregation-based GNNs are node-wise.*

Proof. As shown in Eq. 4.1, the input, intermediate, and output representations of GNNs are node-wise. □

Proposition 1. *Each node's representation learned by GNNs lacks information about the number of neighboring nodes of that node.*

Proof. As shown in Eq. 4.1, GNNs update the representation $\mathbf{h}_i^{(l)}$ by aggregating the representations of node i and its neighbors. In this process, the number of neighbors of node i can hardly be encoded into $\mathbf{h}_i^{(l)}$. This is due to the inherent nature of neighborhood

aggregation scheme in GNNs, i.e., the $\text{AGG}^{(l)}(\cdot)$ in Eq. 4.1 is set-based pooling, aiming to handle irregular sizes of neighborhood sets of different nodes. For example, if the aggregation is MEAN pooling, then the set of representations (i.e., $\{\mathbf{h}_j^{(l-1)} \mid \forall j \in \Gamma_i \cup \{i\}\}$ in Eq. 4.1) will be averaged and the aggregation result could hardly contain information about the size of this set.

Note that attention-based pooling also cannot address this inherent issue of neighborhood aggregation. As shown in Eq. 4.2, attention-based GNNs like GAT [136] essentially replace the original edge weight with attention weight. The set of representations is actually weighted averaged and the result still lacks information related to the size of the neighborhood set of node i. □

Proposition 1 shows that the neighborhood aggregation algorithm of GNNs inherently cannot effectively learn information about the number of neighbors of each node. Essentially, we can address this issue by, for example, adding the node degree as a feature to each node. We note that several previous works have pointed out this inherent issue of GNNs [150, 170]. We present it formally using Proposition 1 for better presenting our following study.

4.3 NCN-dependent Structural Information cannot be Effectively Learned via GNNs

4.3.1 Analytical Study

What can we do when applying the node-wise representations learned by GNNs to downstream graph tasks that involve multiple nodes, such as link prediction or graph classification? A standard way is to combine the representations of the involved nodes into one representation and pass it into the next model parts [150, 142]. For such a combination, we have the following proposition:

Proposition 2. *The combination of two or more nodes' representations learned by GNNs cannot effectively capture NCN-dependent structural information.*

Proof. According to Proposition 1, due to the inherent nature of the neighborhood aggregation algorithm, node-wise representations learned by GNNs cannot effectively capture information related to the number of neighbors of each node, much less to the number of common neighbors between two nodes. The operation of combining representations of two or more nodes also cannot effectively extract NCN-dependent structural information. For example, we can combine the representations of two nodes by concatenation, Hadamard production, etc. [142] and combine more nodes' representations by MEAN pooling

Algorithm 3 GNN-based link prediction

Input: Graph $\mathscr{G} = (\mathbb{V}, \mathbb{E}, \mathbf{X})$ ($|\mathbb{V}| = N$), $\mathbf{X} \in \mathbb{R}^{N \times f}$, trainable node embeddings $\mathbf{E} \in \mathbb{R}^{N \times d}$, ground truth $y_{v,u}$ for link sample (v,u), GNN layers L, epochs K
Output: Link likelihood $\hat{y}_{v,u} \in \mathbb{R}$ for node pair v,u
Initialize: node embeddings $\mathbf{E}$, model weights, etc.
for $i = 0$ **to** K **do**
    for $l = 1$**to** L **do**
        $\hat{\mathbf{h}}_i^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{\mathbf{h}_j^{(l-1)} \mid \forall j \in \Gamma_i \cup \{i\}\right\}\right)$
        $\mathbf{h}_i^{(l)} = \hat{\mathbf{h}}_i^{(l)} \mathbf{W}^{(l)}$
    end for
    $\mathbf{h}_{vu} = \mathrm{COMBINE}\left(\mathbf{h}_v^{(L)}, \mathbf{h}_u^{(L)}\right)$
    $\hat{y}_{v,u} = \mathrm{PREDICTOR}\left(\mathbf{h}_{vu}\right)$
    Calculate loss$(y_{v,u}, \hat{y}_{v,u})$
    Update $\mathbf{E}$, model weights, etc.
end for
Herein $\mathbf{h}_{vu}$ is the link representation for (v,u). COMBINE$(\cdot,\cdot)$ can be Hadamard production, concatenation, etc. PREDICTOR$(\cdot)$ is a predictor like MLP.

[150], Sort pooling [169], etc. These combination operations on node-wise representations learned by GNNs are unlikely to contain the behavior of extracting NCN-dependent structural information.

□

GNNs might learn little structural information related to the number of common neighbors. However, the neighborhood aggregation algorithm of GNNs learns node-wise representations by passing the messages of neighboring nodes of each node to that node and set-based aggregates them [47, 136, 150, 170]. Such set-based aggregation operation inherently washes out the information related to the number of nodes in the set, including the number of common nodes between two nodes. As shown in Fig. 4.1, the node 1 and 4 have common neighbors $5,6$. In GNN learning, the node 1 will receive the messages from $2,3,5,6$, where the number of common neighbors (i.e., $\mathrm{CN}_{1,4} = 2$) can hardly be captured in the aggregation of five representations (i.e., representations of nodes $1,2,3,4,5$). Note that in this example, attention-based aggregation also cannot effectively learn $\mathrm{CN}_{1,4} = 2$ from the aggregation of five representations. The reason is the same as the proof of Proposition 1. In fact, it is difficult to interpret what information the GNN has learned in a rigorous mathematical format. Nevertheless, we can say with certainty that the ability of GNNs in learning NCN-dependent structural information is quite weak.

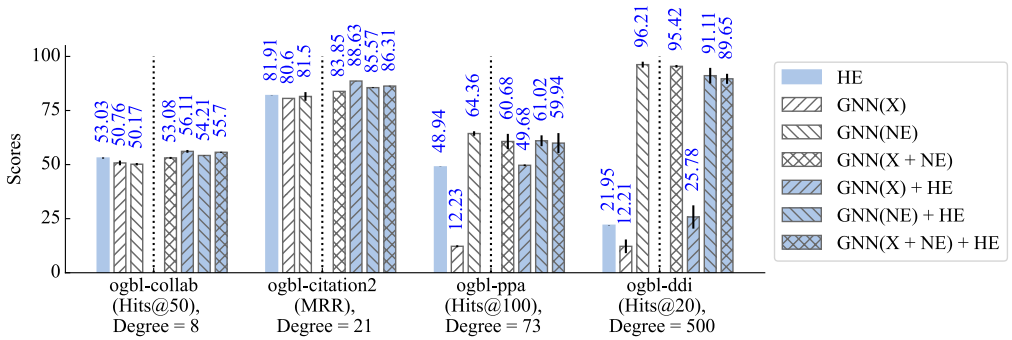


Figure 4.2 The results of Algorithm 3 using heuristic encoding (HE) only, node features (X) only, node embeddings (NE) only, or their combinations. The nodes in ogbl-ddi do not have any features and we use the node degree as the node feature.

4.3.2 Empirical Study

Experimental Design

In this work, we conduct our empirical study based on a commonly-used link prediction algorithm, i.e., Algorithm 3. As shown, given two nodes v, u, the link prediction is performed by combining the two nodes' representations from the last GNN layer into a pair-specific link representation $\mathbf{h}_{vu}$ and then passing it into a predictor like MLP. During the training stage, the node pair (v,u) can be a positive or negative link sample, where a negative sample can be two distant nodes that are not connected to each other.

If Proposition 2 holds, we expect that properly integrating NCN-dependent heuristics into Algorithm 3 could improve the link prediction performance. To this end, we modify the equation in Line 10 of Algorithm 3 to:

$$\begin{aligned} \mathbf{e}_{vu} &= \text{CONCAT}\left(\mathbf{e}_{vu}^{(\text{CN})}, \mathbf{e}_{vu}^{(\text{JA})}, \cdots, \mathbf{e}_{vu}^{(\text{RA})}\right), \\ \hat{y}_{vu} &= \text{PREDICTOR}\left(\text{CONCAT}\left(\mathbf{h}_{vu}, \mathbf{e}_{vu}\right)\right), \end{aligned} \tag{4.3}$$

where $\mathbf{e}_{vu}^{(\text{CN})}, \mathbf{e}_{vu}^{(\text{JA})}, \mathbf{e}_{vu}^{(\text{RA})}$ are trainable embeddings by encoding CN, JA, and RA, respectively. For the purpose of verifying Proposition 2, we only encode NCN-dependent link heuristics. The methodology of heuristic encoding is as follows. For heuristics that are discrete integer values (e.g., CN, SPD), we assign a trainable embedding vector to each integer. In the case of heuristics that are continuous floating-point values (e.g., AA, RA), we partition the value range into small bins and subsequently allocate each bin a unique embedding vector.

Encoding heuristics into embeddings is mainly because if we directly use heuristics as features, we find that the model optimization is challenging, where the model is more

likely to get stuck in a local optimum. This issue could arise due to the high correlation between the heuristic features and the link samples. Encoding heuristics into trainable embeddings can address this challenge successfully.

4.3.3 Experimental Settings

Datasets

We conduct all our experiments on four recently-published OGB link prediction datasets: *ogbl-collab, ogbl-citation2, ogbl-ppa*, and *ogbl-ddi* [54]. All these datasets are constructed based on real-world data, covering diverse realistic applications and spanning different scales (4K - 3M nodes). We do not use previous commonly-used datasets [33] like *Cora, Citeseer, Pubmed*, etc. This is because these datasets usually suffer from a series of issues such as unrealistic and arbitrary data splits, small scale, and data leakage [122, 33]. Especially, Shchur et al. [122] show that different data splits on such datasets lead to inconsistent results in performance evaluation among modern GNN methods, thus rendering them inadequate for meeting our research objectives.

Evaluation Metrics

OGB provides official evaluation protocol [54]. We follow it in the data splits and evaluation metrics (i.e., Hits@50, MRR, Hits@100, and Hits@20 on ogbl-collab, ogbl-citation2, ogbl-ppa and ogbl-ddi, respectively). We report the result on the test set, with mean and standard deviation computed across 5 trials. Additional experimental settings can be found in the Appendix. Code is available at *https://github.com/2023researchup/GNNHE*.

Experimental Results

Fig. 4.2 shows the experimental results, where HE is the model (i.e., Algorithm 3) that only uses heuristics encoding $\mathbf{e}_{vu}$ in Eq. 4.3 (herein we show the best HE, and more HE results are shown in Fig. 4.6 in the Appendix). GNN(X), GNN(NE), and GNN(X+NE) are the models only using the GNN with three different settings of input, i.e., node features (X) only, node embeddings (NE) only, and concatenation of both X and NE (X+NE), respectively. The model GNN(X)+HE uses both $\mathbf{h}_{vu}$ and $\mathbf{e}_{vu}$. For the GNN model, we use GAT [136] with high-hop modification (refer to the Appendix for details) on ogbl-ddi and GCN [64] on the other three datasets.

First, we can see in Fig. 4.2 that HE outperforms GNN(X) on all datasets, suggesting that NCN-dependent heuristics convey meaningful information which could not be effectively learned by the GNN. Second, as we expected, most of the results of combining GNN and HE are better than those only using GNN. Especially, GNN(X)+HE achieves the best on ogbl-collab and ogbl-ciation2. All these results can support Proposition 2.

4.4 Node Embedding in Link Prediction

In Algorithm 3, $\mathbf{h}_i^{(0)}$ can be initialized using feature vector $\mathbf{x}_i \in \mathbf{X}$, node embedding $\mathbf{e}_i \in \mathbf{E}$ or the concatenation of both $\mathbf{x}_i$ and $\mathbf{e}_i$. Some papers may refer to node embedding as an intermediate representation of a node in GNNs. In this work, we clearly distinguish node embeddings from node representations. We consider node embedding as a type of node-wise input feature. The embedding of a node can be viewed as encoding a unique node id into a trainable embedding vector, which is like encoding a unique word id into the word embedding in natural language processing [95]. Note that we can encode any feature into a trainable embedding vector (e.g., encoding node degree to an embedding). The main difference between node embeddings and the embeddings of other node features is that each node embedding vector is unique to that node and can be dynamically trained during the training process, while other node features cannot satisfy both criteria. For example, a node id is unique to that node but cannot be trained; an embedding of a node degree can be trained but is not unique to that node since different nodes could have the same degree.

4.4.1 Experimental Observations

As shown in Fig. 4.2, on two relatively sparse graphs, i.e., ogbl-collab and ogbl-citation2 with graph degrees of 8 and 21, respectively, the performance of GNN(NE) is on par with that of GNN(X), and GNN(X+NE) performs best. By comparison, on two denser graphs, i.e., ogbl-ppa and ogbl-ddi with higher graph degrees of 73 and 500, respectively, GNN(NE) outperforms GNN(X) by a large margin. These results indicate that incorporating node embeddings can enhance the link prediction performance of Algorithm 3. More notably, these observations reveal a strong positive correlation between the performance improvement by node embeddings and the graph degree, whereby denser graphs exhibit greater improvement.

4.4.2 Analytical Insights into Node Embeddings

Proposition 3. *Like Algorithm 3, when the training is supervised by positive and negative link samples, trainable node embeddings can enhance the expressive power of the GNN-based link prediction model.*

Proof. In Algorithm 3, the parameters optimized by the link samples could include model weights, trainable node embeddings, and other feature embedding weights (e.g. node degree embeddings). The key difference between node embedding weights and other learnable weights is that the former is unique to each node but the latter is shared across multiple nodes (e.g., the GNN weight matrix $\mathbf{W}^{(l)}$ in Eq. 4.1 is shared across all nodes; a node degree embedding would be shared by several nodes). The *unique* nature of node

embeddings can bring benefits. As shown in Algorithm 3, when the model training is supervised by a link sample (v,u), for a GNN using node embeddings, the loss calculated based on $(y_{v,u},\hat{y}_{v,u})$ would be used to optimize the node embeddings of nodes v,u and their neighboring nodes (i.e., the nodes involved in calculating $\mathbf{h}_v^{(L)},\mathbf{h}_u^{(L)}$). The link state of (v,u) could be encoded into the node embeddings of these nodes, which would enable the node embedding better represent the corresponding node, similar to the training of word embedding [95]. After being trained with sufficient positive and negative link samples, the node embedding of each node could know which nodes (through their node embeddings) in the graph are more likely to be or not to be connected to that node.

If node embeddings are not used in Algorithm 3, link samples will only supervise the optimization of the model weights that are shared across multiple nodes. The states of link samples could not be effectively retained by the model since these shared weights might learn a common pattern for different nodes rather than unique to a node. By comparison, each node embedding is unique to that node and could learn the link information specific to that node. In this respect, trainable node embeddings could enhance the expressive power of the GNN-based link prediction model.

In Proposition 3, the requirement of *the model training is supervised by link samples* is indispensable. Without this prerequisite, the link state between two nodes could not be encoded into node embeddings. Additionally, negative link samples can allow the embeddings of two distant nodes and their neighbors to see each other during the optimization of the GNN-based model. □

Proposition 3 shows that trainable node embeddings could improve the expressiveness of GNN-based link prediction models. However, this does not mean that the model using node embeddings will certainly perform better than that only using node features, especially in practical applications where careful feature engineering based on domain knowledge is conducted. Besides, trainable node embeddings remain limitations in the inductive setting [131]. For example, when new nodes are added to the graph, the model together with all node embeddings may need to be retrained.

Finding 1. *Following Proposition 3, the denser the graph, the more the enhancement by node embeddings.*

In GNN-based link prediction models like Algorithm 3, node embeddings in a dense graph could be better learned for link prediction than those in a sparse graph. The explanation could be as follows. In a dense graph, a node often has a lot of neighboring nodes, thereby providing numerous opportunities for that node to meet other nodes and encode link relationships of that node with these other nodes into its embedding during the neighborhood aggregation process in GNN training. In contrast, a sparse graph typically contains only a limited number of neighbors for each node. For example, in the case where

a node v has only one neighboring node w, the optimization of the embedding of node v in a GNN would mainly rely on the neighbor w. As a result, the learned embedding of node v would lack sufficient information to identify the relationships between node v and the majority of the other nodes in the sparse graph because node v rarely or never sees them during the training process.

Finding 2. *The learning styles of SimRank, as outlined in Algorithm 2, and the GNN-based link prediction model with node embeddings, as described in Algorithm 3, exhibit certain similarities.*[3]

Comparing Algorithms 2 and 3, several similarities emerge. Firstly, similarity scores in Line 3 of Algorithm 2 and node embeddings in Line 3 of Algorithm 3 both need to be initialized and can be dynamically trained. Secondly, the updating computations of both algorithms (i.e., Line 5 of Algorithm 2 and Line 6 of Algorithm 3) involve neighboring nodes. Moreover, both the learned results (i.e., s_{ij} in Algorithm 2 and $\hat{y}_{v,u}$ in Algorithm 3) describe the existence likelihood of a link between two nodes. However, compared to Algorithm 3 where the trainable parameters include node embeddings, model weights, etc., the expressive power of SimRank is limited, where only the similarity scores between every two nodes can be optimized, with each score always taking the form of a scalar.

Finding 2 implies that although NCN-dependent structural information cannot be effectively learned via GNNs (Proposition 2), other types of pair-specific structural information (e.g., the information captured by SimRank) might be learned through GNNs.

4.5 Limitation Analysis of Existing Methods

In this section, we first present a brief survey of existing link prediction methods and then demonstrate the effectiveness of our findings in identifying the limitations of these methods.

4.5.1 A Survey of Link Prediction Methods

Heuristic Methods.

Traditional link heuristics such as CN, AA [2], Katz index [62], etc. are usually defined based on the number of common neighbors or paths between two nodes [92]. Their effectiveness in link prediction has been confirmed in real-world tasks [82, 92, 66]. However, many link heuristics are designed for specific graph applications and their performance

[3]For Finding 2, we do not compare the performance of Algorithm 2 and Algorithm 3 due to the difficulty of SimRank in computation. For example, the basic memory requirement of SimRank is 415G and 2.42T on ogbl-collab and ogbl-ppa, respectively. Besides, SimRank produces 0 at Hits@20 on ogbl-ddi.

may vary on different graphs [66]. Also, the expressiveness of these methods is limited compared to graph representation learning [168].

Node Embeddings with Matrix Factorization

A family of node embedding methods is those built with the utilization of matrix factorization [65] on the graph adjacency matrix. Among them, MF [93] is a pioneer work extending matrix factorization for link prediction. FSSDNMF [20] proposes a link prediction model based on non-negative matrix factorization. In general, such methods mainly rely on the adjacency matrix and tend to encounter scalability issues when employed on large graphs.

Node Embedding based on Relative Distance Encoding.

Another family of node embedding methods is those based on relative distance encoding. The similarity of nodes in the embedding space reflects the semantic similarity of nodes in the graph [104]. Such methods would learn more similar embeddings for two close nodes than for two nodes that are far apart. Following word embedding [95], methods such as Deepwalk [104], Node2vec [43], UniNet [153] learn node embeddings by treating the nodes as words and treating the sequences of nodes generated based on links as sentences. Inspired by subword tokenization [117], NodePiece [37] explores parameter-efficient node embeddings. However, these methods typically train node embeddings in an unsupervised learning manner without link samples. Solely using such node embeddings for link prediction empirically perform poorly compared to other methods [142].

Graph Neural Networks.

A number of GNN models have been proposed. GCN [64] is a graph convolutional network that learns node representations by summing the normalized representations from the first-order neighbors. GraphSAGE [47] samples and aggregates representations from local neighborhoods. GAT [136] introduces an attention-based GNN architecture. JKNet [151] adds a pooling layer following the last GNN layer and each GNN layer has a residual connection to this layer. Cluster-GCN [25] proposes an efficient algorithm for training deep GCN on large graphs. LRGA [106] incorporates a Low-Rank global attention module to GNNs. Several works such as Mixhop [1], AGDN [128], DEGNN [77] propose techniques to leverage high-hop neighbors. ID-GNN [158] embeds each node by inductively taking into account its identity during message passing. These GNNs have demonstrated promising link prediction performance.

SEAL-type Methods.

SEAL and its subsequent works [168, 77, 131, 154] address the link prediction problem by classifying the subgraphs that are extracted specifically for candidate links. SEAL [168] extracts a local enclosing subgraph for each candidate link and uses a GNN [169] to classify these subgraphs for link prediction. GraiL [131] is developed for inductive link prediction. It is similar to SEAL but it replaces SortPooling [169] with MEAN-pooling. DEGNN [77] proposes a distance encoding GNN. For link prediction, DEGNN uses a SEAL-type strategy but a different labeling technique [170]. Differently from SEAL, Cai et al. [16] transform the enclosing subgraph into a corresponding line graph and address the link prediction task with the node classification problem in its corresponding line graph. Pan et al.[99] follow the enclosing subgraph strategy in SEAL while designing a new pooling mechanism called WalkPool. SUREL [154] proposes an algorithmic technique to improve the computational efficiency of subgraph generation in SEAL. SIEG [3] incorporates the structural information learned from the enclosing subgraphs into the GNN model for link prediction.

Other Methods Developed Specifically for Link Prediction

In addition to SEAL-type methods, various link prediction-specific methods have been proposed. Wang et al. [142] present PLNLP by jointly using the representations learned by a GNN, distance encoding, etc. Yun et al. [161] present Neo-GNN, which weighted aggregates the link prediction scores obtained by heuristics and a GNN. Zhu et al. [177] propose NBFNet that generalizes traditional path-based link heuristics into a path formulation. Singh et al. [124] demonstrate that simply adding a set of edges to the graph as a pre-processing step can improve the performance of link prediction models. Roy et al. [112] propose PermGNN where the neighborhood aggregator is optimized directly by link samples. Zhao et al. [171] study counterfactual questions about link existence by causal inference. Wu et al. [145] propose a RelpNet, which aggregates edge features along the structural interactions between two target nodes. Guo et al. [44] propose cross-model distillation techniques for link prediction. Shang et al. [120] propose a negative link sampling method PbTRM based on a policy-based training method.

4.5.2 Limitation Analysis

In this study, we provide two critical insights into the application of GNNs in link prediction. Firstly, we show that aggregation-based GNNs inherently lack the ability to learn NCN-dependent structural information for link prediction. Secondly, we demonstrate through experimentation that node embeddings can boost the performance of GNN-based link prediction models on dense graphs. These insights can serve as effective avenues to identify

and interpret the limitations of existing link prediction methods. Firstly, we present two case studies to illustrate this.

Case study 1. Can SEAL effectively learn NCN-dependent structural information as it expects? SEAL-type methods have achieved the best performance on several link prediction datasets [168, 77, 131, 154]. SEAL [168] has proven that most link heuristics between two nodes can be computed approximately within an enclosing subgraph extracted specifically for that two nodes. Most SEAL-type methods employ GNNs for representation learning, with the expectation that from such enclosing subgraphs, the GNN can learn the structural information equivalent to link heuristics including CN, AA, Katz, etc. However, whether this expectation holds true has not been thoroughly investigated in existing works. Our insights gained from this work offer novel perspectives to investigate this issue, which suggests that a definitive affirmation of this expectation is unlikely to be reached.

First, the GNNs used in SEAL-type methods, e.g., DGCNN [169] in SEAL [168], R-GCN [114] in GraiL [131], still belong to the type of aggregation-based GNNs. According to Proposition 2, these GNNs in SEAL-type methods inherently cannot effectively learn NCN-dependent structural information.

Furthermore, we have noticed that SEAL-type methods usually use a node labeling technique [170] to add labeling features to each node in an enclosing subgraph extracted specifically for a candidate link. The labeling features of each node describe the relationship of that node to the target two nodes. Fig. 4.3 illustrates such a labeling method, where the labeling features of a node are the shortest path distances from the node to the target pair of nodes. Zhang et al. [170] point out that the labeling features can help the GNN learn the structural information related to the number of common neighbors. Their explanation is as follows. As shown in Fig. 4.3, for node v and u, in the first iteration of the neighborhood aggregation in a GNN, only the common neighbors between node v and u will receive the labeling messages from both v and u; then in the second iteration, the common neighbors will pass such messages back to both v and u, which can encode the number of common neighbors into the representations of node v and u.

However, we question this statement. In the second iteration in the explanation above, apart from the common neighbors, the non-common neighbors of node v also pass their messages back to v. The messages from all neighbors of v are then aggregated through a set-based pooling (e.g., MEAN or attention-based pooling as shown in Eq. 4.2). Such an aggregated result for node v would wash out the distinguishable labeling information. We present an example to illustrate this. As shown in Fig. 4.3, if the pooling method in a GNN is MEAN, then the aggregation of the labeling features of the neighbors of node v would be equal to that of node w, i.e., $\text{MEAN}(\{10,3,2,1\}) = \text{MEAN}(\{4,6,4,3,3\})$. This means that the distinct labeling features of the neighbors of a node are not effectively kept in the aggregated result. In other words, the aggregated results for node v and w in the positive and negative link samples become indistinguishable. Note that attention-based pooling in

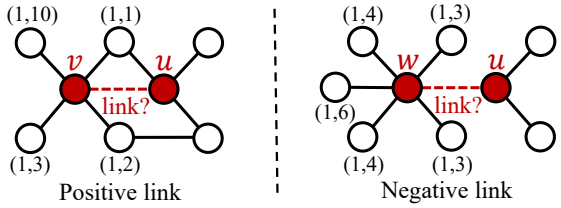


Figure 4.3 Node labeling in SEAL-type methods. The left is a subgraph specific for a positive link sample and the right is a negative one. The labeling features are based on the SPDs from every node (here only show the first-order neighbors of node v or w) to the target pair of nodes. For example, on the left, the node with the labeling $(1, 10)$ indicates that the SPD from this node to node v and u is 1 and 10, respectively.

GNNs like GAT [136] also suffers from the above limitation for the same reason as the proof of Proposition 1. The same goes for node u. It should be noted that our example is merely for illustrative purposes. In practice, a GNN layer contains a series of complicated operations such as linear and non-linear transformations, dropout, residual connection, and others. The structural information in the labeling features could be partially kept in the learned representations.

Although SEAL-type methods cannot effectively learn structural information related to the number of common neighbors, we would highlight that these methods are powerful for link prediction. Such methods transform the pair-specific link prediction problem into a graph-level classification task. Compared to the models like Algorithm 3 that only combines the representations of two target nodes, SEAL-type methods take advantage of the representations of not only two target nodes but also their neighboring nodes in the enclosing subgraph, enabling the model to consider more information of the surrounding environment of the candidate link.

Case study 2. NBFNet lacks the algorithmic ability to leverage node embeddings. NBFNet [177] is a model specifically developed for link prediction. Differently from GNN-based link prediction methods like Algorithm 3 and SEAL-type methods, NBFNet generalizes traditional link heuristics such as Katz index [62], Personalized PageRank [73] into a general formulation and approximates such formulation using a special network. Unlike aggregation-based GNNs that propagate and aggregate node-wise representations, NBFNet is designed to train edge-wise representations while hardly considering node-wise information including node attributes and node embeddings. This would make NBFNet lack the algorithmic ability to train powerful node embeddings and may lead to non-competitive link prediction performance on dense graphs.

It should be noted that the primary objective of our research is to present the insights we have gained, and showcase their potential to aid in identifying the issues of existing link prediction methods. Our focus does not lie in developing solutions to these issues, as

this goes beyond the scope of our main goal. Nevertheless, these identified issues could pave the way for future research.

4.5.3 Further Analysis of Experimental Results

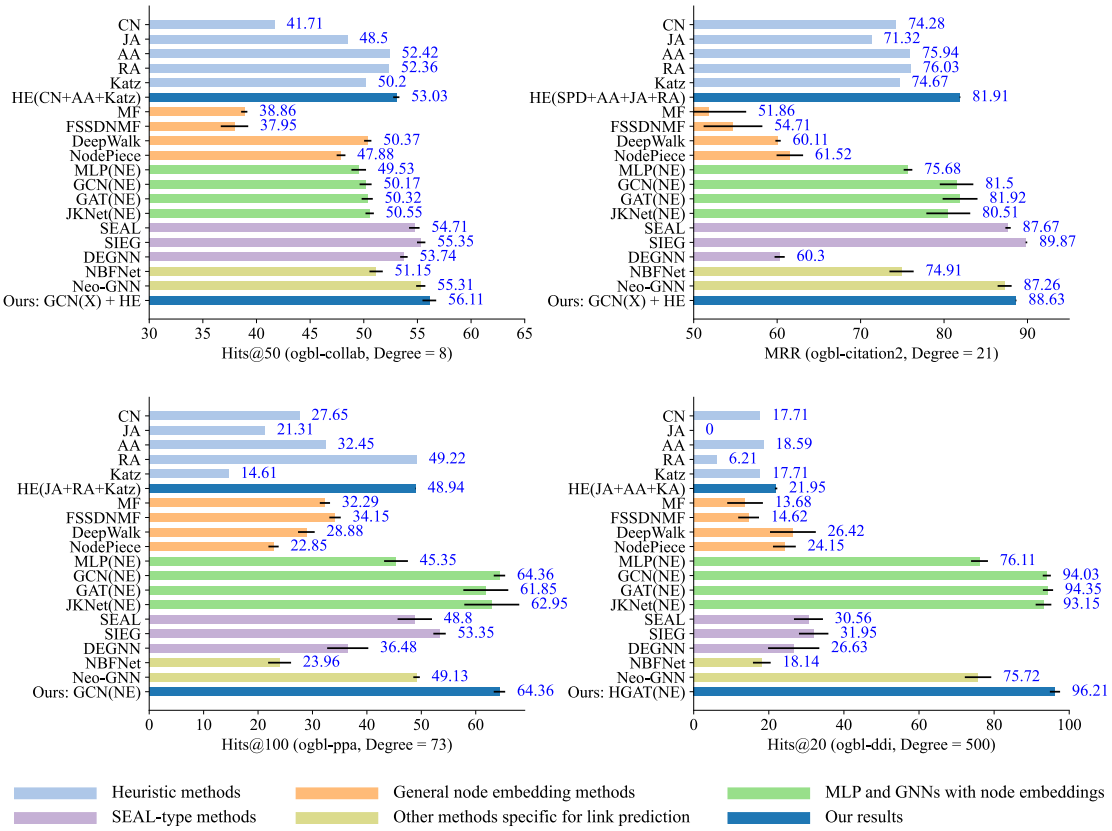


Figure 4.4 Results of different methods for link prediction on four OGB datasets. For MLP and general GNNs, we present their results obtained by utilizing node embeddings, considering the dominant performance of node embeddings as shown in Fig. 4.2.

We expand our limitation analysis of existing link prediction methods by examining their experimental results on four OGB benchmark datasets. In the interest of brevity, we focus our investigation on several mainstream types of methods, as presented in Fig. 4.4.

First, the performance of each heuristic method is not stable across the four datasets. For example, RA performs best on ogbl-ppa but second worst on ogbl-ddi. These results are consistent with the research of [66], which has demonstrated that many link heuristics are designed for specific applications and may perform well only on those applications. Moreover, the unstable performance of every single heuristic method confirms the need of combining multiple heuristics in modern link prediction methods, such as the Algorithm 3 where various heuristics are encoded in Eq. 4.3.

We also report the heuristic encoding results (i.e., HE) obtained through Algorithm 3 when only employing heuristic encoding. As shown in Fig. 4.6 in the Appendix, we find that it is not always true that the more the heuristics used, the better the performance of HE. In contrast, encoding a certain number of heuristics can yield the best performance, whereas encoding too many heuristics would pose an optimization challenge.

In Fig. 4.4, the node embedding methods based on relative distance encoding (DeepWalk [104], NodePiece [37]) exhibit slightly better performance than those based on matrix factorization (MF [93], FSSDNMF [20]). Nevertheless, all these methods fall short compared to other types of methods. This could be attributed to the limitations of such methods, such as reliance solely on the adjacency matrix or unsupervised learning without link samples. This also underscores the critical role of link samples in supervising the learning of node embeddings for link prediction as proposed in our Proposition 3.

Fig. 4.4 also presents the results of MLP and general GNNs (GCN [64], GAT [136] and JKNet [151]) that use node embeddings only. MLP(NE) performs much worse than GNNs, demonstrating the significance of neighborhood information aggregation of GNNs in training node embeddings, considering that MLP updates each node's representation independently of other nodes. Furthermore, GCN(NE) and GAT(NE) perform comparably, indicating that the expressiveness of GCN is sufficient for learning node embedding. The similar performance of GCN and GAT empirically supports our proofs in Proposition 1 and 2, where we point out that the attention mechanism (e.g., GAT) cannot address the inherent issue of GNNs in learning information related to the number of each node's neighbors and of common neighbors between two nodes.

In Fig. 4.4, SEAL-type methods show state-of-the-art performance. Especially, SIEG [3] achieves the best results on two sparse graphs, i.e., ogbl-collab and ogbl-citation2 with graph degrees of 8 and 21, respectively. However, they perform worse than general GNNs with node embeddings (GCN(NE) [64], GAT(NE) [136] and JKNet(NE) [151]) on two dense graphs, i.e., ogbl-ppa and ogbl-ddi. This discrepancy in the performance of SEAL-type methods could be attributed to the algorithmic challenge of training node embeddings using subgraphs. Unlike general GNNs, the algorithm of SEAL-type methods limits each node to perceive other nodes within the subgraph rather than the entire graph, thereby restricting the information flow between nodes and potentially reducing the effectiveness of the learned node embeddings.

Besides, Fig. 4.4 shows two link prediction-specific methods, namely NBFNet [177] and Neo-GNN [161]. NBFNet underperforms on four datasets, which aligns with the limitations identified in Section 4.5.2. Neo-GNN predicts link likelihood by combining the scores obtained by heuristic methods and the result produced by a GNN. It performs on par with the state-of-the-art SEAL-type methods on two sparse graphs (ogbl-collab and ogbl-citation2).

By comparing the performance of different methods on four datasets, it is evident that NCN-dependent information plays an important role in link prediction on sparse graphs. Methods capable of acquiring NCN-dependent information (e.g., SEAL-type methods and Neo-GNN) generally outperform those that cannot. Moreover, our GNN(X)+HE based on Algorithm 3 performs better than SEAL-type methods on ogbl-collab, supporting our limitation analysis of SEAL-type methods in Section 4.5.2, i.e., such methods may not effectively learn the information equivalent to NCN-dependent heuristics. In addition, for link prediction on dense graphs, the contribution of node embeddings becomes dominant. Simple GNNs like GCN[47] with the incorporation of node embeddings can surpass the performance of most existing methods.

4.6 More on the Experiments

4.6.1 GNN Models Used in the Experiments

In our experiments, we use two general GNN models, i.e., GCN [47] and GAT [136]. We also employ the high-hop technique [87, 77] in our algorithm to improve the expressive power of the GNN model.

Both GCN and GAT follow the neighborhood information aggregation scheme. GCN uses MEAN-pooling in the aggregation. It can be formalized as

$$\mathbf{H}^{(l)} = \hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}, \tag{4.4}$$

where $\hat{\mathbf{A}}$ is the normalized version of the graph adjacency matrix $\mathbf{A}$, where $\mathbf{A}$ has self-connections. There are two normalization methods, namely $\hat{\mathbf{A}} = \mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}$ or $\hat{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$. $\mathbf{D}$ is the degree matrix which is a diagonal matrix where $d_{i,i} = \sum_{j\in\mathbb{V}} a_{i,j}$.

GAT [136] is an attention-based GNN. It can be formalized using Equation 4.2, where the attention coefficient $a_{i,j}^{(l)}$ is computed by

$$\alpha_{i,j}^{(l)} = \exp\left(\text{LeakyReLU}\left(\mathbf{a}^{(l)} \cdot \text{CONCAT}\left(\mathbf{h}_i^{(l-1)}, \mathbf{h}_j^{(l-1)}\right)\right)\right), \tag{4.5}$$

where $\mathbf{a}^{(l)} \in \mathbb{R}^{2d}$ is a trainable weight vector in the l-th layer, $\mathbf{h}_i^{(l-1)} \in \mathbb{R}^d$ and $\mathbf{h}_i^{(l-1)} \in \mathbb{R}^d$ are the representations of node i and j from the $(l-1)$-th layer, respectively. $\text{CONCAT}(\cdot,\cdot)$is the concatenation operation. $\text{LeakyReLU}(\cdot)$ is an activation function.

In Equation 4.2, the calculation is only performed when the node i and j are directly linked, i.e., when $a_{i,j} = 1$ in the i,j-th entry of $\mathbf{A}$. Several works [87, 77] have proposed high-hop aggregation techniques, with the goal of enhancing the expressive power of GNNs. We employ such a technique by following the work of [87] to modify the adjacency matrix into a high-hop adjacency matrix. Specifically, the i,j-th entry of h-hop adjacency

Table 4.2 Results on test sets of OGB link prediction datasets. Higher is better. For a fair comparison, we only list the results that do not take advantage of the validation dataset as training data.

	ogbl-ddi Hits@20 (%)	**ogbl-collab** Hits@50 (%)	**ogbl-ppa** Hits@100 (%)	**ogbl-citation2** MRR (%)
MF [93]	13.68 ± 4.75	38.86 ± 0.29	32.29 ± 0.94	51.86 ± 4.43
Grarep [17]	12.99 ± 3.54	39.63 ± 0.44	31.36 ± 1.32	51.25 ± 2.75
DeepWalk [104]	26.42 ± 6.10	50.37 ± 0.34	23.02 ± 1.63	61.05 ± 2.33
Node2vec [43]	23.26 ± 2.09	48.88 ± 0.54	22.26 ± 0.83	61.41 ± 0.11
NodePiece [37]	24.15 ± 3.04	47.88 ± 0.41	22.85 ± 0.94	61.52 ± 1.59
GraphSAGE [47]	83.90 ± 4.74	48.10 ± 0.81	16.55 ± 2.40	82.60 ± 0.36
Mixhop [1]	93.29 ± 2.43	52.09 ± 0.57	52.04 ± 3.37	83.26 ± 0.73
AGDN [128]	95.38 ± 0.94	52.26 ± 0.85	51.33 ± 2.16	83.17 ± 0.54
NGNN [127]	83.86 ± 2.25	52.04 ± 0.83	50.75 ± 1.94	84.92 ± 0.25
ID-GNN [158]	85.13 ± 3.46	51.74 ± 0.46	49.45 ± 1.62	83.31 ± 0.64
VAGNN (Chapter 3)	93.13 ± 3.96	52.74 ± 0.55	50.45 ± 1.32	84.31 ± 0.37
SEAL [168]	30.56 ± 3.86	54.71 ± 0.49	48.80 ± 3.16	87.67 ± 0.32
GraiL [131]	31.76 ± 4.24	54.19 ± 0.52	47.25 ± 2.84	86.59 ± 0.58
DEGNN [77]	26.63 ± 6.82	53.74 ± 0.35	36.48 ± 3.78	60.30 ± 0.61
SUREL [154]	32.31 ± 4.15	54.37 ± 0.46	53.23 ± 1.03	88.83 ± 0.18
LGLP [16]	28.18 ± 3.34	51.59 ± 0.93	49.36 ± 1.95	84.65 ± 0.27
LRGA [106]	62.30 ± 9.12	52.21 ± 0.72	26.12 ± 2.35	66.49 ± 1.59
PLNLP [142]	90.88 ± 3.13	52.92 ± 0.98	32.38 ± 2.58	84.92 ± 0.29
PermGNN [112]	88.49 ± 2.57	51.04 ± 0.59	51.51 ± 1.08	82.54 ± 0.29
RelpNet [145]	48.60 ± 2.37	49.18 ± 0.93	47.08 ± 1.34	79.92 ± 0.15
CFLP [171]	76.12 ± 2.84	46.66 ± 0.72	45.81 ± 1.04	82.59 ± 0.37
PbTRM [120]	75.27 ± 2.95	50.28 ± 1.08	42.57 ± 2.26	80.05 ± 0.47
LLP [44]	46.92 ± 2.36	52.35 ± 0.91	44.03 ± 1.83	81.53 ± 0.73

matrix satisfies $a_{i,j} = 1$ if $\text{SPD}(i, j) \leq h$, otherwise 0, where $\text{SPD}(\cdot)$ computes the shortest path distance between nodes i and j.

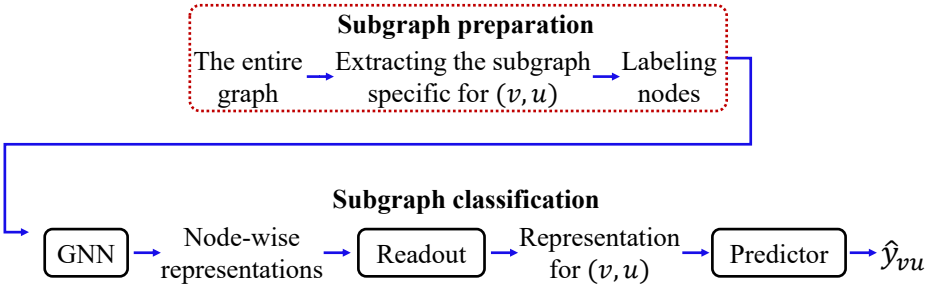


Figure 4.5 The algorithmic flow chart of SEAL-type methods.

4.6.2 SEAL-type Methods.

SEAL and its successors [168, 77, 131, 154] address the link prediction problem by classifying the subgraphs that are extracted specifically for candidate links.

We briefly describe the algorithm flow of SEAL-type methods. As shown in Figure 4.5, given an entire graph $\mathcal{G}$ and a target node pair (v,u), a h-hop enclosing subgraph $\mathcal{G}_{vu}^{(h)}$ with a set of nodes $\mathbb{V}_{vu}^{(h)} = \{v' \mid \text{Spd}(v',v) \leq h \text{ or } \text{Spd}(v',u) \leq h\}$ is extracted from $\mathcal{G}$, where $\text{Spd}(\cdot,\cdot)$ calculates the shortest path distance between two nodes. Then, for each node in $\mathcal{G}_{vu}^{(h)}$, a node labeling method [170] is used to assign a labeling vector to that node as its additional features according to its relationship to (v,u). At the modeling stage, $\mathcal{G}_{vu}^{(h)}$ is fed into a GNN model, and the node-wise representations of the nodes in $\mathbb{V}_{vu}^{(h)}$ are learned. Following the last layer of the GNN, a readout function $\text{READOUT}(\cdot)$ is employed over the learned node-wise representations of all nodes in $\mathbb{V}_{vu}^{(h)}$, and then a representation for $\mathcal{G}_{vu}^{(h)}$ is produced. At last, a predictor $\text{PREDICTOR}(\cdot)$ takes this representation as input to perform link prediction for the node pair (v,u). Formally, the existence likelihood of a link between v,u predicted by SEAL-type methods is

$$\hat{y}_{vu} = \text{PREDICTOR}\left(\text{READOUT}\left(\left\{\mathbf{h}_{v'}^{(L)} \mid \forall v' \in \mathbb{V}_{v,u}^{(h)}\right\}\right)\right). \tag{4.6}$$

The $\text{READOUT}(\cdot)$ (e.g., SortPooling in SEAL [168]) is typically used for graph-level classification, aiming to deal with size differences among graphs.

4.6.3 Implementation Details

We implement our algorithm based on PyTorch and PyTorch Geometric [36]. The model is trained with Adam optimizer [63]. The learning rate is decayed using the ExponentialLR method [79]. We conduct all experiments on ogbl-ddi and ogbl-collab on a Linux machine with 14-core CPU, 192G RAM, and NVIDIA Quadro P6000 (24G), and on ogbl-ppa and ogbl-citation2 on a machine with 32-cores CPU, 512G RAM and NVIDIA A100 (40G).

4.6.4 Additional Experimental Results

We reproduce the results of several existing link prediction methods and we show them in Table 4.2. Also, the results of encoding different heuristics are presented in Fig. 4.6.

4.7 Summary and Implication

This work studies the link prediction capability of GNNs. We show the limitations of GNNs in learning NCN-dependent structural information and the strengths of node embeddings in dense graphs. We also demonstrate that these insights can effectively identify the

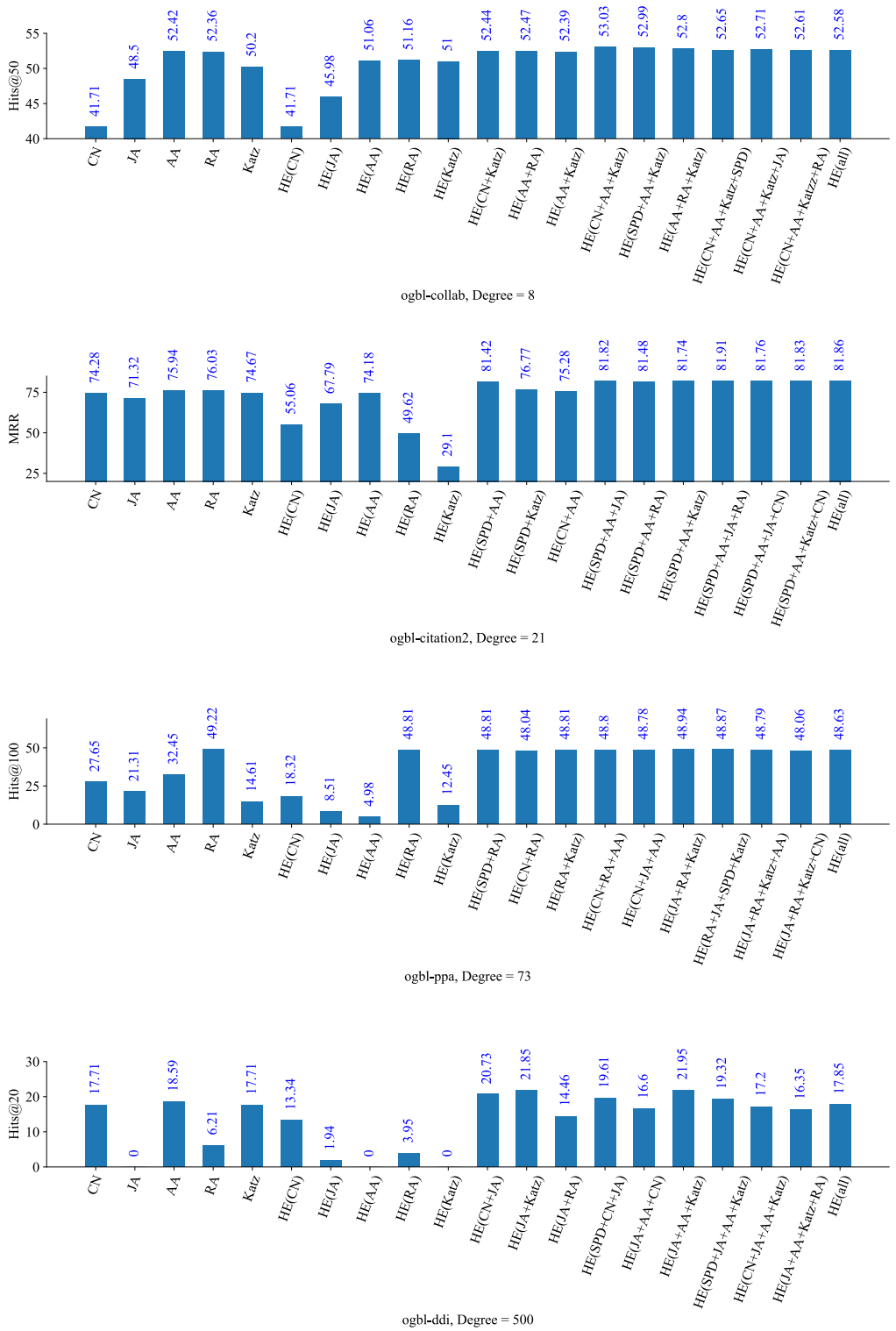


Figure 4.6 Results of encoding different numbers of link heuristics. We can find that the best result on each dataset is achieved by encoding a certain number of heuristics rather than all heuristics.

bottlenecks of existing link prediction methods, which could guide the search for more effective algorithms.

This study carries significant implications for practical link prediction. For sparse graphs, either SEAL-type methods or GNNs with heuristic encoding can yield satisfactory performance. For dense graphs, the GNN with node embeddings is an ideal choice in the transductive setting. In inductive learning, the methods that do not involve the training of node embeddings may be more suitable.

Chapter 5

Exploring the Functionalities of Diverse GNN Techniques in Real-World Scenarios

5.1 Introduction

Numerous techniques have been developed with the purpose of enhancing the expressiveness of GNNs. These advanced techniques include attention mechanisms, high-hop neighborhood construction, residual connections, and various others. Despite notable advancements in GNNs, certain fundamental questions remain unresolved regarding the functionality as well as the effectiveness of these techniques in effectively addressing challenges associated with the real-world implementation of GNNs.

In this chapter, we study these questions based on a real-world application: pipe failure prediction for pipe networks. Several characteristics of the pipe networks make them ideal for validating and exploring various GNN techniques. Firstly, the pipes possess a set of attributes, such as length, material, and diameter. Secondly, the pipes are accompanied by temporal information, including pipe age, historical failure occurrences, maintenance records, etc. Furthermore, the pipes are distributed geographically, which allows us to examine the methods of graph construction. In general, there exist two critical pieces of information embedded in pipe networks, i.e., the structure of pipe connectivity and geographical neighboring effects. Concerning the pipe connectivity, industry practices indicate that if a pipe fails, more failures would be observed on the pipes that are on the same route as this pipe (connected pipes) due to physical effects such as water hammer [115]. In terms of geographical neighboring effects, more failures would be also observed on the nearby pipes even which have no connections to the failed one, since these pipes are exposed to similar environmental factors such as soil properties, ground vibrations, etc. [11, 98].

Lastly, the problem of pipe failure prediction can be transformed into a node classification task, where we treat pipes as nodes in the graph construction.

We present a graph-based failure prediction framework, named MAG (Multi-hop Attention-based GNN) by employing a GNN [151] to learn the features, structure, and geographical neighboring information in the pipe network. GNNs have achieved great success in many applications [84, 157]. The basic idea behind GNNs is to propagate and aggregate neighborhood information over graph structure. However, there are several challenges in applying GNNs to the pipe-network failure prediction, which the proposed MAG is able to address.

- The first challenge is constructing a graph structure that should join the structural and geographical information for the GNN. In this work, we represent the pipes as nodes. Naturally, the edges between nodes can be constructed according to the physical joints between pipes. On the other hand, we treat two nodes as linked if the geographical distance between two corresponding pipes is within a range.

- Furthermore, the neighborhood message aggregation of GNN aims to remove the abrupt noise. However, this aggregation may cause the loss of key information of pipes for failure prediction, especially when the connected pipes vary widely in the water network. MAG will adopt an attention mechanism to learn adaptive attention weights for the adjacent nodes according to their relevance and thus differentiate the dissimilar neighbors in the aggregation process.

- Moreover, GNNs provably suffer from over-smoothing [78] that the representation of the target central node in deep GNN layers may be over smoothed by averaging the information from a too wide range of neighbors, and as a result, the useful information of the target node may be "washed out". To overcome this issue, we apply GNN techniques [151, 75] including residual connections and layer-wise aggregation to MAG, which prevents the learning from over-smoothing by reserving the latent representations of each hidden GNN layer for the final prediction.

In addition, it has been recognized that the pipes' historical failures have high temporal effects on the current state of pipes [31]. To learn the temporal failure pattern, we develop a module as the complement to the GNN module. Inspired by the point process, this module learns two independent temporal effects as shown in Fig. 5.1. One is the base aging effect based on the fact that aging pipes will fail more frequently. The other one is the stimulating effect that a failure will cause more failures shortly. [101, 152].

In this work, we are provided with two real-world large-scale pipe networks by our partner water utilities. Our framework is evaluated on these datasets and experimental results show that it outperforms the statistical and machine learning models, as well as the state-of-the-art GNN baselines.

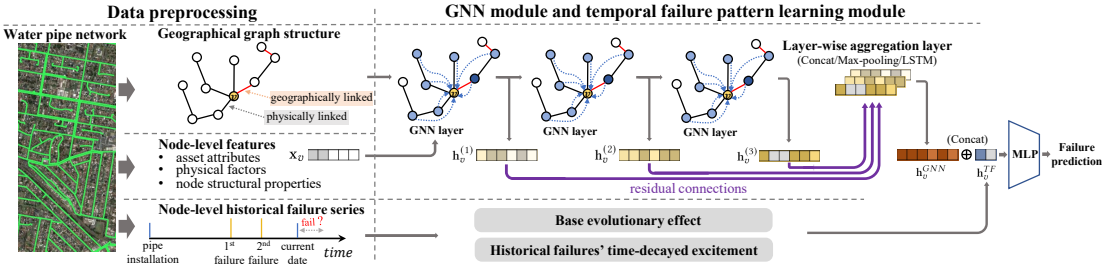


Figure 5.1 The architecture of our proposed framework MAG. It contains two main procedures: data preprocessing and failure prediction. Data preprocessing includes data collection, geographical graph construction, feature engineering, and temporal failure series extraction. In graph construction, two nodes are geographically linked (red edge) if two corresponding pipes without physical joint are geographically close to each other. In each GNN layer, we employ an attention mechanism and multi-hop aggregation. We add a layer-wise aggregation layer following the last GNN layer and the hidden representations of previous GNN layers are reserved in this layer through residual connections. In addition to the GNN module, We use a module to learn the temporal failure pattern including the base evolutionary effect and historical failures' time-decayed excitement on the current state of a pipe. MLP is used as the final failure predictor.

5.2 Related Work in Pipe Failure Prediction

The increasing urbanization has driven the great importance of the maintenance of civil water pipe networks. These networks are composed of a large number of pipes distributed around the whole city area. Any asset failure would lead to a regional or systematic level falling, breaks, or even disasters [80]. Therefore, preventative maintenance for pipes, particularly in aging urban-scale water networks, becomes of vital importance. Due to limited resources, water authorities can hardly afford to comprehensively inspect all assets. Instead, it is necessary to prioritize the pipes that require maintenance. Hence, the ability to identify pipes that are at high risk of failure is a fundamental need of water utilities [119, 70].

Failure analysis of the pipe distribution systems is a problem that has been studied over the past few decades. Historically, statistical models have been an effective tool in this field [31, 58, 80, 81]. Such models commonly treat the failure events as a stochastic point process and model the risk of failure occurrence as a function of time. A good number of statistical models have been applied, for example, Poisson point process [119], Weibull model [31], Cox survival analysis [101], Hawkes point process [152], random survival forests [58], and non-parametric Bayesian model [80].

However, statistical models suffer from some strict assumptions and data deficiencies. Many statistical models assume that failure data follows a specific distribution [101, 31], which is unrealistic in most real-world pipe failure predictions. Moreover, such models

highly rely on previous failure records, which is a problem in the real-world water industry where most pipes have never failed.

Recently, machine learning algorithms have been adopted to develop data-driven models for pipe failure prediction [109, 125, 81]. Compared with statistical models, machine learning models can take much more information including asset attributes, operation conditions, and surrounding environmental factors as input features, and produce more feasible pipe-level predictions [41]. Many failure prediction models have been proposed based on, for example, support vector machine [109], gradient boosting decision tree [125], artificial neural networks [81, 41], etc.

More recently, Farmani et al. [34] proposed an approach for the pipe failure prediction by combining the evolutionary polynomial regression model with K-means clustering approach. Kumar et al. [70] evaluated several machine learning methods on real-world pipe datasets. The experimental results show that the gradient boosting decision tree model outperforms the others. Deep learning has been used in the water industry. Liang et al. [81] developed a long-term hazard function based on recurrent neural networks. Their approach allows a black-box treatment for modeling the hazard function. The proposed model is successfully applied in proactive water pipe maintenance.

In essence, feature-based machine learning models avoid the issues that exist in statistical methods and achieve better predictive performance than statistical methods [125, 41]. However, all these models lack the ability to capture the structural information of pipe networks, which is also the motivation for this research work. As a contribution, we introduce a GNN-based failure prediction framework by jointly considering the features, structure, and temporal failure series.

5.3 Problem Formulation

Formally, given a pipe network with N pipes, a graph $\mathscr{G}(\mathbb{V}, \mathbf{A}, \mathbf{X})$ can be constructed by representing pipes as nodes, where $\mathbb{V}$ is the set of nodes, the adjacent matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ stores the structural information in which rows and columns are indexed by nodes, and $a_{i,j}$ in the i-th row and j-th column of $\mathbf{A}$ indicates the connectivity between node i and j. The value of $a_{i,j}$ is 0 if node i is not linked to j and non-zero if otherwise. The feature matrix $\mathbf{X} \in \mathbb{R}^{N \times f}$ represents the features of nodes (i.e., pipes) where the i-th row of $\mathbf{X}$ is the feature vector $\mathbf{x}_i \in \mathbb{R}^f$ of node i.

In addition, we denote $\mathbb{H}_i$ as the set of historical failure events of pipe i. We cast the failure prediction task as a binary classification problem of whether a pipe will fail within a future time window. Hence the task is to learn a model $\mathscr{M} : \left\{\mathscr{G}(\mathbf{A}, \mathbf{X}), \{\mathbb{H}_i\}_{i=1}^{N}\right\} \rightarrow \mathbf{y} \in \mathbb{R}^N$ where the i-th entry of $\mathbf{y}$ is the estimated failure risk of the i-th pipe.

5.4 Proposed Framework

The whole architecture of our proposed framework is illustrated in Fig. 5.1. In this section, we describe its main components including GNN module, temporal failure pattern learning, and the final prediction.

5.4.1 GNN Module

GNNs have achieved great success in a variety of graph applications by jointly learning the features and relational information of nodes [9, 157, 84]. Most GNNs follow a form of neighborhood information aggregation algorithm [64, 77], where the representation of a node is updated layer-by-layer through aggregating representations of its neighbors and combining the result with itself. Formally, the representation of every node $v \in \mathbb{V}$ in the l-th layer of a GNN is computed as:

$$\mathbf{h}_v^{(l)} = \text{COMBINE}^{(l)}\left(\mathbf{h}_v^{(l-1)}, \text{AGG}^{(l)}\left(\left\{\mathbf{h}_u^{(l-1)}\right\}_{u\in\mathbb{N}(v)}\right)\right) \tag{5.1}$$

where $\mathbb{N}(v)$ is the set of neighbors of node v. We initialize $\mathbf{h}_v^{(0)}$ with the feature vector $\mathbf{x}_v$ of node v. COMBINE$(\cdot)$ and AGG$(\cdot)$ are defined by specific model. GraphSAGE [46] uses concatenation in COMBINE$(\cdot)$ and proposes several AGG$(\cdot)$ functions based on LSTM architecture or max-pooling operation. Graph Convolutional Network (GCN) [64] averages the states of node v and its neighbors at the same time in the aggregation step. Some recent works use attention mechanism to highlight the information of more relevant neighbors in AGG$(\cdot)$, instead of treating all neighbors equally.

In our framework, we implement three recently published GNN techniques including multi-hop aggregation, attention mechanism, residual connections, and layer-wise aggregation.

5.4.2 Multi-hop Aggregation

Early GNNs including GCN [64], GraphSAGE [46], GAT [137], etc. implement 1-hop message aggregation in a GNN layer, which limits the expressive power of GNNs [151, 1, 77]. Recent works present provably more powerful multi-hop GNN architectures. MixHop [1] concatenates the outputs of multiple graph convolutional operations in a layer and each of the operations utilizes different power of the adjacency matrix. DEA-GNN [77] leverages the shortest path distance between any two nodes to control the message aggregation process in GNNs.

In this framework, we follow the multi-hop approach in DEA-GNN [77]. Specifically, the AGG$(\cdot)$ in Eq. 5.1 is changed into:

$$\mathrm{AGG}^{(l)}\left(\left\{\mathbf{h}_u^{(l-1)}\right\}_{u\in\mathbb{N}(v)}\right) \rightarrow \mathrm{AGG}^{(l)}\left(\left\{\mathbf{h}_u^{(l-1)}\right\}_{SPD(v,u)\leq \mathrm{k}}\right) \tag{5.2}$$

where $SPD(\cdot,\cdot)$ calculates the shortest path distance between two nodes, k is a hyper-parameter. Note that setting $k=1$ recovers the traditional 1-hop aggregation. Generally, assigning $k=2$ or 3 is sufficient, which ensures that the aggregation focuses on the local neighboring information and avoids "noisy" information from higher-order neighbors [151, 77]. In practice, k is chosen by balancing the trade-off between computational complexity and performance. Fig. 5.1 illustrates the multi-hop aggregation where the messages of 2-hop neighbors of the central node are aggregated in a GNN layer.

5.4.3 Attention-based GNN Layer

The neighborhood aggregation in the GNN layer may lead to biased representations of nodes, especially when the adjacent nodes have completely different properties. We employ an attention mechanism to tackle this issue. It allows the model to learn adaptive importance weight between two adjacent nodes and thereby differentiate neighbors in the aggregation procedure through highlighting the messages of more relevant nodes while suppressing the contributions of less relevant nodes. Several attention based GNNs have been proposed, such as GAT [137], AGNN [132], Gaan [166]. They present various attention mechanisms to learn the relevance between two adjacent nodes.

In this work, we employ the attention mechanism proposed by GAT [137]. Formally, the attention coefficient for the neighbor node u of central target node v is as follows:

$$\alpha_{vu}^{(l)} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^{(l)}\cdot CONCAT\left(\mathbf{h}_v^{(l-1)},\mathbf{h}_u^{(l-1)}\right)\right)\right)}{\sum_{SPD(v,w)<\mathrm{k}}\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^{(l)}\cdot CONCAT\left(\mathbf{h}_v^{(l-1)},\mathbf{h}_w^{(l-1)}\right)\right)\right)} \tag{5.3}$$

where $\mathbf{a}^{(l)}\in\mathbb{R}^{2d}$ is a trainable weight vector for the l-th layer, $\mathbf{h}_v^{(l-1)}\in\mathbb{R}^d$ and $\mathbf{h}_u^{(l-1)}\in\mathbb{R}^d$ is the output states of node v and u from the $(l-1)$-th layer, respectively. $CONCAT(\cdot,\cdot)$is the concatenation operation. LeakyReLU is an activation function.

For the final formulation of the GNN layer in Eq. 5.1 in our GNN module, we set COMBINE$(\cdot)$ as concatenation and AGG$(\cdot)$ as mean-pooling in the first GNN layer. Combined with the attention mechanism in Eq. 5.3, the output representation of node v given by the first GNN layer is formalized as:

$$\begin{aligned}&\mathbf{h}_v^{(1)} = \mathrm{ReLU}\left(CONCAT\left(\mathbf{x}_v,\mathbf{W}^{(1)}\cdot\sum\alpha_{vu}^{(1)}\mathbf{x}_u\right)\right)\\&\text{where } u: SPD(v,u)<\mathrm{k}\end{aligned} \tag{5.4}$$

where $\mathbf{W}^{(1)}$ is a trainable weight matrix. This setting ensures that the original features $\mathbf{x}_v$ of node v can be reserved in $\mathbf{h}_v^{(1)}$.

Since we use residual connections and layer-wise aggregation in our GNN module (see section 5.4.4), the concatenation operation is no longer necessary for the remaining GNN layers. Therefore, we set the l-th ($l > 1$) GNN layer as:

$$\begin{aligned} &\mathbf{h}_v^{(l)} = \mathrm{ReLU}\left(\mathbf{W}^{(l)} \cdot \sum \alpha_{vu}^{(l)} \mathbf{h}_u^{l-1}\right) \\ &\text{where } l > 1; u : SPD(v,u) < \mathrm{k} \text{ or } u = v \end{aligned} \tag{5.5}$$

where $\mathbf{W}^{(l)}$ is the weight matrix for the l-th layer.

5.4.4 Residual Connections and Layer-wise Aggregation

Common GNNs based on neighborhood aggregation algorithm tend to be shallow and deeper GNNs provably suffer from over-smoothing and gradient degeneration [78, 151]. Stacking increasingly deep layers in the GNN may over average the information from a too wide range of neighbors, and as a result, the useful information of the central target node may be "washed out". To overcome this issue, JKGNN [151] studies the range of neighbors that can influence the central node's representation and proposes a jump knowledge architecture. DeeperGCN [75] presents a novel normalization layer and a pre-activation version of residual connections for GNNs.

To further enhance the expressive power of our framework, we adopt the idea of JKGNN [151]. As illustrated in Fig. 5.1, we add a layer after the last GNN layer. This layer stores the hidden representations from the previous GNN layers. In other words, the output of each GNN layer has a residual or jump connection to this layer. We implement a layer-wise aggregation on these representations in this layer. Consequently, the final representation of node v produced by our GNN module is

$$\mathbf{h}_v^{GNN} = \mathrm{AGG}\left(\left[\mathbf{h}_v^{(1)}, \cdots, \mathbf{h}_v^{(l)}\right]\right) \tag{5.6}$$

We adopt three aggregation schemes for $\mathrm{AGG}(\cdot)$, including concatenation, max-pooling, and LSTM-attention. Concatenation combines the representations together while max-pooling applies a $\max(\cdot)$ over the representations. Both concatenation and max-pooling are not node-adaptive and do not introduce additional parameters associated with the specific node. In contrast, LSTM-attention is node adaptive and needs to calculate the attention score for each node.

5.4.5 Learning Temporal Failure Pattern

The historical failures of pipes are of particular importance to the future failure prediction. Stochastic point process has proven to be an effective tool for dealing with such temporal failure data. In essence, point process [48] is characterized with the conditional intensity function $\lambda(t)$, where $\lambda(t)dt$ is the likelihood for an event occurring within a small window $[t, t+dt]$, given the historical events before t. Various point process models have been proposed such as Poisson point process [119], Hawkes point process [48], Cox point analysis [101], etc. Specifically, the $\lambda(t)$ for Hawkes process [48] is given by

$$\lambda(t) = \lambda_0(t) + \sum_{t_h \in \mathbb{H}, t_h < t} \kappa(t - t_h) \tag{5.7}$$

where t_h is the time of the h-th historical event occurred before t and $\mathbb{H}$ is the set of historical events, $\lambda_0(t) \geq 0$ is the base intensity that is a function of t but independent of the historical events. $\kappa(\cdot)$ is a kernel function that describes how much historical events excite the occurrence of an event at t.

To capture the temporal pattern in the failures data, we add a temporal failure pattern learning module that is inspired by the Hawkes process to our framework. Formally, the node v's representation $\mathbf{h}_v^{TF}$ of Temporal Failures' effect given by this module is:

$$\begin{aligned} h_v^T &= \mu + (\omega - \mu)e^{-\delta t} \\ h_v^F &= \sum_{t_h \in \mathbb{H}_v, t_h < t} \alpha e^{-\beta(t - t_h)} \\ \mathbf{h}_v^{TF} &= \text{ReLU}\left(CONCAT\left(h_v^T, h_v^F\right)\right) \end{aligned} \tag{5.8}$$

where $\mu > 0, \omega > 0, \delta > 0, \alpha > 0, \beta > 0$ are trainable parameters that are shared across all pipes, t is the age of pipe v at the time of prediction, $\mathbb{H}_v$ is the set of historical failures defined in Eq. 5.9.

In our framework, we call h_v^T as *base evolutionary effect* that is only a function of pipe age t, where ω is the initial failure risk at $t = 0$, δ is the rate of exponential decay, and μ describes the reversion level. For example, with the age t being increased, the h_v^T is exponentially decaying if $\omega > \mu$, while exponentially growing if $\omega < \mu$. We call h_v^F as *historical failures' time-decayed excitement* where α is the size of excitement jump for exponential decay function. h_v^F is an aggregation of the excitement effects of all historical failures of pipe v on its state at time t.

5.4.6 Failure Predictor

We use a multi-layer perceptron (MLP) as the predictor of our framework. As shown in Fig. 5.1, it takes the concatenation of the GNN output and temporal failure representation,

i.e. $CONCAT(\mathbf{h}_v^{GNN}, \mathbf{h}_v^{TF})$, as input and predict the final failure risk. The whole model is optimized with the binary cross-entropy loss in an end-to-end manner. The target is assigned as 1 for a pipe if a failure event occurred on this pipe in a given time window, otherwise 0. The time window is set according to the down-streaming task.

It is worth noting that the weights to be trained in our model are shared across all nodes (pipes). This can significantly reduce the computational complexity. On the other hand, it allows our model to be trained with a batched scheme, which is of great importance, especially for the real-world large-scale infrastructure networks.

5.5 Experiments

5.5.1 Datasets

In this application, we are provided with two real-world (water and sewer) pipe datasets including pipe attributes and failure records. We study pipe failures in terms of leaks or breaks on water pipes and chokes on sewer pipes. Table 5.1 summarizes the basic information of two datasets. We also collect *soil* [1], *elevation* [2], and *tree canopy* [3] data from the public resources. Fig. 5.2 shows a part of our data on a map.

Table 5.1 Basic statistics for two pipe datasets

Datasets	Pipes	Joints	Failures
water pipe network	363,648	257,362	82,525 (leaks or breaks)
sewer pipe network	941,142	2,947,183	201,330 (chokes)

5.5.2 Data Preprocessing

Geographical Graph Structure Constructing The pipe network is geographically distributed in a large-scale area. Pipes that are nearby to each other tend to exhibit similar failure trends, regardless of whether the pipes are physically connected or not. Different from the graph construction methods for common networks such as social networks [9], recommender systems [157], and biochemical interaction networks [84], in this work, we propose a method of constructing geographical graph structure according to not only the physical connections but also the distance between pipes. Formally, the geographical graph structure is defined as:

Definition 1. ***Geographical graph structure***. *Let geographical graph structure be represented by* $\mathbf{A} \in \mathbb{R}^{N \times N}$ *that is the adjacency matrix of the graph with N nodes.* $a_{i,j}$ *in the*

[1] http://www.clw.csiro.au/aclep/soilandlandscapegrid
[2] http://elevation.fsdf.org.au
[3] https://data.gov.au

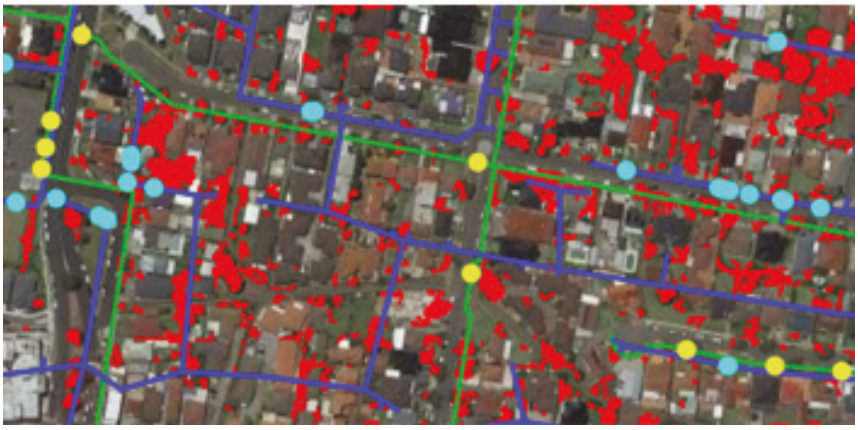


Figure 5.2 An example of our data shown on the map. It includes water pipes (green lines) and failures (yellow dots), sewer pipes (blue line) and failures (light blue dots), as well as tree canopy (red polygons).

i-th row and j-th column of $\mathbf{A}$ *is assigned as a non-zero value if either node i and node j are connected or* $dist(i,j) < \rho$*, where* $dist(i,j)$ *calculates the distance between nodes i and j. Otherwise,* $a_{i,j} = 0$.

In this application, we use the centroid of each pipe in the dataset as the node and construct the unweighted geographical graph structure. As shown in the Table 5.1, the constructed graph consists of 363,648 and 941,142 nodes for water pipe dataset and sewer pipe dataset, respectively.

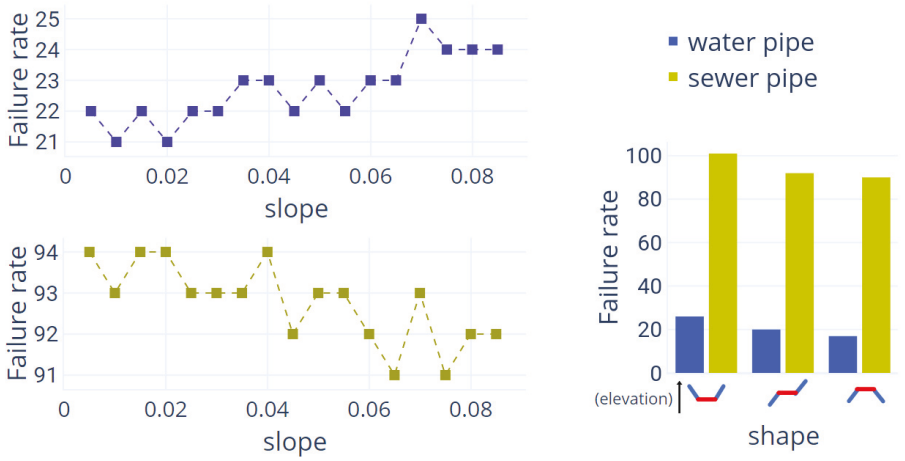


Figure 5.3 Failure analysis on elevation-based features. Failure rate is defined as the number of pipe failures per 100KM per year. Slope is the elevation difference between two ends of a pipe divided by pipe length. More than 95% of pipes have the slope less than 0.08. There are three types of shapes for three adjacent pipes in the elevation direction.

Feature Engineering Feature engineering is a commonly used approach to enhance the performance of the model. Table 5.2 lists the features we crafted. For pipe basic attributes,

Table 5.2 Feature description

Feature	**Description**
Age	age of pipe at the time of prediction
Length	pipe length
Diameter	pipe diameter
Failures	number failures in the last $1, 5, 10$ years
Material	water pipe: CICL, DICL, PVC, others sewer pipe: VC, SGW, PVC, PE, others
Coating	water pipe: Tar, Concrete, others sewer pipe: None
Soil	AWC, BDW, CLY, DER, DES, ECE, NTO, PHC, PTO, SLT, SND, SOC
Tree canopy	tree canopy coverage
Slope	elevation variance of a pipe
Shape	vertical shape of three connected pipes
Node degree	node degree
Node PR	node PageRank [45]
Node cluster	clustering coefficient [45]

age, length, and diameter are continuous features. For failures, we count the number of failures for each pipe in the last $1, 5, 10$ years, respectively. Pipe material and external coating material (only water pipes have) are categorical features. We group the materials that are used rarely into "others" and then one-hot-encode them.

Environmental and physical conditions are important factors leading to pipe failure. We craft 12 soil features that indicate the water capacity, density, composition (clay, silt, sand, organic carbon), effective cation exchange, pH, etc. It has been recognized that tree root is the main cause of pipe failure since the roots are more likely to grow around pipes and absorb water and nutrients from pipes [98, 108]. Based on the available tree canopy data, we use the percentage of a pipe covered by the tree canopy as the feature of this pipe. We also study the failure rate on the features engineered based on elevation data. Two elevation-based features are used for failure prediction. As shown in Fig. 5.3, one feature is the slope that reflects the elevation variance of a pipe. The other one is the vertical shape of three adjacent pipes for the middle pipe.

In addition, we calculate three node-level graph structural properties including node degree, PageRank, and clustering coefficient [45]. Overall, there are 31 and 29 features for water and sewer pipes, respectively.

Temporal Failure Series For a pipe, the influence of historical failures on its current state shows a strong temporal pattern. In the data preprocessing as shown in Fig. 5.1, we

extract the failure records of each pipe i into a time series:

$$\mathbb{H}_i = \{t_0, \cdots, t_h, \cdots\} \tag{5.9}$$

where t_h is a time interval from the installation to the h-th historical failure event of pipe i.

5.5.3 Experimental Settings

This section describes our experiment setups, the evaluation results on two real-world datasets, and further analysis including the sensitivity study of aggregation methods, failure detection rate analysis, hyper-parameter tuning, and training time.

Baselines We compare the performance of our framework against three classes of baselines. (1) statistical models: Weibull model [31], Hawkes Process model [152], Random survival forests (RSF) [58]. Note that the statistical models commonly predict the survival likelihood for a group of pipes as a function of time. We transform such predictions into failure probability for individual pipes [41]. (2) feature-based machine learning models: Random Forests (RF), Support Vector Machine (SVM) [109], Gradient Boosting Decision Tree (GBDT) [23], and a fully-connected neural network (MLP). (3) graph-based GNNs: plain GCN [64], GraphSAGE [46], GAT [137], JKGNN [151], DeeperGCN [75] and DEA-GNN [77].

For brevity, we abbreviate our proposed Multi-hop Attention based GNN (MAG) that uses Concatenation, LSTM, or MAX-pooling in the layer-wise aggregation as MAG-Concat, MAG-LSTM, or MAG-MAX, respectively. Besides, we also report the result of MAG-nonGeo that uses the graph structure without geographical edge.

Evaluation metrics. We use four basic metrics [85] including AUC, F1 score, Precision, and Recall to compare the performance of the models. In addition, the goal of this work is to find the pipes that are at high risk of failure. Therefore, the predictor is expected to rank the failed pipes higher than the good ones in the validation step. In this work, we use the detection rate to measure such ranking performance of the models. Formally, *detection rate*@k is defined as:

$$\textit{detection rate}@k = \frac{N_{detected}(k)}{N_{total}} \tag{5.10}$$

which is the fraction of failed assets $N_{detected}(k)$ that are successfully detected in the top-k of ranking predictions among all failed assets N_{total}.

Implementation Our method is implemented based on Pytorch. We adopt Adam optimizer with a learning rate $\in \{0.001 \sim 0.0001\}$ and a dropout rate $\in \{0, 0.1, 0.2, 0.5\}$. We

report the best result across these hyper-parameters. We implement batch normalization for stable and fast training [33]. We stop the optimization process when the AUC on the validation set does not increase for 50 epochs. Testing is performed with the optimized model. The experiments are repeated 10 times with randomly extracting validation and testing set and the mean results are reported. We run all the experiments on a machine with a NVIDIA A100 GPU, 32-cores CPU, and 256 GB of RAM.

Table 5.3 Results for two pipe networks

Model	**water pipe network**				**sewer pipe network**			
	AUC	F1	Precision	Recall	AUC	F1	Precision	Recall
Weibull [31]	0.667	0.261	0.167	0.458	0.631	0.197	0.136	0.427
Hawkes [152]	0.685	0.278	0.178	0.464	0.65	0.221	0.158	0.441
RSF [58]	0.706	0.269	0.183	0.482	0.668	0.263	0.165	0.466
RF	0.765	0.372	0.294	0.586	0.771	0.357	0.247	0.559
SVM [109]	0.787	0.373	0.318	0.591	0.775	0.362	0.296	0.563
MLP	0.769	0.369	0.309	0.581	0.773	0.355	0.285	0.561
GBDT [23]	0.811	0.411	0.351	0.615	0.797	0.396	0.335	0.587
GCN [64]	0.789	0.3686	0.336	0.578	0.761	0.342	0.312	0.545
GraphSAGE [46]	0.798	0.37	0.336	0.585	0.772	0.353	0.314	0.556
GAT [137]	0.828	0.432	0.379	0.637	0.815	0.417	0.354	0.612
JKGNN [151]	0.837	0.439	0.405	0.665	0.835	0.425	0.381	0.638
VAGNN (Chapter 3)	0.842	0.428	0.401	0.667	0.836	0.431	0.383	0.632
DeeperGCN [75]	0.839	0.439	0.391	0.657	0.829	0.4291	0.383	0.631
DEA-GNN [77]	0.806	0.391	0.351	0.607	0.791	0.375	0.329	0.574
MAG-nonGeo	0.842	0.441	0.405	0.669	0.839	0.433	0.385	0.642
MAG-Concat	**0.855**	**0.457**	0.434	**0.683**	0.853	0.449	**0.401**	0.659
MAG-LSTM	0.851	0.452	**0.437**	0.679	**0.855**	**0.452**	0.398	**0.661**
MAG-MAX	0.831	0.433	0.371	0.651	0.822	0.419	0.377	0.628

5.5.4 Main Results

Table 5.3 summarizes the results on two datasets. It suggests that the feature-based machine-learning algorithms are consistently better than statistical models. GBDT achieves the best among traditional machine learning models. We implement GBDT using a variant, i.e. XGBoost [23] which is a provably powerful decision-tree model. It has gained state-of-the-art results in many industrial applications and data science tasks, especially for the data that contains many categorical features. For GNN based baselines, GCN [64] and GraphSAGE [46] achieve poor performance. Both methods also underperform the GBDT. One reason is that the strategy of information aggregation in GCN is average, which may gradually wash out the useful information of pipe failure prediction [78]. GraphSAGE uses a concatenation scheme in the aggregation stage, which makes its performance better than GCN. Although GraphSAGE reserve the original features of nodes in the first layer, it has no residual connections to the deep layers and has the same over-smoothing issue as GCN. DEA-GNN [77] is a form of multi-hop GNN. It performs slightly better than

GraphSAGE, which demonstrates the power of multi-hop aggregation. However, DEA-GNN does not use methods such as attention mechanism or residual connections to address the over-smoothing issue and performs worse than GBDT.

GAT [137], JKGNN [151], and DeeperGCN [75] outperform the other baselines. GAT employs a multi-head attention mechanism in the aggregation process of each GNN layer, which can highlight the information of more relevant neighbors while suppressing the contributions of less relevant nodes. Its result indicates the importance of attention mechanisms when applying GNNs in real-world graph applications, for example, failure prediction in pipe networks where pipes are with greatly different attributes. JKGNN and DeeperGCN achieve the best performance among all baselines. Both of them use the residual scheme and attention aggregation to overcome the issues existing in GNNs [75].

For the variants of our model, MAG-Concat and MAG-LSTM yield the best performance on both datasets. In practice, MAG-LSTM has more trainable parameters than MAG-Concat and thus pays an additional computational cost. MAG-nonGeo outperforms the best baselines but underperforms the MAG-Concat and MAG-LSTM. MAG-nonGeo takes the graph structure that does not consider the geographical neighboring effect. The GNN architecture of our framework is much similar to JKGNN and DeeperGCN. MAG-nonGeo is still better than them, indicating that the temporal failure representations learned by our model have a clear effect on failure prediction. MAG-MAX performs better than GAT while worse than the best results of JKGNN and DeeperGCN. This may be because the layer-wise max-pooling aggregation biases the representations and is not suitable for pipe failure prediction.

5.6 Further Analysis

5.6.1 The Sensitivity of Aggregation Methods

To investigate the sensitivity of COMBINE$(\cdot)$ and AGG$(\cdot)$ methods in the setting of the GNN layer (Eq. 5.1), we studied the performance of two baseline GNNs (i.e. plain GCN [64] and GraphSAGE [46]) on our two real-world pipe datasets. The comparison result is shown in Fig. 5.4. GraghSAGE with concatenation as COMBINE$(\cdot)$ and mean-pooling as AGG$(\cdot)$ achieves the best, followed by LSTM-aggregator. GCN shows the third-best result and outperforms the GraphSAGEs with a max-pooling setting.

This result recovers several characteristics of our datasets. Firstly, the promising performance of concatenation means that the features of a pipe itself play a critical role in the failure prediction. Secondly, the majority of pipes in two datasets have a degree of 2, namely a pipe is connected to only two pipes. Mean-pooling aggregation outperforms slightly LSTM-aggregator, indicating that mean-pooling may be more suitable for our dataset while the degree of most nodes is too small for LSTM-aggregator. Moreover,

max-pooling is not suitable for both COMBINE$(\cdot)$ and AGG$(\cdot)$. This may be because max-pooling selects the maximum of the features of neighbors and "washes out" the important properties of the target pipe, especially considering the fact that the pipe failure is highly sensitive to the pipe attributes such as length, size, age, and many more [83, 70].

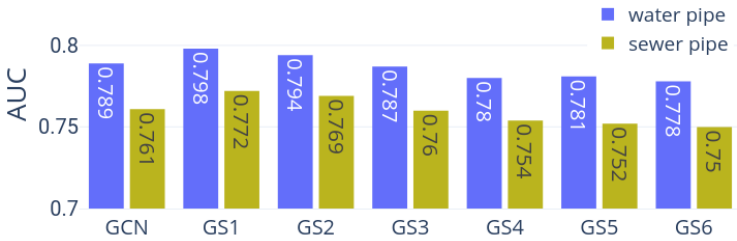


Figure 5.4 Comparison of baseline GCN [64] and GraphSAGE [46]. GS1 is the GraphSAGE with concatenation as COMBINE$(\cdot)$ and mean pooling as AGG$(\cdot)$, GS2: concatenation and LSTM aggregator, GS3: concatenation and max pooling, GS4: max and mean pooling, GS5: max pooling and LSTM aggregator, GS6: max and max pooling.

5.6.2 Failure Detection Rate

The performance on the detection rate is of great importance for real-world pipe prioritization. Proactive maintenance is usually time-consuming and requires a lot of financial and labor costs, particularly for large-scale pipe networks. Decision-makers are much concerned about the detection rate of high-risk pipes in the top ranking predictions.

Fig. 5.5 and Fig. 5.6 show the detection rate at top ranking predictions for water and sewer dataset, respectively. RSF performs the worst, which is consistent with previous works [125, 41]. It has been recognized that statistical models are more appropriate for long-term pipe management planning while less competitive in pipe-level prediction compared with feature-based models [125]. Overall, our framework improves the detection rate by around 10% compared with the best baseline JKGNN. More importantly, the proposed framework detects more than 50% of failures in the top 20% of ranking pipes, meaning that 50% of pipes that are at high risk of failures can be detected if we only inspect the top 20% of the prioritized pipes in practice.

5.6.3 Hyper-parameters and Training Time

The best-tuned hyper-parameters of our framework are summarized in Table 5.4. The ρ in Def. 1 restricts the distance range within which we treat the pipes as geographically neighboring. It is $100m$ for water pipes and $45m$ for sewer pipes. This may be because the majority of sewer pipes are shorter than most water pipes. The best batch size is 1500 and 1000 for two datasets, which is much larger than common graph applications such as

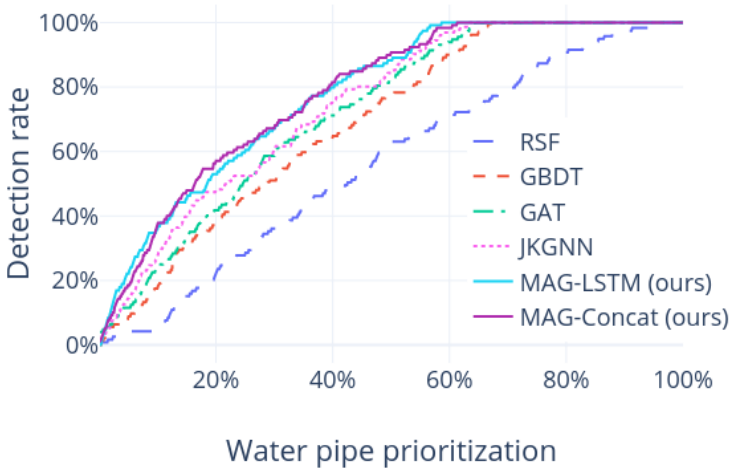


Figure 5.5 Failure detection rate vs ranking predictions on water pipe dataset.

Table 5.4 Best tuned hyper-parameters for our framework

Hyper-parameter	Water pipe dataset	Sewer pipe dataset
ρ in Def. 1	$100(m)$	$45(m)$
k in Eq. 5.2	2	2
GNN layers	3	4
Batch size	3000	2000
Dropout rate	0.2	0.1

social networks [9] and recommender systems [157]. This may be due to the extreme label imbalance in our datasets where most pipes are not failed.

Our framework can be optimized with batch normalization on a machine with NVIDIA A100 GPU, 32-cores CPU, and 256G RAM using about 10 hours, and predict all pipes (Table 5.1) using less than 2 minutes when we put all data on RAM where the maximum usage of RAM is up to 124G.

5.7 Application

5.7.1 Pipe Prioritization Platform

Although we do not give an explicit indication for future pipe failure, the water utility can deploy our work to obtain pipe risks for their down-streaming maintenance work and fit it into a proactive maintenance plan. For example, our prediction produces a risk ranking of the pipes for a given cohort, which addresses the pipe prioritization. Accordingly, we developed the interactive Pipe Prioritization Platform, named Pcube, and

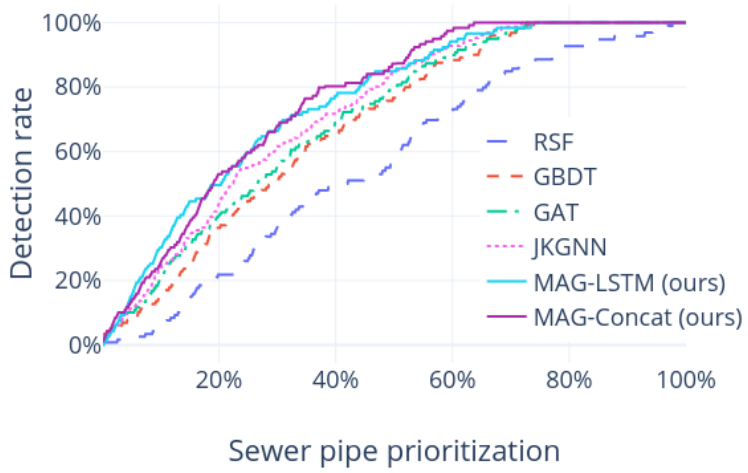


Figure 5.6 Failure detection rate vs ranking predictions on sewer pipe dataset.

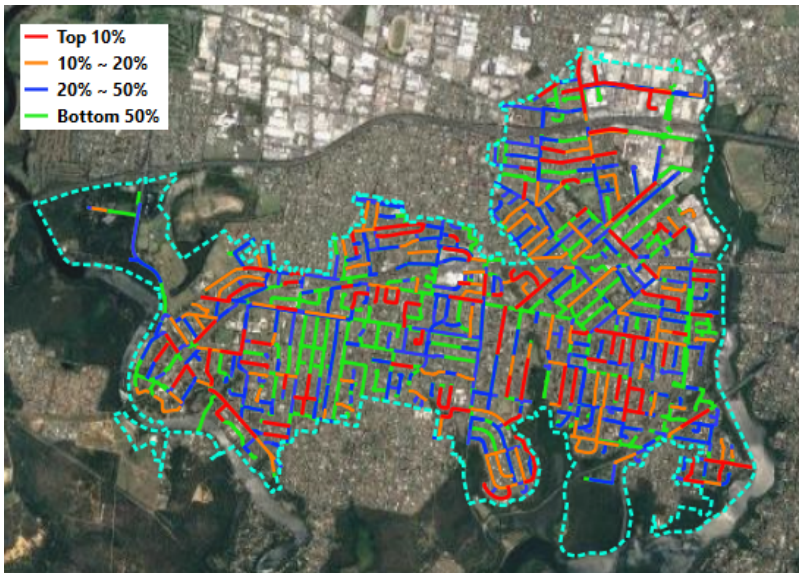


Figure 5.7 An example of risk map based on pipe prioritization.

embedded our proposed framework as the data processing component and the machine learning component. The platform also includes an analytic component and a visualization component. The visualization can provide prioritization and risks in GIS-based UI designs.

We show the risk map plot in Fig. 5.7. The map is based on Pcube deployment at a metropolitan-level water utility for the proactive maintenance purpose. The map showing risks of individual pipes provides operational convenience for planners to understand the potential impact of maintenance work and the potential pipe burst. The map shows our direct output. However, in reality, the analytic component will be involved to further discuss

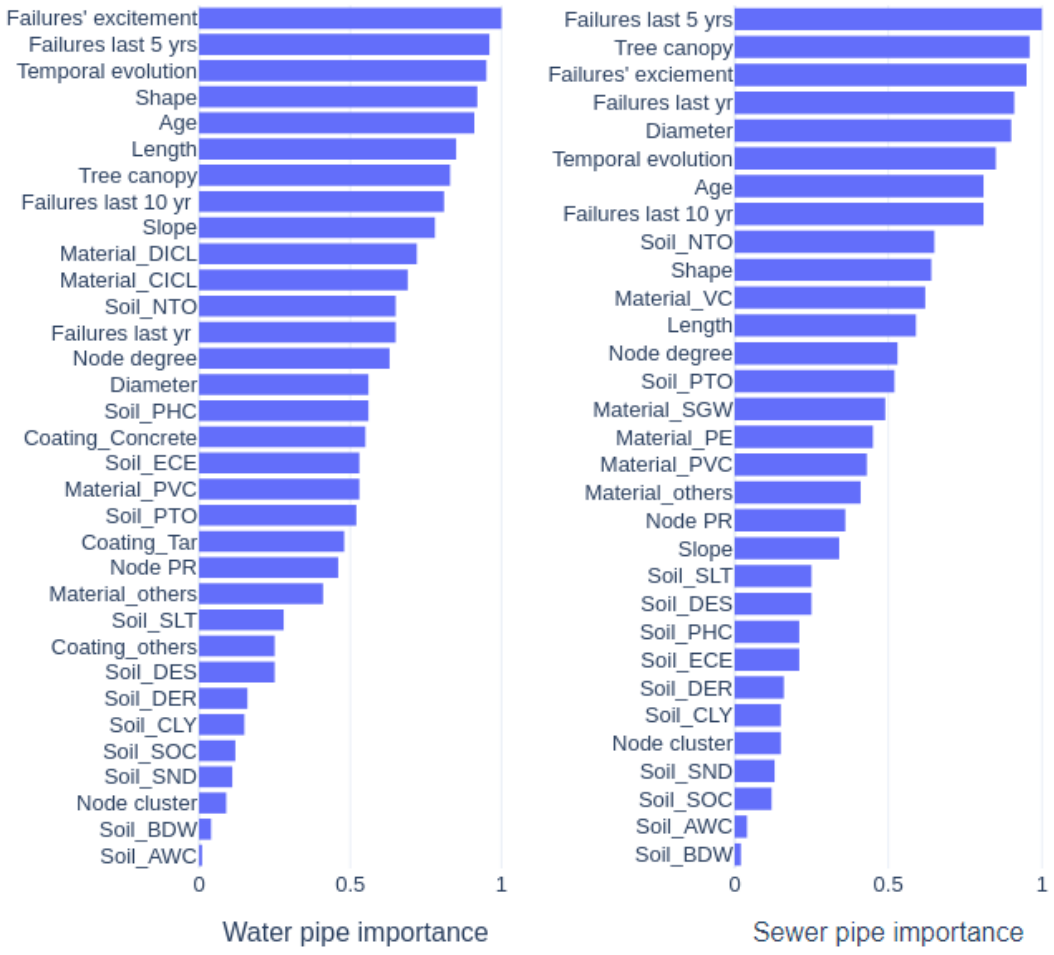


Figure 5.8 Feature importance.

the risks, by considering both our model output and the social-economic consequence of failures.

5.7.2 Feature Importance

The on-site maintenance, even condition assessment, is costly. Although a huge amount of budget has been planned, water utilities must know why the prediction has been made[4]. To show this, we include the feature importance in the analytic component.

We use the permutation feature importance method [7], where the feature importance score is calculated as the increase of prediction error when a feature is permuted, for example, the values in one feature channel are replaced with random values. Specifically, we have the original AUC^{orig} as shown in Table 5.3. We then calculate the permuted AUC^{perm} with the same settings but a feature channel is permuted. Then the feature importance can be obtained by the AUC difference $\text{AUC}^{orig} - \text{AUC}^{perm}$.

The feature importance results are shown in Figure 5.8. For water pipes, three failure-related features are the most important ones. The high importance of the features "failures' excitement" and "temporal evolution" indicates that the layer we especially used to learn the temporal failure pattern is a critical supplement to the GNN module. The elevation-based features "shape" and "slope" are also ranked top in the results, which is consistent with the analysis as shown in Fig. 5.3. Four soil properties (NTO, PHC, ECE, PTO) are relatively more important than other soil features. This may be because high NTO (Total Nitrogen), PHC (soil pH), ECE (Effective Cation Exchange), and PTO (Total phosphorus) could corrode the pipes [11]. The result shows that the material DICL (Ductile Iron Cement Lined pipe) and CICL (Cast Iron Cement Lined pipe) are important for failure prediction, which is consistent with previous studies [105].

For sewer pipes, the historical failures are the same important as for water pipes, indicating that learning previous failures information is of great importance for the failure prediction model. This can also explain why many traditional statistical models can give reliable predictions only using historical failures. Besides, the features of sewer pipes such as diameter, material VC, soil NTO, and PTO are also important, which have also been recognized by domain experts.

Another example of lessons learned from the model is the importance of "tree canopy". It is at the seventh and the second place in the feature importance result, respectively. Accordingly, the industry investigation revealed that the tree root is one of the main causes of pipe failures for both water and sewer system [98, 108].

[4]Please note that it is not necessary for the causal reasoning, but it is useful for domain experts to understand/check.

5.8 Summary

In this work, we develop a GNN-based failure prediction framework for pipe networks by jointly considering the features, structure, geographical neighboring effects, and historical failure information. We propose a novel method to construct graph structure depending on the physical connections and geographical distances between pipes. We employ multi-hop aggregation, attention mechanism in each GNN layer, and the residual connections and layer-wise aggregation after the last GNN layer to tackle the issues of GNNs. We develop a temporal failure pattern learning module in our framework to learn the base evolutionary effects and historical failures' time decayed excitements on the state of pipes. The proposed prediction model is evaluated on two real-world pipe networks and achieves state-of-the-art results. The primary task of our framework is to produce pipe prioritization which provides the water utility with data-driven support for pipe maintenance. Also, we study the feature importance which can help water experts to further identify and understand the failure causes. Our framework can be extended to large-scale infrastructure networks for asset-level failure prediction.

Chapter 6

Conclusion and Future Work

This chapter summarizes the main contributions of this thesis as well as discusses the potential future work.

6.1 Conclusion

In conclusion, this thesis explores the generalization, scalability, and interpretability of GNNs. In Chapter 2, we present a comprehensive literature review of GNNs and related graph representation learning methods, highlighting their unique ability to handle graph-structured data. The development of GNNs is traced back to the introduction of GCN, which adapted the concept of convolution to the graph domain. Since then, various GNN architectures have been proposed, each addressing different challenges and leveraging different aspects of graph data.

The thesis consists of three research works, each contributing to the advancement of GNNs in different domains. The first research work, presented in Chapter 3, introduces VAGNN, a general and scalable GNN framework. VAGNN incorporates an optimizable virtual adjacency matrix to improve the generalization and scalability of GNNs. It enables flexible construction of the neighborhood set for aggregation, allowing for the inclusion of high-hop neighbors and global nodes as well as selective removal of 1-hop neighbors. VAGNN offers a selection of various attention mechanisms and the option to incorporate supplementary information for enhanced control over the aggregation process, improving the generalization ability of VAGNN. Theoretical analysis demonstrates the scalability of VAGNN, making it well-suitable for large graphs. Experimental evaluations validate the generalization and scalability capabilities of VAGNN, laying the foundation for further exploration in developing more efficient techniques.

The second research work, discussed in Chapter 4, focuses on the interpretive analysis of GNNs for link prediction. It investigates the ability of GNNs to learn pair-specific structural information related to the number of common neighbors for link prediction. The

research work identifies the limitations of existing GNN-based link prediction models and explores the impact of different components such as node features, node embeddings, and heuristic information. The empirical evaluations on real-world datasets provide valuable insights into the strengths and limitations of GNNs for link prediction tasks.

The third research work, presented in Chapter 5, demonstrates our exploration of the roles of various GNN techniques in solving real-world applications. It proposes a GNN-based approach called MAG to predict potential failures in water pipe networks. MAG leverages the spatial and temporal information in the pipe network, including the structural connectivity and geographical neighboring effects. The framework addresses challenges such as capturing both structural and geographical information, handling over-smoothing in GNNs, and incorporating temporal patterns. Experimental evaluations on real-world datasets demonstrate the superiority of MAG over statistical and machine learning models.

Overall, this thesis has advanced the understanding and application of GNNs in various domains. It has contributed to the development of novel GNN frameworks, provided insights into the interpretability of GNNs for link prediction, and investigated the functionality and effectiveness of advanced GNN approaches in practical problems. The research works presented in this thesis have laid the groundwork for further advancements in GNNs and their applications, contributing to the ongoing progress in graph representation learning and analysis of complex systems represented as graphs.

6.2 Future Work

Based on the findings presented in this thesis, several promising avenues for future research emerge.

Neighborhood Construction Methods The significance of neighborhood construction methods, as elucidated by our VAGNN framework, cannot be overstated. Our empirical demonstrations underscore the pivotal role of varying neighborhood construction techniques in influencing the efficacy of distinct GNN models across diverse graph-oriented tasks. Embracing more adaptable approaches to neighborhood construction holds the potential to catalyze the evolution of more robust and potent GNN architectures. Furthermore, a compelling research trajectory lies in the realm of automating model-driven neighborhood method selection. This entails investigating strategies to enable the model to autonomously determine the optimal combination of 1-hop neighbors, local neighboring nodes, and global nodes ratio, thereby maximizing performance outcomes.

Negative Samples in Link Prediction In our second research work, the link prediction model training is supervised by positive samples (actual links) and negative samples

(spurious links). In our experiments, we have noticed the pivotal role played by the selection of negative samples in molding model performance. We observed that distinct choices in this regard could wield significant influence. The underlying mechanism is clear, i.e., the inclusion of negative samples compels the model to assign lower scores to links during training, thereby bestowing superior link prediction capabilities upon non-connected node pairs within the test set. However, it is crucial to note that this approach could introduce a potential challenge. Namely, instances that are negative during training but transition to positive in the test set can result in a notable performance decline. This phenomenon opens up a novel avenue for refining link prediction performance, reframed through the lens of optimizing negative sample selection.

Bibliography

[1] Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Ver Steeg, G., and Galstyan, A. (2019). Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR.

[2] Adamic, L. A. and Adar, E. (2003). Friends and neighbors on the web. *Social networks*, 25(3):211–230.

[3] Ai, B., Qin, Z., Shen, W., and Li, Y. (2022). Structure enhanced graph neural networks for link prediction. *arXiv preprint arXiv:2201.05293*.

[4] Allamanis, M. (2019). The adverse effects of code duplication in machine learning models of code. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pages 143–153.

[5] Alon, U., Brody, S., Levy, O., and Yahav, E. (2018). code2seq: Generating sequences from structured representations of code. *arXiv preprint arXiv:1808.01400*.

[6] Alon, U., Zilberstein, M., Levy, O., and Yahav, E. (2019). code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29.

[7] Altmann, A., Toloşi, L., Sander, O., and Lengauer, T. (2010). Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347.

[8] Aytar, Y., Vondrick, C., and Torralba, A. (2016). Soundnet: Learning sound representations from unlabeled video. *Advances in neural information processing systems*, 29.

[9] Backstrom, L. and Leskovec, J. (2011). Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 635–644.

[10] Barceló, P., Geerts, F., Reutter, J., and Ryschkov, M. (2021). Graph neural networks with local graph parameters. In *Advances in Neural Information Processing Systems*, volume 34, pages 25280–25293.

[11] Barton, N. A., Farewell, T. S., Hallett, S. H., and Acland, T. F. (2019). Improving pipe failure predictions: Factors affecting pipe failure in drinking water networks. *Water research*, 164:114926.

[12] Bengio, Y. et al. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127.

[13] Bouritsas, G., Frasca, F., Zafeiriou, S., and Bronstein, M. M. (2022). Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668.

[14] Brody, S., Alon, U., and Yahav, E. (2021). How attentive are graph attention networks? In *International Conference on Learning Representations*.

[15] Brody, S., Alon, U., and Yahav, E. (2022). How attentive are graph attention networks? In *International Conference on Learning Representations*.

[16] Cai, L., Li, J., Wang, J., and Ji, S. (2021). Line graph neural networks for link prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[17] Cao, S., Lu, W., and Xu, Q. (2015). Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900.

[18] Cao, S., Lu, W., and Xu, Q. (2016). Deep neural networks for learning graph representations. In *Thirtieth AAAI conference on artificial intelligence*.

[19] Chen, D., O'Bray, L., and Borgwardt, K. (2022a). Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, pages 3469–3489. PMLR.

[20] Chen, G., Wang, H., Fang, Y., and Jiang, L. (2022b). Link prediction by deep non-negative matrix factorization. *Expert Systems with Applications*, 188:115991.

[21] Chen, J., Ma, T., and Xiao, C. (2018). FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*.

[22] Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. (2020a). Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR.

[23] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

[24] Chen, Z., Chen, L., Villar, S., and Bruna, J. (2020b). Can graph neural networks count substructures? *Advances in neural information processing systems*, 33:10383–10395.

[25] Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. (2019). Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266.

[26] Consortium, G. O. (2019). The gene ontology resource: 20 years and still going strong. *Nucleic acids research*, 47(D1):D330–D338.

[27] Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. (2020). Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271.

[28] De Las Rivas, J. and Fontanillo, C. (2010). Protein–protein interactions essentials: key concepts to building and analyzing interactome networks. *PLoS computational biology*, 6(6):e1000807.

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

[30] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

[31] Duchesne, S., Beardsell, G., Villeneuve, J.-P., Toumbou, B., and Bouchard, K. (2013). A survival analysis model for sewer pipe structural deterioration. *Computer-Aided Civil and Infrastructure Engineering*, 28(2):146–160.

[32] Dwivedi, V. P. and Bresson, X. (2020). A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*.

[33] Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. (2020). Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*.

[34] Farmani, R., Kakoudakis, K., Behzadian, K., and Butler, D. (2017). Pipe failure prediction in water distribution systems considering static and dynamic factors. *Procedia Engineering*, 186:117–126.

[35] Feng, J., Chen, Y., Li, F., Sarkar, A., and Zhang, M. (2022). How powerful are k-hop message passing graph neural networks. In *Advances in Neural Information Processing Systems*, volume 35, pages 4776–4790.

[36] Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[37] Galkin, M., Wu, J., Denis, E., and Hamilton, W. L. (2021). Nodepiece: Compositional and parameter-efficient representations of large knowledge graphs. *arXiv preprint arXiv:2106.12144*.

[38] Gao, H. and Ji, S. (2019). Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR.

[39] Geerts, F. and Reutter, J. L. (2021). Expressiveness and approximation properties of graph neural networks. In *International Conference on Learning Representations*.

[40] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR.

[41] Giraldo-González, M. M. and Rodríguez, J. P. (2020). Comparison of statistical and machine learning models for pipe failure modeling in water distribution networks. *Water*, 12(4):1153.

[42] Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE.

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

[44] Guo, Z., Shiao, W., Zhang, S., Liu, Y., Chawla, N., Shah, N., and Zhao, T. (2022). Linkless link prediction via relational distillation. *arXiv preprint arXiv:2210.05801*.

[45] Hagberg, A., Swart, P., and S Chult, D. (2008). Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[46] Hamilton, W., Ying, Z., and Leskovec, J. (2017a). Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034.

[47] Hamilton, W., Ying, Z., and Leskovec, J. (2017b). Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034.

[48] Hawkes, A. G. (1971). Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90.

[49] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[50] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97.

[51] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

[52] Hoang, V. T., Jeon, H.-J., You, E.-S., Yoon, Y., Jung, S., and Lee, O.-J. (2023). Graph representation learning and its applications: A survey. *Sensors*, 23(8):4168.

[53] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[54] Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. (2020). Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133.

[55] Hua, C., Rabusseau, G., and Tang, J. (2022). High-order pooling for graph neural networks with tensor decomposition. In *Advances in Neural Information Processing Systems*, volume 35, pages 6021–6033.

[56] Huang, S., Bao, Z., Li, G., Zhou, Y., and Culpepper, J. S. (2020). Temporal network representation learning via historical neighborhoods aggregation. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1117–1128. IEEE.

[57] Isham, V. and Westcott, M. (1979). A self-correcting point process. *Stochastic processes and their applications*, 8(3):335–347.

[58] Ishwaran, H., Kogalur, U. B., Blackstone, E. H., Lauer, M. S., et al. (2008). Random survival forests. *Annals of Applied Statistics*, 2(3):841–860.

[59] Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579.

[60] Jeh, G. and Widom, J. (2002). Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543.

[61] Karypis, G. and Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392.

[62] Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43.

[63] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[64] Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

[65] Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

[66] Kovács, I. A., Luck, K., Spirohn, K., Wang, Y., Pollis, C., Schlabach, S., Bian, W., Kim, D.-K., Kishore, N., Hao, T., et al. (2019). Network-based prediction of protein interactions. *Nature communications*, 10(1):1–8.

[67] Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. (2021). Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34.

[68] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

[69] Kuang, W., Zhen, W., Li, Y., Wei, Z., and Ding, B. (2021). Coarformer: Transformer for large graph via graph coarsening.

[70] Kumar, A., Rizvi, S. A. A., Brooks, B., Vanderveld, R. A., Wilson, K. H., Kenney, C., Edelstein, S., Finch, A., Maxwell, A., Zuckerbraun, J., et al. (2018a). Using machine learning to assess the risk of and prevent water main breaks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 472–480.

[71] Kumar, S., Zhang, X., and Leskovec, J. (2018b). Learning dynamic embeddings from temporal interaction networks. *Learning*, 17:29.

[72] Landrum, G. (2006). Rdkit: Open-source cheminformatics. 2006. *Google Scholar*.

[73] Langville, A. N. and Meyer, C. D. (2011). Google's pagerank and beyond. In *Google's PageRank and Beyond*. Princeton university press.

[74] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[75] Li, G., Xiong, C., Thabet, A., and Ghanem, B. (2020a). Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*.

[76] Li, J., Zhu, X., and Chen, J. Y. (2009). Building disease-specific drug-protein connectivity maps from molecular interaction networks and pubmed abstracts. *PLoS computational biology*, 5(7):e1000450.

[77] Li, P., Wang, Y., Wang, H., and Leskovec, J. (2020b). Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems*, 33:4465–4478.

[78] Li, Q., Han, Z., and Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*.

[79] Li, Z. and Arora, S. (2020). An exponential learning rate schedule for deep learning. In *International Conference on Learning Representations*.

[80] Li, Z., Zhang, B., Wang, Y., Chen, F., Taib, R., Whiffin, V., and Wang, Y. (2014). Water pipe condition assessment: a hierarchical beta process approach for sparse incident data. *Machine learning*, 95(1):11–26.

[81] Liang, B., Li, Z., Wang, Y., and Chen, F. (2018). Long-term rnn: Predicting hazard function for proactive maintenance of water mains. In *Proceedings of the 27th acm international conference on information and knowledge management*, pages 1687–1690.

[82] Liben-Nowell, D. and Kleinberg, J. (2007). The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031.

[83] Lin, P., Zhang, B., Wang, Y., Li, Z., Li, B., Wang, Y., and Chen, F. (2015). Data driven water pipe failure prediction: A bayesian nonparametric approach. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 193–202.

[84] Lin, X., Quan, Z., Wang, Z.-J., Ma, T., and Zeng, X. (2020). Kgnn: Knowledge graph neural network for drug-drug interaction prediction. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20 (International Joint Conferences on Artificial Intelligence Organization)*, pages 2739–2745.

[85] Ling, C. X., Huang, J., and Zhang, H. (2003). Auc: a better measure than accuracy in comparing learning algorithms. In *Conference of the canadian society for computational studies of intelligence*, pages 329–341. Springer.

[86] Lipton, Z. C., Berkowitz, J., and Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*.

[87] Liu, S., Chen, L., Dong, H., Wang, Z., Wu, D., and Huang, Z. (2019). Higher-order weighted graph convolutional networks. *arXiv preprint arXiv:1911.04129*.

[88] Liu, X., Ding, J., Jin, W., Xu, H., Ma, Y., Liu, Z., and Tang, J. (2021). Graph neural networks with adaptive residual. In *Advances in Neural Information Processing Systems*, volume 34, pages 9720–9733.

[89] Liu, Z., Chen, C., Yang, X., Zhou, J., Li, X., and Song, L. (2018). Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 2077–2085.

[90] Mahdavi, S., Khoshraftar, S., and An, A. (2018). dynnode2vec: Scalable dynamic network embedding. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3762–3765. IEEE.

[91] Manessi, F., Rozza, A., and Manzo, M. (2020). Dynamic graph convolutional networks. *Pattern Recognition*, 97:107000.

[92] Martínez, V., Berzal, F., and Cubero, J.-C. (2016). A survey of link prediction in complex networks. *ACM computing surveys (CSUR)*, 49(4):1–33.

[93] Menon, A. K. and Elkan, C. (2011). Link prediction via matrix factorization. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 437–452. Springer.

[94] Mialon, G., Chen, D., Selosse, M., and Mairal, J. (2021). Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*.

[95] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

[96] Munikoti, S., Agarwal, D., Das, L., Halappanavar, M., and Natarajan, B. (2023). Challenges and opportunities in deep reinforcement learning with graph neural networks: A comprehensive review of algorithms and applications. *IEEE Transactions on Neural Networks and Learning Systems*.

[97] Nguyen, G. H., Lee, J. B., Rossi, R. A., Ahmed, N. K., Koh, E., and Kim, S. (2018). Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*, pages 969–976.

[98] Obradović, D. (2017). The impact of tree root systems on wastewater pipes. *Zbornik radova, Zajednički temelji*, pages 65–71.

[99] Pan, L., Shi, C., and Dokmanić, I. (2021). Neural link prediction with walk pooling. *arXiv preprint arXiv:2110.04375*.

[100] Pandhre, S., Mittal, H., Gupta, M., and Balasubramanian, V. N. (2018). Stwalk: learning trajectory representations in temporal graphs. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pages 210–219.

[101] Park, S., Jun, H., Agbenowosi, N., Kim, B. J., and Lim, K. (2011). The proportional hazards modeling of water main failure data incorporating the time-dependent effects of covariates. *Water resources management*, 25(1):1–19.

[102] Park, W., Chang, W.-G., Lee, D., Kim, J., et al. (2022). Grpe: Relative positional encoding for graph transformer. In *ICLR2022 Machine Learning for Drug Discovery*.

[103] Penrose, M. et al. (2003). *Random geometric graphs*, volume 5. Oxford university press.

[104] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710.

[105] Pietrucha-Urbanik, K. (2015). Failure analysis and assessment on the exemplary water supply network. *Engineering failure analysis*, 57:137–142.

[106] Puny, O., Ben-Hamu, H., and Lipman, Y. (2020). Global attention improves graph networks generalization. *arXiv preprint arXiv:2006.07846*.

[107] Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., and Tang, J. (2018). Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 459–467.

[108] Randrup, T. B., McPherson, E. G., and Costello, L. R. (2001). Tree root intrusion in sewer systems: review of extent and costs. *Journal of Infrastructure Systems*, 7(1):26–31.

[109] Robles-Velasco, A., Cortés, P., Muñuzuri, J., and Onieva, L. (2020). Prediction of pipe failures in water supply networks using logistic regression and support vector classification. *Reliability Engineering & System Safety*, 196:106754.

[110] Rong, Y., Bian, Y., Xu, T., Xie, W., Wei, Y., Huang, W., and Huang, J. (2020). Self-supervised graph transformer on large-scale molecular data. *Advances in Neural Information Processing Systems*, 33:12559–12571.

[111] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer.

[112] Roy, I., De, A., and Chakrabarti, S. (2021). Adversarial permutation guided node representations for link prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 9445–9453.

[113] Sankar, A., Wu, Y., Gou, L., Zhang, W., and Yang, H. (2018). Dynamic graph representation learning via self-attention networks. *arXiv preprint arXiv:1812.09430*.

[114] Schlichtkrull, M., Kipf, T. N., Bloem, P., Berg, R. v. d., Titov, I., and Welling, M. (2018). Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer.

[115] Schmitt, C., Pluvinage, G., Hadj-Taieb, E., and Akid, R. (2006). Water pipeline failure due to water hammer effects. *Fatigue & Fracture of Engineering Materials & Structures*, 29(12):1075–1082.

[116] Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. (2008). Collective classification in network data. *AI magazine*, 29(3):93–93.

[117] Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.

[118] Seo, Y., Defferrard, M., Vandergheynst, P., and Bresson, X. (2018). Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer.

[119] Shamir, U. and Howard, C. D. (1979). An analytic approach to scheduling pipe replacement. *Journal-American Water Works Association*, 71(5):248–258.

[120] Shang, Y., Hao, Z., Yao, C., and Li, G. (2022). Improving graph neural network models in link prediction task via a policy-based training method. *Applied Sciences*, 13(1):297.

[121] Shaw, P., Uszkoreit, J., and Vaswani, A. (2018). Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468.

[122] Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. (2018). Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*.

[123] Shi, W. and Rajkumar, R. (2020). Point-gnn: Graph neural network for 3d object detection in a point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1711–1719.

[124] Singh, A., Huang, Q., Huang, S. L., Bhalerao, O., He, H., Lim, S.-N., and Benson, A. R. (2021). Edge proposal sets for link prediction. *arXiv preprint arXiv:2106.15810*.

[125] Snider, B. and McBean, E. A. (2020). Improving urban water security through pipe-break prediction models: Machine learning or survival analysis. *Journal of Environmental Engineering*, 146(3):04019129.

[126] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

[127] Song, X., Ma, R., Li, J., Zhang, M., and Wipf, D. P. (2021). Network in graph neural network. *CoRR*, abs/2111.11638.

[128] Sun, C. and Wu, G. (2020). Adaptive graph diffusion networks with hop-wise attention. *arXiv preprint arXiv:2012.15024*.

[129] Szklarczyk, D., Gable, A. L., Lyon, D., Junge, A., Wyder, S., Huerta-Cepas, J., Simonovic, M., Doncheva, N. T., Morris, J. H., Bork, P., et al. (2019). String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic acids research*, 47(D1):D607–D613.

[130] Taubin, G. (1995). A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358.

[131] Teru, K., Denis, E., and Hamilton, W. (2020). Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*, pages 9448–9457. PMLR.

[132] Thekumparampil, K. K., Wang, C., Oh, S., and Li, L.-J. (2018). Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*.

[133] Tran, P. V. (2018). Learning to make predictions on graphs with autoencoders. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 237–245. IEEE.

[134] Trivedi, R., Farajtabar, M., Biswal, P., and Zha, H. (2018). Representation learning over dynamic graphs. *arXiv preprint arXiv:1803.04051*.

[135] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[136] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018a). Graph attention networks. In *International Conference on Learning Representations*.

[137] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018b). Graph attention networks. In *International Conference on Learning Representations*.

[138] Vincent-Cuaz, C., Flamary, R., Corneli, M., Vayer, T., and Courty, N. (2022). Template based graph neural network with optimal transport distances. In *Advances in Neural Information Processing Systems*, volume 35, pages 11800–11814.

[139] Wang, D., Cui, P., and Zhu, W. (2016). Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234.

[140] Wang, H., Yin, H., Zhang, M., and Li, P. (2022). Equivariant and stable positional encoding for more powerful graph neural networks. In *International Conference on Learning Representations*.

[141] Wang, K., Shen, Z., Huang, C., Wu, C.-H., Dong, Y., and Kanakia, A. (2020). Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413.

[142] Wang, Z., Zhou, Y., Hong, L., Zou, Y., and Su, H. (2021). Pairwise learning for neural link prediction. *arXiv preprint arXiv:2112.02936*.

[143] Weisfeiler, B. and Leman, A. (1968). The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, 2(9):12–16.

[144] Wishart, D. S., Feunang, Y. D., Guo, A. C., Lo, E. J., Marcu, A., Grant, J. R., Sajed, T., Johnson, D., Li, C., Sayeeda, Z., et al. (2018). Drugbank 5.0: a major update to the drugbank database for 2018. *Nucleic acids research*, 46(D1):D1074–D1082.

[145] Wu, E., Cui, H., and Chen, Z. (2022). Relpnet: Relation-based link prediction neural network. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 2138–2147.

[146] Wu, L., Chen, Y., Shen, K., Guo, X., Gao, H., Li, S., Pei, J., Long, B., et al. (2023). Graph neural networks for natural language processing: A survey. *Foundations and Trends® in Machine Learning*, 16(2):119–328.

[147] Wu, Z., Jain, P., Wright, M., Mirhoseini, A., Gonzalez, J. E., and Stoica, I. (2021). Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems*, 34:13266–13279.

[148] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24.

[149] Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. (2018). Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530.

[150] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018a). How powerful are graph neural networks? In *International Conference on Learning Representations*.

[151] Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. (2018b). Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pages 5453–5462. PMLR.

[152] Yan, J., Wang, Y., Zhou, K., Huang, J., Tian, C., Zha, H., and Dong, W. (2013). Towards effective prioritizing water pipe replacement and rehabilitation. In *Twenty-third international joint conference on artificial intelligence*. Citeseer.

[153] Yao, X., Shao, Y., Cui, B., and Chen, L. (2021). Uninet: Scalable network representation learning with metropolis-hastings sampling. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 516–527. IEEE.

[154] Yin, H., Zhang, M., Wang, Y., Wang, J., and Li, P. (2022). Algorithm and system co-design for efficient subgraph-based graph representation learning. *arXiv preprint arXiv:2202.13538*.

[155] Yin, Y., Zhang, J., Pei, Y., Cheng, X., and Ji, L. (2019). Mhdne: Network embedding based on multivariate hawkes process. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 409–421. Springer.

[156] Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. (2021). Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34.

[157] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983.

[158] You, J., Gomes-Selman, J. M., Ying, R., and Leskovec, J. (2021). Identity-aware graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10737–10745.

[159] Yun, S., Jeong, M., Kim, R., Kang, J., and Kim, H. J. (2019). Graph transformer networks. *Advances in neural information processing systems*, 32.

[160] Yun, S., Kim, S., Lee, J., Kang, J., and Kim, H. J. (2021a). Neo-gnns: Neighborhood overlap-aware graph neural networks for link prediction. In *Advances in Neural Information Processing Systems*, volume 34, pages 13683–13694.

[161] Yun, S., Kim, S., Lee, J., Kang, J., and Kim, H. J. (2021b). Neo-gnns: Neighborhood overlap-aware graph neural networks for link prediction. *Advances in Neural Information Processing Systems*, 34:13683–13694.

[162] Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473.

[163] Zeng, H., Zhang, M., Xia, Y., Srivastava, A., Malevich, A., Kannan, R., Prasanna, V., Jin, L., and Chen, R. (2021). Decoupling the depth and scope of graph neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pages 19665–19679.

[164] Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. (2020). Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*.

[165] Zhang, C., Ren, M., and Urtasun, R. (2019). Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations*.

[166] Zhang, J., Shi, X., Xie, J., Ma, H., King, I., and Yeung, D. Y. (2018a). Gaan: Gated attention networks for learning on large and spatiotemporal graphs. In *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*.

[167] Zhang, J., Zhang, H., Xia, C., and Sun, L. (2020). Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*.

[168] Zhang, M. and Chen, Y. (2018). Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31:5165–5175.

[169] Zhang, M., Cui, Z., Neumann, M., and Chen, Y. (2018b). An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

[170] Zhang, M., Li, P., Xia, Y., Wang, K., and Jin, L. (2021). Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34:9061–9073.

[171] Zhao, T., Liu, G., Wang, D., Yu, W., and Jiang, M. (2022). Learning from counterfactual links for link prediction. In *International Conference on Machine Learning*, pages 26911–26926. PMLR.

[172] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2018). Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*.

[173] Zhou, J., Liu, L., Wei, W., and Fan, J. (2022). Network representation learning: from preprocessing, feature extraction to node embedding. *ACM Computing Surveys (CSUR)*, 55(2):1–35.

[174] Zhou, T., Lü, L., and Zhang, Y.-C. (2009). Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630.

[175] Zhu, X., Ghahramani, Z., and Lafferty, J. D. (2003). Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919.

[176] Zhu, Z., Zhang, Z., Xhonneux, L.-P., and Tang, J. (2021a). Neural bellman-ford networks: A general graph neural network framework for link prediction. In *Advances in Neural Information Processing Systems*, volume 34, pages 29476–29490.

[177] Zhu, Z., Zhang, Z., Xhonneux, L.-P., and Tang, J. (2021b). Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems*, 34.

[178] Zuo, Y., Liu, G., Lin, H., Guo, J., Hu, X., and Wu, J. (2018). Embedding temporal network via neighborhood formation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2857–2866.