Designing Machine Learning Models for Graph Analytics

by

Hanchen Wang

A THESIS SUBMITTED IN FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Australian Artificial Intelligence Institute (AAII)
Faculty of Engineering and Information Technology (FEIT)
University of Technology Sydney (UTS)

June, 2021

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, Hanchen Wang declare that this thesis, is submitted in fulfilment 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: 15/06/2021

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to my principal supervisor Prof. Ying Zhang. I have benefited greatly from his constant support, knowledge, guidance and continuous encouragement. He is not only my valued mentor, but also a true friend. As a friend, he is always ready to help me. Whenever I feel frustrated, he is ready to offer advice and practical support. I cannot be more grateful for the incredible advice and support he has given me on my research and career.

Secondly, I would like to express my greatest gratitude to Prof. Defu Lian and my co-supervisor Lu Qin. They always provides insightful advice on my research topics. Their guidance significantly extended my knowledge and helped me pursue a correct research direction all the time. I would like to thank Prof. Xuemin Lin, Prof. Wenjie Zhang and Prof. Wei Wang for their support, they serve as examples of excellent researchers for me and share invaluable ideas and experiences with me.

Thirdly, I would like to acknowledge Prof. Xiangjian He, Prof. Xiaoyang Wang, Dr. Wanqi Liu, Dr. Dong Wen, Dr. Yiguang Lin, Dr. Chen Chen, Mr. Peilun Yang , Mr. Jianke Yu and Mr. Han Chen for supporting my works during my Ph.D. study. I thank Dr. Wanqi Liu for sharing her enjoyment of conducting research and her rich experience in research and coding. I also thank Mr. Peilun

Yang for the wonderful ideas and inspirations he gave me in our discussions.

My special thanks go to the following people: Prof. Long Yuan, Prof. Jianye Yang, Dr. Xin Cao, Prof. Fan Zhang, Prof. Yixiang Fang, Prof. Zengfeng Huang, Prof. Shiyu Yang, Prof. Peng Cheng, Prof. Weiwei Liu, Dr. Xubo Wang, Dr. Dian Ouyang, Dr. Longbin Lai, Dr. Xing Feng, Dr. Haida Zhang, Dr. Fei Bi, Dr. Shenlu Wang, Dr. Wei Li, Dr. Chen Zhang, Dr. Yang Yang, Dr. Kai Wang, Dr. You Peng, Dr. Boge Liu, Dr. Wentao Li, Dr. Conggai Li, Dr. Mingjie Li, Dr. Xuefeng Chen, Dr. Bing Li, Dr. Shuiqiao Yang, Dr. Jiaojiao Jiang, Mr. Yu Hao, Mr. Yuren Mao, Ms. Xiaoshuang Chen, Mr. Yuanhang Yu, Mr. Yixing Yang, Mr. Zhengyi Yang, Mr. Qingyuan Linghu, Mr. Michael Ruisi Yu, Mr. Chenji Huang, Mr. Bohua Yang, Mr. Jiahui Yang, Mr. Yilun Huang, Mr. Rong Hu, Mr. Junhua Zhang, Mr. Yuxuan Qiu, Mr. Kongzhang Hao, Mr. Yizhang He, Mr. Shunyang Li, Mr. Gengda Zhao, Mr. Qingshuai Feng, Mr. Zhuo Ma, Ms. Yuting Zhang. The days we spent together bear unforgettable memories.

Finally, I would also like to acknowledge my father Mr. Yongming Wang, and my mother Ms. Jing Chen, who provide me with selfless love and endless encouragement. I also thank all my relatives and friends for their love and support. Thanks to my wife Ms. Yufan Chen for her timely encouragement and inspiration in my hard times and bringing me joyful life.

Abstract

With growing popularity of the machine learning methods, there have been a great number of machine learning methods proposed for graph analytics. In this thesis, we design three machine learning based models for the popular graph analysis tasks such as node classification, graph representation learning, graph interaction prediction and subgraph matching.

Firstly, we design a binarized graph neural network to efficiently obtain the vector representations for vertices and graphs. Recently, there have been some breakthroughs in graph analysis by applying the Graph Neural Networks (GNNs) following a neighborhood aggregation scheme, which demonstrate outstanding performance in many tasks. However, we observe that the parameters of the network and the embedding of nodes are represented in real-valued matrices in existing GNN-based graph embedding approaches which may limit the efficiency and scalability of these models. It is well-known that binary vector is usually much more space and time efficient than the real-valued vector. This motivates us to develop a binarized graph neural network to learn the binary representations of the nodes with binary network parameters following the GNN-based paradigm. Our proposed method can be seamlessly integrated into the existing GNN-based embedding approaches to binarize the model parameters and learn the compact embedding. Extensive experiments indicate that the proposed binarized graph neural network, namely BGN, is orders of magnitude more efficient in terms of both time and space while matching the state-of-the-art performance.

Secondly, we first design a graph of graphs neural network for entity interaction prediction, and then extend the model to support the graph classification task with more expressive representations. Entity interaction prediction is essential in many important applications such as chemistry, biology, material science, and medical science. The problem becomes quite challenging when each entity is

represented by a complex structure, namely structured entity, because two types of graphs are involved: local graphs for structured entities and a global graph to capture the interactions between structured entities. We observe that existing works on structured entity interaction prediction cannot properly exploit the unique graph of graphs structure. In this thesis, we propose a Graph of Graphs Neural Network, namely GoGNN, which extracts the features in both structured entity graphs and the entity interaction graph in a hierarchical way. We also propose the dual-attention mechanism that enables the model to preserve the neighbor importance in both levels of graphs. Based on GoGNN, we further propose a Powerful Graph Of graphs neural Network, namely PGON, which has 3-Weisfeiler-Lehman expressive power and captures the attributes and structural information from both structured entity graphs and entity interaction graph hierarchically. Extensive experiments are conducted on real-world datasets, which show the superior performance of GoGNN and PGON compared to other state-of-the-art methods on both graph classification and graph interaction prediction tasks.

Thirdly, we design a reinforcement learning based query vertex ordering model for subgraph matching. Subgraph matching is a fundamental problem in various fields that use graph structured data. Subgraph matching algorithms enumerate all isomorphic embeddings of a query graph q in a data graph G. We apply the Reinforcement Learning (RL) and Graph Neural Networks (GNNs) techniques to generate the high-quality matching order for subgraph matching algorithms. Instead of using the fixed heuristics to generate the matching order, our model could capture and make full use of the graph information, and thus determine the query vertex order with the adaptive learning-based rule that could significantly reduce the number of redundant enumerations. With the help of the reinforcement learning framework, our model is able to consider the long-term

benefits rather than only consider the local information at current step.

PUBLICATIONS

- **Hanchen Wang**, Defu Lian, Ying Zhang, Lu Qin, Xiangjian He, Yiguang Lin, and Xuemin Lin."Binarized graph neural network." *World Wide Web Journal (2021).* (**Chapter 3**)
- **Hanchen Wang**, Defu Lian, Ying Zhang, Lu Qin, Xuemin Lin. "GoGNN: Graph of Graphs Neural Network for Predicting Structured Entity Interactions." *In the Proceedings of 29th International Joint Conference on Artificial Intelligence. (IJCAI 2020)* (**Chapter 4**)
- **Hanchen Wang**, Defu Lian, Wanqi Liu, Dong Wen, Chen Chen, Xiaoyang Wang. "Powerful Graph of Graphs Neural Network for Graph Classification" *World Wide Web Journal (2021).* (**Chapter 4**)
- **Hanchen Wang**, Ying Zhang, Lu Qin, Wei Wang, Wenjie Zhang. "Reinforcement Learning Based Query Vertex Ordering Model for Backtracking Based Subgraph Matching" *In Submission* (**Chapter 5**)

Contents

List of Figures

Chapter 1

Introduction

With the growth of graph structured data, graph analytics has attracted great attention in both academia and industry. As a fundamental research problem, graph analytics is widely used in numerous important areas such as database management system, social network, biology, medical science and traffic network. Thanks to the advancement of machine learning methods, we can applied the learning-based techniques to build novel graph analysis methods which enjoy better efficiency and accuracy. This thesis focuses on designing machine learning methods for graph analytics. In order to improve the accuracy and efficiency of the classification and link prediction problems, we propose the Binarized Graph Neual Network, Graph of Graph Neural network and it extension in this thesis. We also proposed a novel learning-based model for the subgraph matching task to generate the order plan to eventually reduce the time cost of subgraph matching query. This thesis proposes the following three models. Firstly, we designed a binarized graph neural network to learn binary representations for graphs, which improve the efficiency of the graph neural networks. Secondly, we proposed machine learning models for the graph of graphs for entity interaction prediction and graph classification tasks. Thirdly, we developed a reinforcement

learning based query vertex ordering methods to accelerate the query processing procedure of subgraph matching. This thesis provides the effective and efficient solution to the most popular graph analytic problems: node classification and link prediction. Besides, we also tackle the problem of subgraph matching to explore the exploitation of the machine learning methods in the graph analytic area. Next, we detailed the background, motivations and contributions for each proposed model.

1.1 Binarized Graph Neural Network

Graph analysis provides powerful insights into how to unlock the value graphs hold. Due to this power, techniques for analyzing graphs are becoming an increasingly popular topic of study in both academics and industry. To effectively and efficiently support important analytic tasks on graph data, such as node/graph classification, node clustering, community detection, node recommendation, link prediction and graph visualization, a variety of graph embedding techniques (See [35, 17] for a comprehensive survey) have been developed. Graph data is mapped into low-dimension data such that the proximity relationship among graph nodes (i.e., objects) is preserved and the off-the-shelf machine learning methods, which are designed to handle vector representations, can be immediately applied.

The existing graph embedding techniques can be roughly classified into three broad categories: (1) random walk based embedding (e.g., Deepwalk [78] and Node2vec [33]) ; (2) node similarity based embedding (e.g., LINE [98] and NetMF [81]); and (3) graph neural networks (GNN) based embedding (e.g., GCN [46], GraphSage [34], GAT [100] and AS-GCN [41]). As reported by

Leskovec et al. in their tutorial on graph embedding at WWW 2018[1], the first two categories of embedding techniques are only able to learn a "shallow" representation of the graph nodes due to the simplicity of the models. It is shown in [46, 34] that the neural network based embedding methods significantly outperform the state-of-the-art techniques in the first two categories for the node classification task. Therefore, exploring how to use neural network to create a "deep" representation more efficiently is a promising direction in graph representation learning. However, most of the existing graph neural network models suffer from the scalability issue due to the high time and space cost of the real-valued model.

Recently, there have been some researches on learning binary graph embedding (e.g., [64, 93, 113]), in which each node is represented by a binary vector (code), instead of a real-valued vector. It has been shown that the binarized graph embedding can achieve much better time and space efficiency.

Time efficiency. It is well-known that the distance computation of binary vectors (i.e., Hamming distance) is much more efficient than that of real-valued vectors (e.g., Euclidian distance). In addition to the specifically tailored search algorithms (e.g., [80]), the dot product between binary vectors can also enjoy the hardware support (e.g., *xnor* and build-in CPU instruction *popcount*).

As stressed in a recent work [58] from *DeepMind*, the pairwise dot product of the vectors has been intensively used by the model for some specific tasks (e.g., graph similarity computation in [2]). Thus, the binary vector has been used in their graph matching network (GMN) to speedup the computation.

Space Efficiency. The binary embedding can represent the node in a compact way while well preserving the structure information. As shown in [64], INH-MF can achieve competitive graph node classification performance with 128 bits for

[1] `http://snap.stanford.edu/proj/embeddings-www`

each node compared to the conventional embedding approaches (e.g., DeepWalk) with 128 dimensions (i.e., 128×64 bits) per node. This will be a great advantage when we face a large-scale graph because the binarized embedding of a graph is more likely to be accommodated in the main memory.

The existing GNN-based methods have demonstrated outstanding performance in various tasks such as node classification [34, 46, 100, 41], link prediction [115, 44], graph similarity match [2, 58] and graph clustering [102, 118]. However, they may suffer from the limitation of the memory and speed due to the use of real-valued vectors for node and graph representations and model parameters.

Given the outstanding embedding quality, various applications of the GNN-based approaches and the space and time efficiency of the binarized representation, one may wonder if we can design a binarized GNN-based graph embedding approach such that we can achieve a good trade-off between embedding quality and time/space efficiency in the GNN-based methods.

We notice that the existing binarized graph embedding methods [64, 93] rely on the discretization of the matrix factorization following the node-similarity based approaches. They cannot be extended to binarize the GNN-based embedding due to the inherently different natures of two categories of approaches.

As to our best knowledge, the only attempt for the binarization of GNN is from *DeepMind* in their recent work [58]. Their binarization method converts each learned d-dimensional real-valued vector into a d-dimensional "nearly" binary vector by applying well-known binarization function *tanh* to approximate hamming distance for the binarization and optimization. However, the output of *tanh* is not exact binary value and cannot be accelerated by the binary logic operations (e.g., xnor and popcount). As an alternative, one may consider the Binarized Neural Network (BNN) (e.g., [42]) for the graph embedding so that the

representation is naturally binarized. However, BNN is not designed for graph data, and as to our best knowledge, there is no existing graph embedding work based on BNN.

These issues motivate us to develop a new binarized graph embedding technique which can be integrated into existing GNN-based models to binarize the parameters and produce high-quality binarized graph embeddings. The key challenge is how to generate compact embedding vectors with binary network parameters in an effective way. To address the challenge, we design a binarized graph neural network framework to learn the binary parameters and representations efficiently and effectively.

Contributions. The principle contributions of our proposed binarized graph neural network are summarized as follows:

- To the best of our knowledge, this is the first study on binarized graph neural network (GNN) with binary parameters to generate binary graph representations. The proposed method, namely **BGN**, can be seamlessly integrated into the existing GNNs.

- An end-to-end binarized graph neural network framework is proposed with binary weights and activations. This binarized framework can immediately reduce the memory consumption for the network; the bit-wise operations between the binary vectors can substantially speedup the inference time of the model and the gradient estimator enables our model to effectively process back-propagation through discrete parameters and activations.

- Extensive experiments on multiple benchmark networks are conducted for node classification task. The results demonstrate that our proposed method outperforms existing binarized embedding methods with a big margin. Compare to the real-valued GNNs, our BGN model can achieve nearly

state-of-the-art performance while consuming much fewer computation resources (up to 1/28 parameter and embedding memory space and 1/20 inference time).

- Binarization approaches are employed on the GNN-based application GMN to show that, by applying our BGN techniques, GMN model can dramatically reduce the time and space complexity while keeping the performance competitiveness.

- Experiments further show that our proposed BGN technique allows users to achieve a trade-off between the space/time and embedding quality in a flexible way by tuning different level and setting of binarization on the parameters and activations.

1.2 Graph of Graphs Neural Network

Interactions between the structured entities like chemicals are the basis of many applications such as chemistry, biology, material science, medical science, and environmental science. For example, the knowledge of chemical interactions is a helpful guide for the toxicity prediction, new material design and pollutant removal [111]. In medical science, understanding the interaction between drugs is vital for drug discovery and side effect prediction which can save millions of lives every year [72].

One immediate way to investigate the interactions between two structured entities is to conduct experiments for them in the laboratory or clinics. However, due to the enormous number of structured entities, it is infeasible in terms of both time and resource to examine all possible interactions. Thanks to the advances in the computational approaches for the structured entity interaction prediction, a variety of techniques have been proposed to predict the interactions

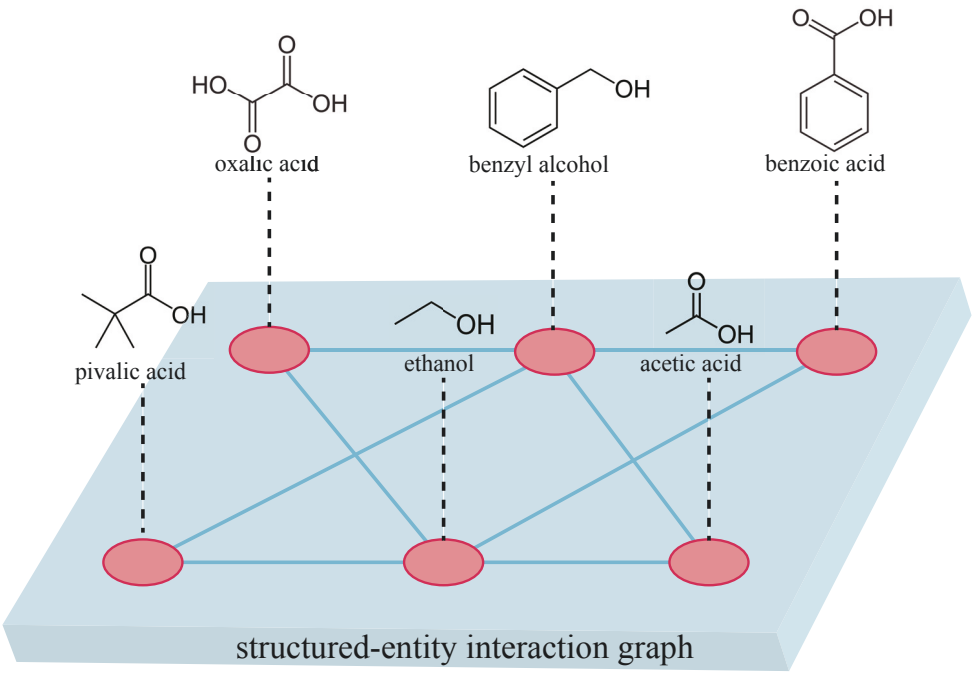


Figure 1.1: Interaction graph of molecule graphs.

among structured entities effectively and efficiently by utilizing the deep neural network or graph neural network (GNN) techniques such as DeepCCI [49] for chemical-chemical interaction prediction and DeepDDI [85] for drug-drug interaction prediction.

We observe that the interactions among structured entities can be naturally modeled by the graph-of-graphs (a.k.a. network-of-networks) where each structured entity is a *local* graph, and the interactions of the entities form a *global* graph. In Figure 1.1, we take the chemical-chemical interactions as an example. Each chemical molecule is a structured entity and can be represented by a *local* graph (i.e., molecule graph) where nodes represent atoms and the bonds among the atoms are the edges. On the other hand, the interactions (edges) among the

structured entities (nodes) form a *global* graph. However, the existing studies for structured entity interaction prediction do not make full use of the graph-of-graphs model and only consider partial information. For instance, MR-GNN [111] only considers the local structure information of entities and their pairwise similarity; Decagon [121] focuses on the interaction graph and only treats the structured entity as a simple node. Other works such as DeepCCI and DeepDDI even do not consider the graph structure information.

These limitations motivate us to develop a new approach to fully exploit the graph-of-graphs (GoG) model to predict the structured entity interactions. In particular, we propose a novel model called Graph of Graphs Neural Network(**GoGNN**). Our model builds a graph neural network with attention-based pooling over local graphs and attention-based neighbor aggregation on the global graph such that GoGNN is able to capture broader information that enhances the performance on the prediction. Furthermore, the GNNs on both levels of graphs play synergistic effects on improving the representativeness of GoGNN.

The work in [104] introduces GoGNN, the graph of graphs neural network, to solve graph interaction prediction tasks. However, these methods cannot powerfully and distinguishably represent graph structural information together with graph features. Besides, GoGNN only focus on the entity interaction problem, which limits its generalizability.

Motivated by the shortcomings of the current models, we propose our **P**owerful **G**raph **O**f graphs neural **N**etwork (PGON) to capture graph interaction and graph structural information for multiple graph analysis tasks based on the idea of GoGNN that preserves the information from both local and global interaction relationships within and between graphs. PGON uses the invariant and equivariant functions to build the graph neural network following the 2-folklore Weisfeiler-Lehman (FWL) isomorphism test algorithm such that PGON

is able to powerfully preserve the structural information for each molecular or protein graph. PGON also follows the graph of graphs framework. In that way, PGON could consider both levels of the graphs to extract broader information, which improves the performance of PGON on graph analysis tasks.

Contributions. The contributions of the graph of graphs neural network can be summarized as follows:

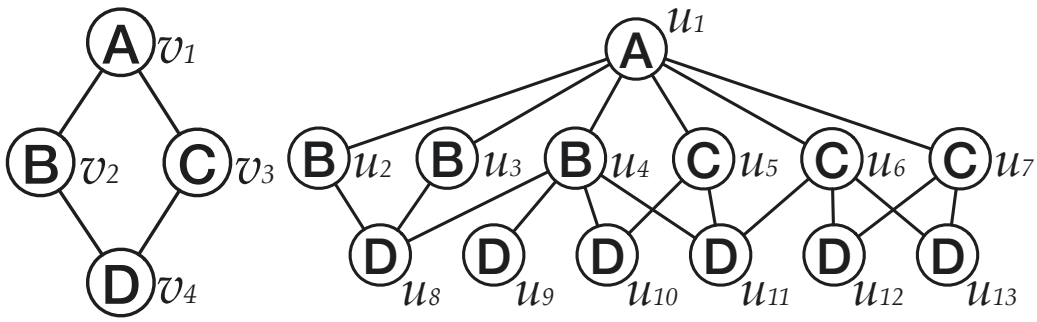
- To the best of our knowledge, this is the first work to systematically apply the graph neural network on graph-of-graphs model, namely Graph of Graphs Neural Network (**GoGNN**), to the problem of structured entity interaction prediction.
- The proposed GoGNN mines the features from both local entity graphs and global interaction graph hierarchically and synergistically. We design dual attention architecture to capture the significance of the substructures in the local graphs while preserving the importance of the interactions within the global graph.
- The extensive experiments conducted on the real-life benchmark datasets show that GoGNN outperforms the state-of-the-art structured entity interaction prediction methods in two representative applications: chemical-chemical interaction prediction and drug-drug interaction prediction.
- We exploit the equivariant and invariant mapping functions to build the graph neural network following the 2-FWL test. Therefore, our model is able to represent the graph structure as powerfully as a 3-Weisfeiler-Lehman test algorithm.
- We study both graph classification and graph interaction prediction tasks from a graph of graphs perspective. Our model could exploit more topo-

logical information and the interactions between local graphs to enhance performance.

- We evaluate our proposed PGON using both graph classification and graph interaction prediction tasks on real-world datasets. PGON has superior performance compared to other baseline methods.

1.3 RL-based Query Vertex Ordering

In recent years, graph structured data has been increasingly ubiquitous. Graph analysis also becomes much more important and popular in the area of data analytics. Subgraph matching is one of the most fundamental problems in graph analysis. Subgraph matching aims to find all embeddings in a data graph G that are isomorphic to a query graph q. For example, in Figure 1.2, $\{(v_1, u_1), (v_2, u_4), (v_3, u_5), (v_4, u_{10})\}$ is a match from q to G. Due to its importance, subgraph matching is widely used in various areas in both academia and industry [86, 79, 94, 112].



(a) Query Graph (b) Data Graph

Figure 1.2: Example query graph and data graph.

Subgraph matching has been proved to be a NP-complete problem [29], which means that the computational complexity is factorial in the worst case. Though

this limit is intrinsic and cannot be overcome, great research efforts still lead to significant advances in reducing unpromising intermediate results, and thus reduce the computational complexity in the average case. The subgraph matching algorithms can be categorized in two classes by their main methodologies: join-based algorithms and backtracking search based algorithms. The join-based approaches [50, 51, 52] convert the query q to a multi-way join by mapping vertices and edges in q to attributes and relations. These approaches evaluate the multi-way join to find matching results. Another class of approaches are the backtracking search based algorithms, which recursively extend intermediate results by mapping query vertices to data vertices along an order of query vertices [97]. Many methods adopt the state space representation for subgraph matching, where each state represents an intermediate result [11], *i.e.,* a mapping from query vertex to data vertex. The methods explore the state space with given order to find feasible state as a match. Among these methods, the query vertex order plays an important role in reducing the average time complexity for subgraph matching, because a well-designed query vertex order could enable the algorithm to prune out the unpromising results as much as possible [7, 36, 37].

In this thesis, we focus on the backtracking search based subgraph matching algorithms. As shown in [95], it is critical to choose a good matching order for the backtracking search based algorithms because it will greatly affect the search performance. Therefore, existing backtracking based algorithms put a lot of efforts to generate good matching orders. Clearly, finding the optimal matching order is time-prohibitive, and existing algorithms resort to a variety of heuristic strategies. For instance, QuickSI [91] proposed a *infrequent-edge first* ordering method which converts q into a weighted graph based on the label frequency of query edges in data graph and generates the matching order along the ascending order of their weights. RI [8] generates the order based on the

structure of q, which selects the vertex with the maximum degree in q as the first vertices, and then picks the vertex that has the most neighbors in order ϕ as the next node. VF2++ [43] picks the vertex whose label is the least frequent in data graph as the first vertex in ϕ. Next VF2++ generates a BFS tree rooted at the first vertex and adds query vertices to ϕ depth-by-depth with tie-break rule based on degree and label frequency of the query node. Other methods such as GraphQL [39], CFL [7] CECI [6] and DP-iso [36] generate the order ϕ by building auxiliary structure of q based on the degrees, labels and candidate sizes of query vertices.

However, these ordering methods only utilize the local structural and label information by greedy heuristics. Consequently, these methods are lacking the ability of fully exploiting the information within the query and data graphs; meanwhile, these methods can only obtain the locally optimal order produced by the greedy search algorithm. The auxiliary structure-based query ordering methods in CFL and DP-iso, which dynamically generate matching orders, can result in a large number of unsolved queries, and hence cannot achieve the state-of-the-art performance according to the empirical study [95].

In this thesis, we propose the first Reinforcement Learning based Query Vertex Ordering method for backtracking based subgraph matching, namely RL-QVO, to fill this research gap. RL-QVO aims to be a more efficient and scalable technique to conduct subgraph matching. We introduce the motivations for this model as follows:

Motivation 1: Make Full Use of the Graph Information. As mentioned previously, the recent subgraph matching models generate ϕ based on the preset priority of heuristics such as degree, label frequency, path frequency, candidate size and etc. In practice, the order is usually determined by partial heuristics, and other statistics are somehow ignored. Besides, the priority of the heuristics

cannot be adapted for graphs with different distributions. Consider q and G in Figure 1.2, if we use the ordering method of RI which selects the first node with the highest degree, the order will be generated randomly due to the structural symmetry of q. Accordingly, the following order is also generated on a random basis. Clearly, the order generated by RI cannot guarantee to reduce the redundant intermediate results for this query. Otherwise, if we build the order according to the label frequency in this case, vertex v_1 will be selected as the starting vertex in ϕ, which eventually reduces unpromising intermediate results. Therefore, adaptive priority for heuristics under particular query graph could significantly enhance the ordering methods, which requires the whole picture of graph information and wise decision making based on these information.

Our Approach: Capture the Graph Information with Graph Neural Network. Regarding the above observation, we exploit the graph neural networks [47, 101, 108] whose ability of feature extraction and representation learning has been widely proven theoretically and empirically, to capture the information hidden in q. We carefully design the initial feature for each query vertex based on the popular heuristics. With the help of GNN's message passing for neighbor aggregation, RL-QVO can naturally capture the query graph's structural and attribute information. Furthermore, thanks to the generality of the machine learning based techniques, RL-QVO is able to select the next node according to current query vertex feature representations.

Motivation 2: Limitation of Greedy Heuristics. Most existing ordering methods utilize the greedy search methods with preset rules which will intrinsically fall into local optimum. Such methods intend to add the vertex that could reduce the most redundant intermediate results to the order ϕ at every step. However, these methods cannot consider the long-term query time cost. The exact optimal order can only be found after all possible order permutations are

evaluated. Therefore, it is challenging for us to develop a subgraph matching algorithm that can break the limitation of the greedy criterion.

Our Approach: Avoiding the Limitation with Reinforcement Learning. Motivated by this drawback, we design a reinforcement learning based framework to learn a policy for matching order generation. We craft the long-term cumulative reward for the RL-based model to force the model to consider the overall computational cost when selecting the next node for matching order ϕ. Instead of only focusing on the current step, the RL-based framework allows the learned policy to consider the states multiple steps ahead before making the decision. We also design an entropy reward to encourage the model to explore the unvisited states. Furthermore, instead of enumerating all possible orders and evaluating the quality of each of them, RL-QVO samples a subset of such orders and learns the policy from the observed rewards. Accordingly, RL-QVO has more probability to produce matching orders that are closer to the optimal ones compared to the existing ordering methods.

The contributions of reinforcement learning based query vertex ordering are summarized as follows:

- To the best of our knowledge, we are the first to employ Reinforcement Learning technique to obtain high-quality matching order for the subgraph matching task.
- Our Reinforcement Learning based model could fully exploit the graph information and select the next query node for the matching order at every step according to the cumulative reward, which can consider the long-term benefits.
- Extensive experiments conducted on 6 real-life graphs demonstrate that the high-quality matching order obtained by RL-QVO can significantly

improve the performance of the backtracking based subgraph matching in terms of the query processing time.

Chapter 2

Literature Review

In this chapter, we overview the related works of machine learning methods for graph analytics. Section 2.1 introduces the representation learning methods for graphs, including the graph embedding techniques and binary hashing. In Section 2.1.2, we review the existing graph neural networks. We first introduce several representative graph neural networks for node-level and graph-level learning. We then list the popular applications of graph neural networks. In Section 2.2, we present the related works on the graph related applications. Specifically, we focus on the related works on the graph of graphs, structured entity interaction prediction and subgraph matching tasks, which also are the studied problems in this thesis. In Section 2.3, we briefly introduce the machine learning techniques used in this thesis, *e.g.,* the reinforcement learning methods and binaried neural networks.

2.1 Graph Representation Learning

In this section, we first review graph embedding methods which are effective methods of graph representation learning. Then, we review the binary hashing

methods that are frequently used to reduce the time and space complexity in many applications.

2.1.1 Graph Embedding

A key problem in machine learning on graphs is finding a way to incorporate information about the structure of the graph into the desired machine learning model. Graph embedding is one of the most promising approaches because it maps nodes into a low-dimensional space such that the structure of the graph is well preserved. Once accomplished, an existing machine learning approach (e.g., k-means clustering) can be used to assimilate and analyse the graph in the embedded low-dimensional space. Loosely following the seminal graph embedding approach, DeepWalk, three broad categories of embedding methods have appeared in the literature: (1) node similarity based embedding methods (e.g., LINE, NetMF), which rely on the proximity of the nodes w.r.t various similarity metrics. The matrix factorization techniques have been used to learn the embedding of the nodes. (2) Random walk based embedding methods (e.g., Deepwalk and node2vec) which encode the nodes by applying the Skip-Gram technique [75] on the random walks; and (3) graph neural networks (GNN) based embedding methods (e.g., GCN, GraphSage and GIN) which apply the neural network techniques on graph to learn the representations of the nodes.

Most of the existing graph embedding studies use the real-valued vector to encode the graph nodes following the above three computing paradigms. Recently, three unsupervised approaches [64, 93, 113] have been proposed to learn the *binary embedding* of the graphs following the node-similarity based embedding methods. Particularly, INH-MF [64] and DNE [93] are independently developed for binarized graph embedding based on the discretization of the matrix factorization on proximity graphs. BANE proposed in [113] is a natural extension of

DNE by considering both structure and attribute similarities on the attributed graphs.

2.1.2 Graph Neural Networks

Recently, the graph neural networks have been proven to be a popular and powerful technique for graph analytics. In this section, we first introduce several representative graph neural networks for node-level and graph-level applications. We then introduce the graph neural networks designed for specific applications.

Node-level GNNs. Most GNNs are designed for node-level applications such as node classification and link prediction [47, 100, 116, 34, 68, 63, 62]. They rely on the node embedding techniques like skip-gram, autoencoder and neighbor aggregation methods like GCN, GraphSAGE, etc. These methods focus on the node relations within the graph and use the low-dimension representations to preserve the structural and attribute information.

Graph-level GNNs. Recently, some research works on GNNs are proposed for graph-level applications such as graph classification[117, 55] and graph matching[59]. These works learn the graph representations for each graph individually or pair-wisely without considering the interactions between the graphs.

2.1.3 Graph Neural Networks for Specific Applications

There are several applications that are based on the GNN. Such as Graph Matching Network [58] and SimGNN [2]. These models utilize GNN and use the similarity (distance) of graph embedding to approximate the graph edit distance and graph similarity.

The Graph Matching Network (i.e., GMN) is a novel GNN-based framework proposed by *DeepMind* to compute the similarity score between input pairs of

graphs. Separate MLPs will first map the input nodes in the graphs into vector space. Then the propagation layer will aggregate the messages of the edges and cross-graph matching vector by MLP or GRU with input concatenation of node representations and edge vectors. Matching function is applied to compute the attention coefficients based on the node information between the input pair of graphs. The matching function is based on the softmax function over node vectors which requires the calculation of vector space similarity like Euclidean, cosine similarity or dot product between all pairs of node representations. This attention coefficients calculation across two graphs requires a computation cost of $\mathcal{O}(\mid V_1 \mid\mid V_2 \mid d)$, where V_1 and V_2 indicate the number of vertices of input graph 1 and 2 respectively, and d is the dimension of the node representation. The match vector $\mu_{j\to i}$ is concatenated with the message vector $\mathbf{m}_{j\to i}$ and the node representation $\mathbf{h}_i^{(t)}$, then the concatenation is fed into MLP or a recurrent neural network core to produce the new node representations. Given the learned node representations of graph, the aggregation module proposed in [60] is used to obtain the graph representations. The similarity score in vector space such as Euclidean similarity, cosine similarity and approximate hamming similarity will be computed between graph representations to approximate the similarity between the input graphs.

2.1.4 Binary Hashing

The binary hashing has been widely used to learn the binary vectors (codes) of the objects in many applications. The most popular application is the approximate nearest neighbor search in high dimension space where binary hashing methods encode high-dimensional objects (e.g., documents and images) to binary codes, while preserving similarity distance in the original space. Many learning to hash approaches have been proposed including unsupervised meth-

ods (e.g., [87, 66]), supervised methods (e.g., [92]), and deep learning based methods (e.g., [65]). Please refer to [105] for a comprehensive survey. Recently, three approaches [64, 93, 113] have been proposed to learn the binary embedding of the graphs following the node-similarity based embedding methods. As to our best knowledge, there is no existing work on the binarized graph embedding based on GNNs.

2.2 Graph Applications

In this section, we review the related works about the graph-based applications. We first introduce the applications of a specific graph structure, *i.e.,* graph of graphs. We then review the models designed for structured entity interaction prediction task. Lastly, we review the approaches proposed to solve subgraph matching, an important and fundamental problem in graph analytics literature.

2.2.1 Graph of Graphs

In most real-world systems, an individual network is one component within a much larger complex multi-level network. Applying the graph theory paradigm to these networks has led to the development of the concept of "Graph of Graphs" (also known as "Network of Networks"). [18] introduces the theoretical research development [24], applications [77] and phenomenological model [84] on the network of networks. These works enable us to understand and model the interdependent critical infrastructures. SEAL[57] proposed graph neural network in a hierarchical graph perspective for graph classification task. With significant differences between GoGNN and SEAL in tasks, loss functions and optimizers, GoGNN is the first work to develop graph neural network technique on graph of graphs for structured entity interaction prediction problem.

2.2.2 Structured Entities Interaction Prediction

In many real-life applications such as chemistry, biology, material science, and medical science, we need to understand the interactions between the structured entities. In recent years, a variety of techniques have been proposed for structured entity interaction prediction in some specific applications. In this thesis, we focus on two representative applications: chemical-chemical interaction prediction and drug-drug interaction prediction.

Many computational methods have been proposed for these two applications. DeepCCI and DeepDDI [49, 85] utilize the conventional convolutional neural network and PCA on the chemical data. Some models are graph neural network-based. For example, Decagon [121] performs the GCN on drug-protein interaction graph; MR-GNN [111] proposes a model with dual graph-state LSTMs that extracts local features of molecule graphs, and MLRDA [14] utilizes graph autoencoder with a novel loss function to predict the drug-drug interactions.

2.2.3 Subgraph Matching

Subgraph matching is an important and fundamental problem in database literature, which is widely applied in both academia and industry. There are two categories of subgraph matching methods. Most subgraph matching models can be categorized into backtracking search algorithms, such as QuickSI [91], CFL [7], RapidMatch [96], Turbo-iso [37], DP-iso [36], VF2 [15], VF2+ [12], VF2++ [43], VF3 [11], the subgraph matching system GraphZero [71], the benchmark work LSQB [74] and etc. These models follow the filtering, ordering and enumeration framework introduced above to obtain all the matches for query nodes. Specifically, their enumeration processes are backtracking search procedures that are able to find all possible matches. Backtracking search al-

gorithms generally use greedy approaches to generate matching order based on some heuristic rules or cost estimation. The other category is the join-based methods like TwigTwig [50]and SEED [51]. These algorithms usually aim at the subgraph enumeration problem, where the label information is not considered and distributed techniques are usually utilized. These algorithms convert subgraph enumeration problem to a multi-way join task. Among the join-based algorithms, the worst-case optimal join has the running time that is correlated to the maximum query output size [76]. It has been proved that the WCOJ based database system LogicBlox [1] outperformed other database systems (*e.g.,* PostgreSQL, Neo4j and MonetDB). In this thesis, we aim at designing an ordering method for backtracking search based algorithms.

Recently, there have been several research attempts [69, 67, 13] to develop subgraph matching algorithms on a learning basis. These methods focus on counting of isomorphic subgraphs or producing approximate matching results, which are different from the target in this thesis, *i.e.,* finding exact subgraph mappings from queries to data graph.

2.3 Machine Learning Techniques

In this section, we introduce two important machine learning techniques used in this thesis. We first reviews the reinforcement learning based methods that are used for graph-related problems. We also review the binarized neural networks that aim to reduce the space and time complexity of neural network.

2.3.1 Reinforcement Learning.

Using reinforcement learning for graph-related problems is becoming increasingly popular. [19, 48, 61, 30, 83] aim to solve the NP-hard combinatorial optimiza-

tion problems with reinforcement learning and graph representation learning techniques. Please see [5] for detailed survey. Several RL-based models have been proposed for conventional graph analysis tasks. For example, [40] exploits a GNN-based policy network in promoting the performance of active learning for cross-domain graph analysis tasks such as classification; [114, 22] utilizes graph policy networks on the molecular graphs for chemical and medical applications such as chemical/drug discovery and chemical reaction prediction.

2.3.2 Binarized Neural Networks

Binarized neural networks was first proposed by BNN [16]. The binarization technique proposed in [16] is used by most network binarization models. Among them, XNOR-Net [82] and DoReFa-Net [119] are the most popular ones because of their great performance on the image classification task.

XNOR-Net was proposed to have high accuracy of classification task on the ImageNet dataset while XNOR-Net has 58× faster convolutional operations and 32× memory saving. DoReFa-Net replaces the binarization by quantization which allows the model to change the bit size for weights, activations and even gradient calculations during backpropagation.

However, these methods are all designed for computer vision tasks. Though they perform well on the image dataset, they cannot be adapted to graph analysis tasks directly. In this thesis, we propose a novel binarized graph neural network for graph representation learning and downstream applications.

Chapter 3

Binarized Graph Neural Network

3.1 Chapter Overview

In this chapter, we introduce the methodological details about the binarized graph neural network and the evaluation result of the model. This work is published in [103]. Section 3.2 formally introduces the background and preliminaries related to the binarized graph neural network. Section 3.3 presents the details of our proposed model BGN, including the overall framework, the binarization function, training objectives and improvement techniques. Section 3.4 reports the experiment results which demonstrate the effectiveness and efficiency of binarized graph neural network. Section 3.5 concludes this chapter.

3.2 Background and Preliminaries

Recent studies have revealed that graph neural network can perform excellently on label classification tasks. The existing GNN-based graph embedding approaches share the same computing paradigm. GNNs take graph nodes' feature and neighborhood information as the input. During the training, the repre-

Notation	Definition
G	the graph dataset
$\mathcal{V}, E$	the set for nodes and edges in the graph.
v, e	node and edge in the graph.
$\mathbf{x}_v$	the feature information for node v.
η_v	the neighborhood nodes of node v.
$(\cdot)^{\mathbf{b}}$	denotes that the vector or matrix is binary-valued.
$\mathbf{h}_v$	the hidden representation of node v.
$\mathbf{W}$	the weight matrix in the neural network.
$\mathcal{B}(\cdot)$	the binarization function which is used to transform the real-valued vector or matrix into binary-valued vector or matrix.
α_{ij}	the attention coefficient between node i and node j.

Table 3.1: Summary of Notations

sentations of nodes (real-valued vectors) at each layer will be updated by the aggregators and non-linear activation functions. The output representations will be fed into the task-specific layer to calculate the loss of the model. Based on that, the model will be optimized by the optimizer through backpropagation. The main differences among these GNN-based graph embedding approaches are the design of the aggregator which combines the context representations and the loss function designed for different graph analytic tasks.

These models have real-valued parameters and learn a real-valued representation for each node in an end-to-end manner for graph node classification. However, the real-valued parameters and representations are space-consuming for storage and time-consuming for multiplication computation, especially for large-scale graphs. To address these issues, in this chapter we devise a novel binarized graph neural network, namely BGN, with binary parameters in the neural network to learn binary embedding representations for node classification task.

The important notations used throughout the chapter are summarized in Table 3.1.

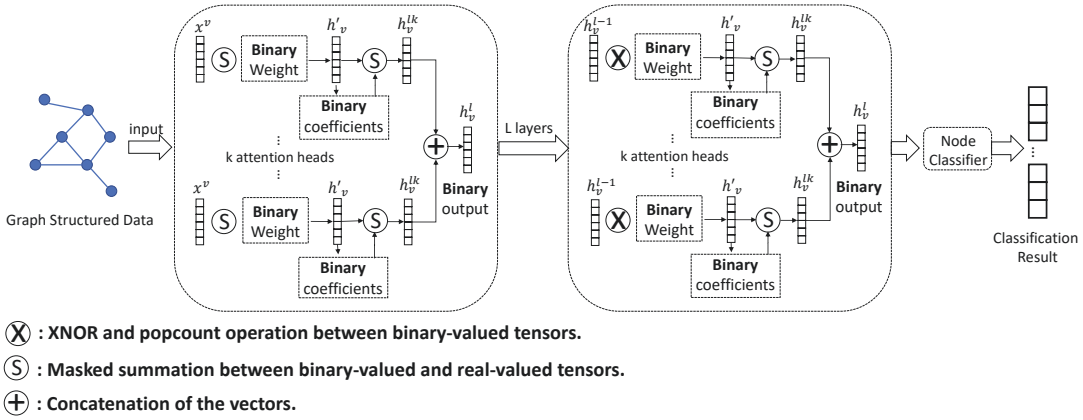


Figure 3.1: The overall framework of the proposed model BGN. (a) All input node features are projected into a unified representation space by binary-valued weights.(b) Masked summation between binary matrix and real-valued matrix is employed to speed up the dot product. (c) Binary attention coefficients are produced based on the hidden representations. (d) Output of the layer is calculated via multi-head attention mechanism. (e) xnor and popcount are employed to calculate the dot product between binary-valued matrix. (f) Loss calculation and end-to-end optimization for the node classification task.

3.3 Model

As illustrated in Figure 3.1, we introduce a new graph neural network with binarized weights and activations. Our model BGN (**B**inarized **G**raph Neural **N**etwork) is based on the attention mechanism and can be easily adapted into other graph neural network frameworks. For a given graph, BGN takes the nodes and their contexts including feature and neighborhood structure information as input. Binarization function will transform the weights, activations and even coefficients into binarized vectors to reduce the time and space complexity, while the attention mechanism enables the nodes to attend over their neighborhoods' features. We also apply the balance function to ensure that $+1$ and -1 are almost equal with each other in the binarized vectors. Furthermore, the gradient

estimator is used for backpropagation of gradients through discretization.

The following subsections present the listed key components of our model:

- Section 3.3.1 introduces the **framework** of our work.
- Section 3.3.2 introduces the **binarization** of our model in detail, including the **forward propagation** and **backpropagation**.
- Section 3.3.3 describes the **optimization objective** of our model.
- Section 3.3.4 introduces the techniques we used to reduce the time and space complexity and improve the performance.
- Section 3.3.5 introduces the **adaptation** of our model to other GNN frameworks.

3.3.1 Framework

Algorithm 1 illustrates the framework of our model. We follow the attention mechanism introduced in [99, 100] to involve the *importance* of the node's neighborhoods into the graph representation learning process. Given a graph $G(\mathcal{V}, E)$, where $\mathcal{V}$ and E denote the set of graph nodes and edges respectively, we use nodes features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}, \mathbf{x}_v \in \mathbb{R}^m$ and the neighborhood information of nodes $\{\eta_v, \forall v \in \mathcal{V}\}$ as inputs. Our model will first produce the binarized node representations $\mathbf{h}_v^b \in \{+1, -1\}^d$ for each node within the input graph. After that, the binarized node embeddings will be fed into the output layer to compute the loss for some specific tasks like node classification.

Attention Mechanism. Our proposed framework is based on the graph attention mechanism. The attention layer is utilized in our model to learn the importance of every node to other nodes. The key is to get the *importance* of

Algorithm 1: Binarized Graph Neural Network

Input: Graph $G(\mathcal{V}, E)$, node features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$, number of layers L, binarization function $\mathcal{B}(\cdot)$, number of attention heads K, neighbors of node $\{\eta_v, \forall v \in \mathcal{V}\}$

Output: Classification result $\{\mathbf{C}_v, \forall v \in \mathcal{V}\}$

Let $\mathbf{h}_v^0 = \mathbf{x}_v, \forall v \in \mathcal{V}, \forall u \in \eta_v$; **for** $l = 1, 2, \ldots, L-1$ **do**
  $\alpha_{uv}^l = \mathcal{B}(Softmax_v(\mathbf{W}_0^l \mathbf{x}_u, \mathbf{W}_0^l \mathbf{x}_v))$;
  for $v \in \mathcal{V}$ **do**
    for $k \in K$ **do**
      $\mathbf{h}_v^k = \mathcal{B}(\sum_{u \in \eta_v} \alpha_{uv}^k \mathbf{W}_k^l \mathbf{x}_u)$;
    $\mathbf{h}_v^l = Concat_{k=1}^K(\mathbf{h}_v^k)$;
$\alpha_{uv}^L = \mathcal{B}(Softmax_v(\mathbf{W}_0^L \mathbf{h}_u^{L-1}, \mathbf{W}_0^L \mathbf{h}_v^{L-1}))$;
for $v \in \mathcal{V}$ **do**
  $\mathbf{C}_v = Softmax(\sum_{u \in \eta_v} \alpha_{uv}^k \mathbf{W}^L \mathbf{h}_u^{L-1})$;
return $\mathbf{C}_v$, the classification result for node $v \in \mathcal{V}$;

one node's feature to other nodes that is the attention coefficients of the input graph, afterwards, the node's feature can attend on other nodes. Inspired by [100], we perform *masked attention* to the model to keep the structural information of the input graph. Only the attention coefficients of one node with its neighborhood nodes i.e., $\alpha_{ij}, v_j \in \eta_i$ will be computed.

In order to obtain the attention coefficients, we use a shared **binarized** weight matrix $\mathbf{W} \in \{+1, -1\}^{m \times d'}$ to apply the linear transformation to each node. Softmax function is used to normalize the coefficients, but unlike the model proposed in [100], LeakyRelu activation is not employed in our model while the sign function is used to binarize the attention coefficients. With the following Equation (3.1), we will get a **binarized** attention coefficient matrix $\mathcal{A} \in \{+1, 0, -1\}^{N \times N}$ where α_{ij} is the element of the matrix $\mathcal{A}$ (0 is contained in the matrix since we only compute the attention coefficients between neighbors

such that the matrix is sparse).

$$\alpha_{ij} = \mathcal{B}'(Softmax_j(\mathbf{W}\mathbf{x}_i, \mathbf{W}\mathbf{x}_j)) \tag{3.1}$$

where $\mathcal{B}'$ is the binarization function for attention coefficients which maps 0 to 0, positive values to $+1$ and negative values to -1.

Once the attention coefficient matrix is obtained, it will be used to compute the output of the attention layer. The attention coefficients will multiply the linear transformed node feature. We employ the *multi-head attention mechanism* to stabilize the learning process. The binarization function, which is served as activation function, is applied to every attention head to binarize the pre-activations. And concatenation of the output of K independent attention heads is the output of the attention layer. Therefore, the output node representation will be like following:

$$\mathbf{h}_i = \|_{k=1}^{K} \mathcal{B}(\sum_{j \in \eta_i} \alpha_{ij}^k \mathbf{W}^k \mathbf{x}_j) \tag{3.2}$$

Where $\|$ means the concatenation of the vectors and $\mathbf{h}_i$ is the output **binarized** node representation where $\mathbf{h}_i \in \{+1, -1\}^d$.

After several attention layers, the node representation will be fed into the last layer to calculate the loss for specific task which is classification in this chapter. We will introduce the learning objective in the Section 3.3.3.

3.3.2 Binarization

In this section, we introduce how to obtain a graph neural network with binary parameters that can learn binary representations. Section 3.3.2 introduces the binarization function used to transform the real-valued parameters and pre-activations into binary space. Section 3.3.2 introduces the gradient estimators that enable the binarized model to be optimized by the off-the-shelf optimizers

such as Adam and SGD.

Forward Propagation

Binarization function is important in our model. Specific binarization function will be chosen in the forward propagation calculation process to binarize the weights and the activations. In that way the low-bit parameters and activations will help to reduce the time and space complexity. In our case, various binarization functions will work, and the most straightforward example is the sign function. As mentioned in [16] and [82], deterministic and stochastic binarization based sign function are widely applied to the continuous pre-activations as well as the real-valued weights to obtain binarized activations and weights.

$$\mathcal{B}_{det}(x) = \begin{cases} +1 & x \geq 0, \\ -1 & else, \end{cases} \tag{3.3}$$

The above equation is the deterministic binarization function, where x is the real-valued variable. The stochastic binarization is the sign function with probability:

$$\mathcal{B}_{stoch}(x) = \begin{cases} +1 & \text{with probability } p = \sigma(x), \\ -1 & \text{with probability } 1 - p, \end{cases} \tag{3.4}$$

where σ denotes the sigmoid function, that is $\sigma(x) = 1/(1 + exp(-x))$. The stochastic binarization is more appealing but needs the computer to generates random bits while the deterministic binarization is easier to calculate. Deterministic binarization function(i.e., Equation (3.3)) is applied for the binarization of weights and activations because the deterministic sign function provides more stable and reproducible results. Please note that we use a variant of deterministic sign function which maps 0 to 0 to binarize the attention coefficients.

Backprobagation

In this part, we describe how to backpropagate the gradients through the binarization function. We adapt the gradient estimator into our model for better optimization.

Propagation gradients through binarization function. It is obvious that the binarization function has zero derivative almost everywhere, which leads to the zero gradients of the loss function w.r.t the pre-activations and weights. The trainable variables cannot be updated with zero gradient. Therefore, the model cannot be trained by simple backpropagation, and the estimation of the gradients should be obtained for optimization. Previous studies have investigated how to propagate gradients through stochastic discrete functions. Below we investigate two popular unbiased gradient estimators for binarization function: straight through estimator and REINFORCE estimator [107].

Straight through estimator. The straight-through estimator is proposed a simple unbiased gradient estimator. It estimates the derivative of binarization function $\mathcal{B}(\mathbf{h})$ of pre-activation or weight $\mathbf{h}$ as $\mathbf{1}$ (a vector or matrix whose elements are all 1). Let $\mathbf{h}^b$ denote the binarized representation and $\mathbf{h}$ denote the pre-activation before binarization. The straight-through estimation of the gradient of the loss L w.r.t the pre-activation $\mathbf{h}$ is thus:

$$g_h = \frac{\partial L}{\partial \mathbf{h}} = \frac{\partial L}{\partial \mathcal{B}(\mathbf{h})} \cdot \frac{\partial \mathcal{B}(\mathbf{h})}{\partial \mathbf{h}} = \frac{\partial L}{\partial \mathbf{h}^b}\mathbf{1} = g_{h^b}\mathbf{1} \tag{3.5}$$

This gradient will then be back-propagated to obtain the gradient of quantities (i.e., pre-activations or weights) that influence $\mathbf{h}$.

REINFORCE estimator. The reinforce estimator is proposed in [4] to estimate the expectation of the gradient $\frac{\partial L}{\partial \mathbf{h}}$ of loss L with regard to the pre-activation vector or weight $\mathbf{h}$. When binarization function $\mathcal{B}(\cdot)$ is stochastic with the prob-

ability given by sigmoid, it has been proven that:

$$\mathbb{E}(\frac{\partial L}{\partial \mathbf{h}}) = \mathbb{E}[(\mathcal{B}(\mathbf{h}) - \sigma(\mathbf{h}))(L - c)] \tag{3.6}$$

where σ is the sigmoid function and c is a constant vector. To minimize the variance of the estimation, c can be chosen as:

$$c = \frac{\mathbb{E}[(\mathcal{B}(\mathbf{h}) - \sigma(\mathbf{h}))^2 L]}{\mathbb{E}[(\mathcal{B}(\mathbf{h}) - \sigma(\mathbf{h}))^2]} \tag{3.7}$$

The reinforce estimator can work directly on the weights and pre-activations without actual computation of the gradient. The estimation is obtained by monitoring numerator and denominator during the training process.

Compared with straight through estimator, reinforce estimator is more advanced with better performance in many applications. However, we observe that its performance is not superior than the straight through estimator. On the other hand, straight through estimator helps the model to obtain the gradient faster than the reinforce estimator due to its simplicity. The comparison between these two gradient estimators with regards to the performance is included in Section 3.4. In practice, we choose straight through estimator for our model in the experiments.

3.3.3 Optimization Objectives

Existing GNN-based graph embedding approaches provide an end-to-end model, most of which focus on the node classification task. Therefore, our model is also learned for the node classification task. Below, we introduce the objective of BGN and the learning process that optimizes the parameters.

For the node classification learning, we feed the binarized embedding $\mathbf{h}_v^b$ into the output layer to predict the class label for the node. The predicting probability

of label $\mathbf{C}_i$ is written as:

$$p(\mathbf{C}_{vk} \mid \mathbf{h}_v^b) = Softmax_\zeta^k(\sum_{u \in \eta_v} \alpha_{uv}^L \mathbf{W}^L \mathbf{h}_u^b) \tag{3.8}$$

where ζ denotes the number of labels for each node. After obtaining the classification result in Equation (3.8), we calculate the cross-entropy as the loss for the node classification task.

$$L_{class} = - \sum_{v \in \mathcal{V}_{labeled}} \sum_{k=1}^{\zeta} \mathbf{C}_{vk}^{\mathcal{L}} \log(\mathbf{C}_{vk}) \tag{3.9}$$

where $\mathcal{V}_{labeled}$ is the set of nodes that have label information which are used for training process, $\mathbf{C}_{vk}^{\mathcal{L}}$ is the multi-hot encoding for ground truth classification labels.

The gradients will be back propagated via estimator and be applied on the optimization of parameters by the off-the-shelf optimizer during the training process.

3.3.4 Techniques to Improve the Model

Several techniques are used on binarized graph neural network model to reduce the time and space complexity and improve the performance. Logic operation *xnor* between binary values, build-in CPU instruction *popcount* and the masked summation are used to replace the tradition arithmetic operation dot product to reduce time complexity. The Figure 3.2 is a toy example that introduces the differences between these operations. Balance function is used to make $+1$ and -1 to be balanced in the embedding vectors which can raise the performance of the GMN. Also, the binary parameters of the neural network and the binary node representations can reduce the space complexity intuitively.

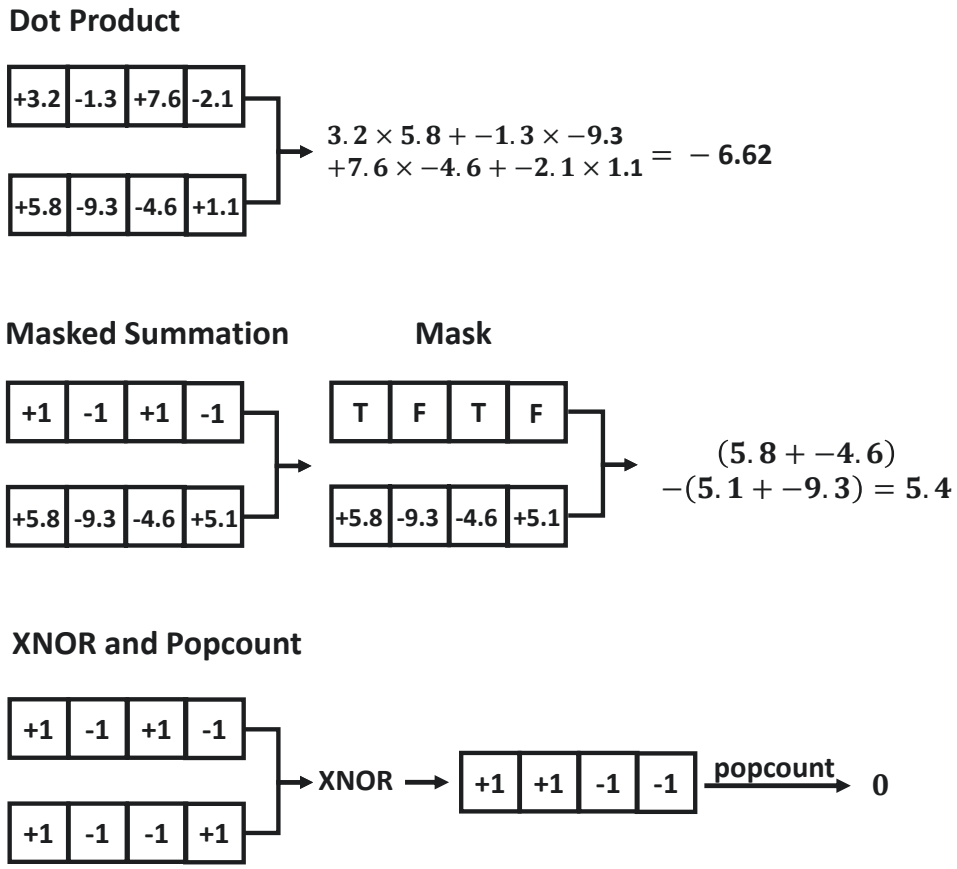


Figure 3.2: The toy examples of (a) dot product (b) Masked summation and (c) *xnor* and *popcount* instruction

XNOR and Popcount

The logic *xnor* and CPU build-in instruction *popcount* between binary matrices are used to replace the dot product between them.

As shown in Table 3.2, *xnor* produces binary value with input of $+1$ and -1. Instruction *popcount* is then be employed to count the number of bits that is set to 1. The *xnor* can be more than one order of magnitude faster than the dot product which can dramastically reduce the time complexity. As mentioned

Input A	Input B	Output
+1	+1	+1
+1	-1	-1
-1	+1	-1
-1	-1	+1

Table 3.2: *xnor* calculation

in [16], a 32-bit floating point multiplier costs about 200 Xilinx FPGA slices, whereas a 1-bit *xnor* gate costs only 1 slice.

Masked Summation

Masked summation is used to replace the dot product between binary matrix and real-valued matrix. The binary matrix will be transformed into the mask matrix with "True" and "False". During the multiplication, the real-valued vector will be masked by the corresponding mask vector, then the positive and negative masked vector are produced with only the elements at the same position as "True" and "False" on the mask vector. The model calculates the summations of the positive and negative masked vector separately. The subtraction of these two summation results is the result of dot product between the given matrices.

The masked summation can reduce the time complexity of dot product of two matrix. Usually, the time complexity of naive dot product between two real-value matrices $M_1 \in \mathbb{R}^{m \times n}$ and $M_2 \in \mathbb{R}^{n \times d}$ is $\mathcal{O}(mnd)$, while the time complexity of *masked summation* between binary matrix $M_1 \in \{-1, +1\}^{m \times n}$ and real-valued matrix $M_2 \in \mathbb{R}^{n \times d}$ is $\mathcal{O}(nd)$. Theoretically and also in practice, the masked summation can significantly reduce the time complexity of our proposed binarized graph neural network.

Balance Function

The distribution of $+1$ and -1 is sometimes unbalanced in the representation vectors. For example, if most pre-activations $\mathbf{h}$ have *positive* elements, the output graph representation vector of binarization function $\mathbf{h}^b$ will be formed mainly by $+1$. Then the dot product of two vectors will be d which is the dimension of the vectors. This unwanted situation should be avoid because it dramatically lower the effectiveness of the proposed model, especially when the BGN is applied to GMN which requires a great number of dot product between representations. As a result, we apply the following balance function to the pre-activations before binarization in order to balance the distribution of *positive* and *negative* elements of pre-activations:

$$Balance(\mathbf{h}) = \mathbf{h} - \overline{\mathbf{h}} \tag{3.10}$$

where the $\overline{\mathbf{h}}$ is the vector whose elements are all mean value of the pre-activation vector $\mathbf{h}$. The balance function ensures that the pre-activation vectors contain almost half *positive* and half *negative* elements, which leads to the balance distribution of $+1$ and -1 after binarization.

3.3.5 Adapted to Other GNN Based Models

The proposed binarized graph neural network is a very general framework that can be adapted to other graph neural network-based model to project the real-valued parameters and activations into the binary space to reduce the space and time cost. We introduce how we binarize the state-of-the-art GNN-based model AS-GCN [41] and the graph matching network.

Binarization of AS-GCN

AS-GCN is a general framework that is designed for fast representation learning based on graph neural networks such as GCN. Therefore, the binarization of AS-GCN is similar to our proposed BGN. We use deterministic binary function to binarize the parameters and pre-activations of AS-GCN. And straight through estimator is employed for back propagation. The binarized model is denoted as BGN-ASGCN in our experiment.

Binarization of GMN

As mentioned above, the time cost of GMN comes mainly from the pair-wise node similarity computation. We utilize the deterministic binarization function (Equation (3.3)) on the preactivations and transform the node and graph representations into binary codes such that the xnor can be applied to replace the dot product. Straight through estimator (Equation (3.5)) is used for the back propagation. Furthermore, we noticed that the distribution of $+1$ and -1 is usually not symmetric which dramatically lower the performance, hence, balance function (Equation (3.10))is employed on the graph representations.

3.4 Experiment

We conduct extensive experiments to evaluate the performance of our model for the node classification task on real-world network datasets. We compare the time and space efficiency thoroughly between the proposed model and other baseline models. The case study shows the effectiveness and efficiency brought by our framework on the GNN-based application such as GMN.

Dataset	#Nodes	#Edges	#Classes	#Labled Nodes
Cora	2708	5429	7	140
Citeseer	3327	4732	6	120
Pubmed	19717	44338	3	60

Table 3.3: Citation Datasets

3.4.1 Dataset

To facilitate the comparison between our model and the relevant baselines, we conduct the classification experiments on three well-known citation network datasets: **Cora**, **Citeseer** and **Pubmed** [90]. Each dataset contains bag-of-words representations of documents and citation links between the documents. Graph G is constructed based on the citation links. In the classification task, we only use 20 labeled instances per class for training. The test data contains 1000 nodes as in GCN, GAT and AS-GCN.

The details of the datasets are summarized in the Table3.3.

3.4.2 Baseline Methods

The following GNN-based and binary embedding methods are compared as baselines:

GCN (Graph Convolutional Network) [100] is a semi-supervised neural network method for node classification.

GAT (Graph Attention Network) [100] is a graph neural network model which first exploits the attention mechanism to solve the node classification task.

AS-GCN (Adaptive Sampling over GCN) [41] is a state-of-the-art method for node classification task. AS-GCN aims to increase the scalability of GCN using adaptive sampling. The experiments demonstrate that the application of BGN can further reduce the time and space complexity of AS-GCN.

GAT-binary and **ASGCN-binary** are the models that directly apply sign

function on the node representations learned by the original version of GAT and AS-GCN. The naively binarized representations will be fed into the task-specific layer to learn the classification result.

GAT-tanh and **ASGCN-tanh** are the models that employ the binarization function *tanh* used by *DeepMind*'s work. *tanh* function is used to binarize the parameters and embedding vectors of GAT and AS-GCN. We clip the value of the parameters and activations in both models to make sure that *tanh* can produce "exact" binary codes.

INH-MF [64] is a MF-based information network hashing algorithm that learns binary codes as node embedding which can preserve high-order proximity.

BANE(Binarized Attributed Network Embedding) [113] is an extension of DNE [93] which based on the Weisfeiler-Lehman proximity matrix factorization learning function to produce binary node representations.

3.4.3 Experiment Setup

For the performance experiment, we evaluate the models with the same bit-width representations. For the experiment of inference efficiency, the embedding dimension of our method and other baseline methods are all set to 64. During training process, the whole graph can be seen, but only a few nodes are labeled while most nodes have no label information. We put all nodes information in one training phase due to the need of calculation for graph attention coefficients.

For this classification task, we report the average accuracy of the evaluated GNN-based embedding approaches after ten independent runs using the accuracy metric introduced in [46, 100]. Because INH-MF and BANE only produce the binary embedding vectors but have no build-in classifier, we employ the one-vs-rest logic regression implemented by Liblinear [26] to obtain the classification result of the networks, in which 90% nodes are labeled.

All the experiments were conducted on the server which is running RHEL 7.5 and has 2x 2.4GHz Intel Xeon E5-2680 v4 (14 Cores) CPU, 256GB 2400MHz ECC DDR4-RAM and 2x NVIDIA Quadro P5000 16GB Graphics Card (GPUs) (2560 Cores).

3.4.4 Classification Results

Because our model produces the compact representations for vertices, we compare the performance between our model and other baselines with the same bit width.

Comparison Among Binary Embedding Methods

We compare the classification results between our model and other binary-valued embedding methods.

As shown in the Figure 3.3, under different embedding dimensions, BGN outperforms all the other binary-valued embedding methods significantly on all three datasets. With the help of the graph neural network, our model can make better use of the graph structured data and feature information and is trained specifically for the node classification task. Therefore, our model outperforms other MF-based binarized graph embedding models by a significantly large margin. In comparison with the naively binarized GAT-binary and ASGCN-binary, our model considers the binary property of parameters and vectors during the training process, hence our model achieves better accuracy. In terms of GAT-tanh and ASGCN-tanh, because *tanh* function has zero gradient when the output is nearly $+1$ or -1 and has real-value output when the gradient is not zero. This property determines that *tanh* function is not suitable for binarize the neural network. When the input values are clipped to produce exact binary parameters and embeddings via *tanh* function, the gradient will be zero which results in the

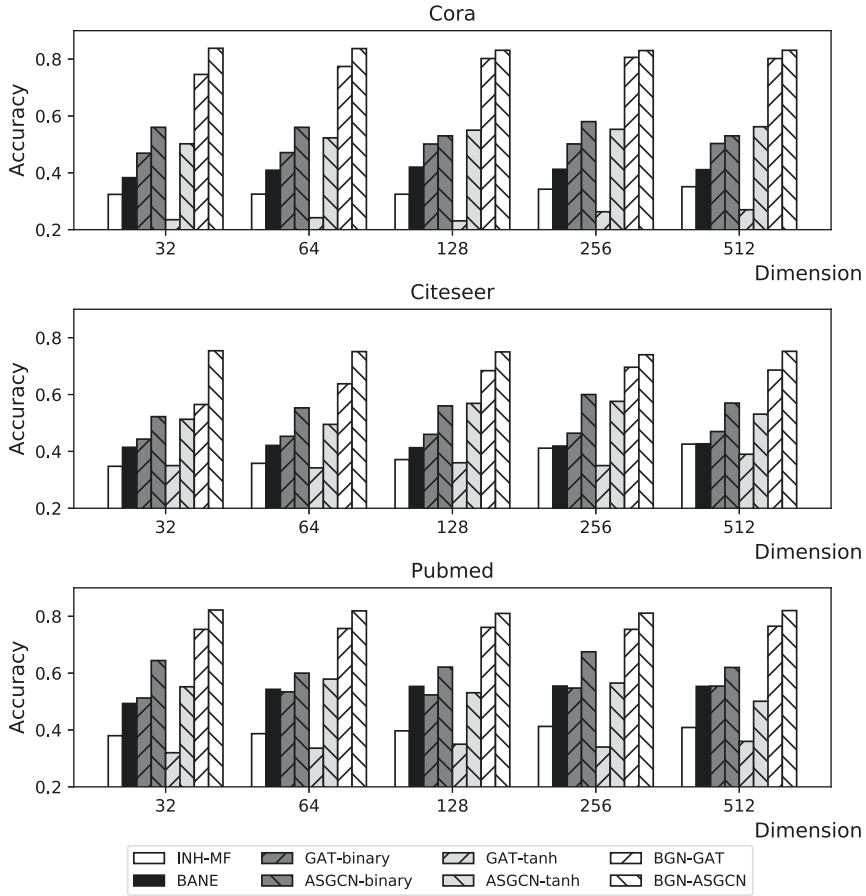


Figure 3.3: Classification results of three citation network dataset among the binary-valued embedding methods with different embedding dimensions

insufficient optimization and worse performance than BGN.

Comparison among the GNN-based methods

We compare our model with other GNN-based methods (GCN, GAT and AS-GCN). All baseline methods produce the real-valued embedding vectors each dimension of which is encoded by at least 32 bits. Compared with these methods, each dimension of the embedding vectors learned by our model is only encoded by 1 bit. As a result, a real-valued 16 dimension vector requires at least 256 bits while a binary vector only requires 16 bits. Figure 3.4 shows the performance of the models with bit width varies for a single embedding vector.

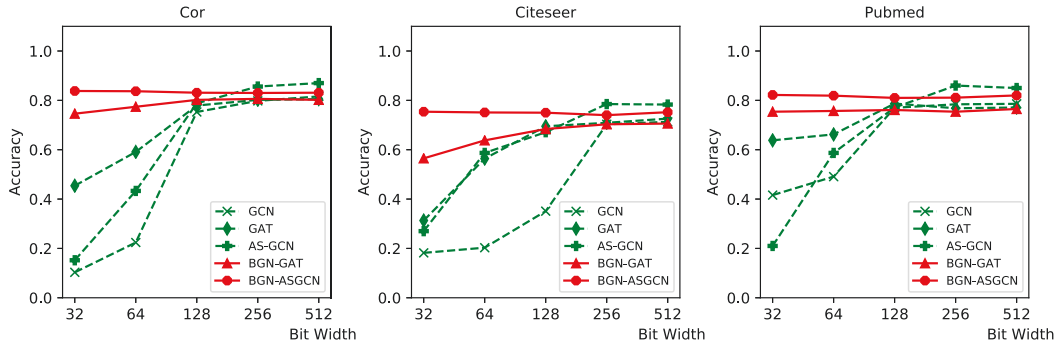


Figure 3.4: Classification results of three citation network dataset among the GNN-based methods with varied bit width for embedding vector

Our model significantly outperforms all the baseline methods with low bit width. When getting more space for the learned representations, our model can still achieve competitive classification results compared with the state-of-the-art graph neural network-based methods. In conclusion, the performance gap between our model and baselines with large bit-width representations is acceptably small while our model's performance is notably better with the low bit-width representations.

3.4.5 Comparison of Time and Space Efficiency

In this section, we report the inference time and space efficiency of our model. The inference is the process that produces the classification result when we have already trained the model. Acceleration is brought by the *xnor* and *popcount* operation with just little sacrifice on the classification performance. In this experiment, we train the binary parameters and activations of our model, then replace dot product operation between binarized matrices by *xnor* and *popcount* and also replace the dot product between binary matrix and real-valued matrix by masked-summation during the inference process.

Dataset		GAT	AS-GCN	BGN-GAT	BGN-ASGCN
Cora	Time(s)	1.9×10^{-1}	1.0×10^{-1}	1.0×10^{-2}	8.0×10^{-3}
	Space(bit)	2.46×10^{8}	3.04×10^{6}	1.32×10^{7}	1.97×10^{5}
	Accuracy	84.0%	87.3%	77.7%	84.1%
Citeseer	Time(s)	2.8×10^{-1}	2.8×10^{-1}	1.4×10^{-2}	1.8×10^{-2}
	Space(bit)	7.60×10^{6}	7.83×10^{6}	2.49×10^{5}	4.86×10^{5}
	Accuracy	72.1%	78.9%	63.7%	77.2%
Pubmed	Time(s)	3.8×10^{1}	4.54×10^{0}	1.1×10^{0}	2.1×10^{-1}
	Space(bit)	1.03×10^{6}	1.06×10^{6}	3.85×10^{4}	7.01×10^{4}
	Accuracy	78.2%	89.0%	75.7%	82.0%

Table 3.4: Comparison of performance, inference time and memory space required for the parameters between the real-valued and BGN-based models.

Table 3.4 reports the experiment results. Our model under the binarized framework is more than one order of magnitude faster than the baseline methods GAT and AS-GCN with regards to the inference time. The proposed model can be up to 29× faster and save up to 28× space compared with the baseline methods.

3.4.6 Analysis of Binarization

In this section, we introduce the effect of the estimator and binarization level with regard to the space, time and performance. We compare the space, inference time and performance between BGN-GAT and GAT on the Cora dataset. We fix the dimension of embedding vector to 64 for both methods and change the setting of BGN to show the space and time saving compared with the baseline GAT.

Result is shown in Table 3.5 where BGN^{w}, BGN^{e}, BGN^{we} and BGN^{wec} mean that the BGN is with weights binarized, embedding vectors binarized, weights and embedding vectors binarized, weights, embedding vectors and attention co-

Method	Estimator	Param space	Vec space	Speed up	Accuracy
GAT	N/A	1×	1×	1×	84.0%
BGN^{w}	STE	1/28	1×	3.7×	80.5%
BGN^{w}	Reinforce	1/28	1×	3.8×	80.3%
BGN^{e}	STE	1×	1/1.02	1.3×	81.2%
BGN^{e}	Reinforce	1×	1/1.02	1.2×	81.3%
BGN^{we}	STE	1/28	1/1.02	5.7×	77.2%
BGN^{we}	Reinforce	1/28	1/1.02	6.1×	77.5%
BGN^{wec}	STE	1/28	1/19	18.7×	77.7%
BGN^{wec}	Reinforce	1/28	1/19	19.1×	76.9%

Table 3.5: Trade-off between time/space efficiency and classification accuracy of proposed BGN w.r.t the level and setting of binarization.

efficients binarized based on the graph attention mechanism respectively. We can conclude from the Table 3.5 that (1) when the weights, activations and attention coefficients are all binarized, the BGN-GAT can save largest space for parameters and the output vectors while holding acceptable classification accuracy. (2) Straight through estimator and reinforce estimator have similar accuracy on the node classification task. Therefore, we choose the STE for our model in the above experiments because of its simplicity and certainty. (3) Compared with original GAT, BGN-GAT can save 28× space for model parameters, 19× space for activations and achieve 19× speed up.

3.4.7 Case Study

In this section, we investigate how binarized graph neural network improve the time efficiency of the GNN-based applications such as GMN. GMN is the graph matching network which is proposed to find node correspondence between two graphs. The graph matching is widely used in many important applications, such as pattern match, object detection and visual tracking. Because GMN needs to compute the pair-wise dot product between node and graph embedding

vectors, the time consumption is extremely high when the number of nodes in each graph goes up. However, with the binary representations, we can apply *xnor* between binary vectors to replace the dot product, which will alleviate the time complexity problem significantly. The following experiment results will introduce the performance and time complexity of GMN with binary node and graph representations compared with the origin version. The graph similarity will then be used for the graph matching task.

Experiment Setup

We follow the experiment setting of [58] to test the performance of Binarized GMN. The training data is generated by sampling binomial graphs G_1 with n nodes and edge probability p [25]. Then the positive example G_2 is generated by randomly substituting k_p edges from G_1 with new edges and negative example G_3 is generated by substituting k_n edges from G_1, where $k_p < k_n$. In the experiment, we set $k_p = 1$, $k_n = 2$ and $p = 0.2$. We also set the hamming similarity between vectors as loss function, which is more suitable for the binary-valued vectors as the loss function to train the model. The model needs to predict a higher similarity score for positive pair (G_1, G_2) than negative pair (G_1, G_3). The evaluation metric remains the same: (1) pair AUC - the area under the ROC curve for classifying pairs of graphs as similar or not on a fixed set of 1000 pairs and (2) triplet accuracy - the accuracy of correctly assigning higher similarity to the positive pair in a triplet than the negative pair on a fixed set of 1000 triplets.

Inference time and Graph Matching Performance

We report the graph matching accuracy and inference time of the binarized and original GMN with regards to the number of nodes in each graph. The default setting in GMN is 20 nodes per graph, which is quite small for real-world networks. We set the number of nodes in one graph from 20 to 160 and keep other settings the same as described above to evaluate the performance and

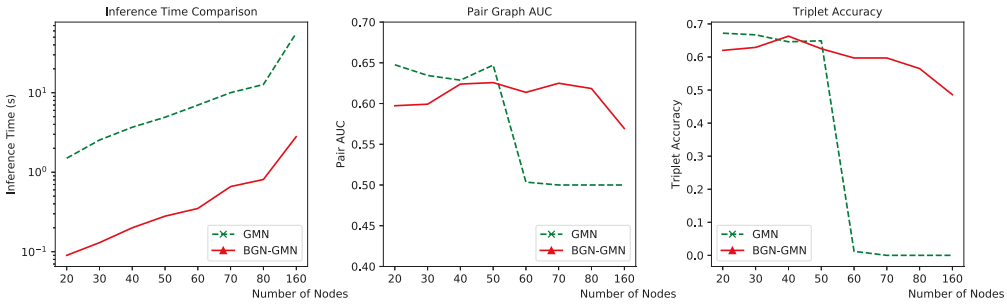
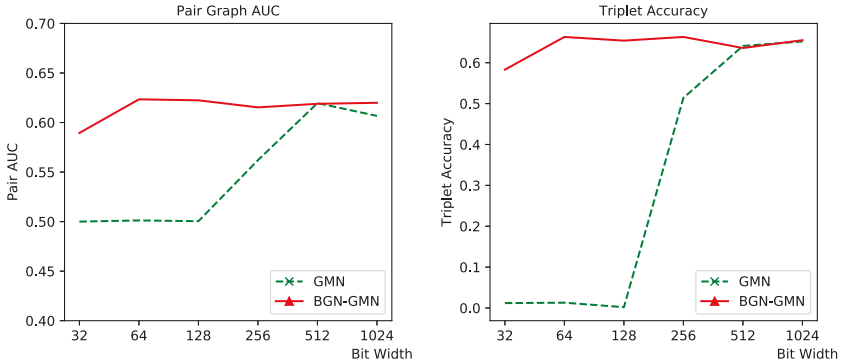


Figure 3.5: The performance of graph matching and inference time for GMN and BGN-GMN w.r.t the number of nodes per graph

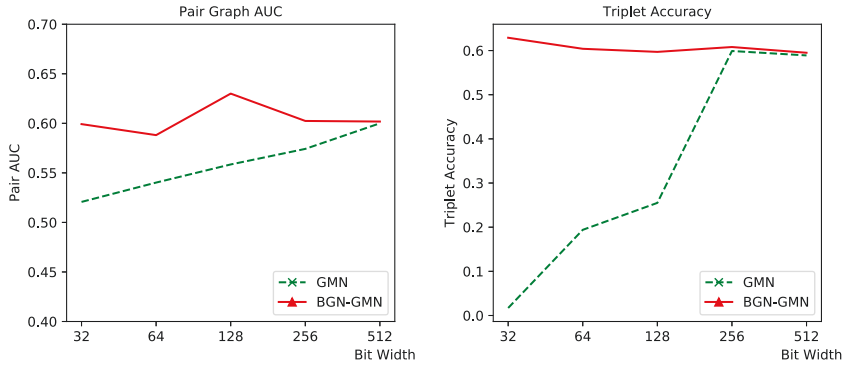
inference time. The dimensions of node and graph representations are set to 32 and 64 respectively.

As shown in Figure 3.5, the inference of BGN-GMN is significantly faster. This is because of the fact that the similarity computation (pair-wise dot product) between node representations of two graphs mainly accounts for the time complexity of GMN. Under the same dimension of node and graph embedding vectors, BGN-GMN is up to 21× faster than the baseline model in terms of the inference time with the help of the replacement of dot product by fast operations such as *xnor* and *popcount* between binary vectors.

In terms of graph matching task, the original version of GMN has better performance when the number of nodes in each graph is small. However, when the number of nodes gets larger, the pair AUC and triplet accuracy will both decay. When the number of nodes is more than 60, the real-valued representations cannot tell the similarity difference between the graphs. Hence, the model is not able to learn the different similarity scores for positive and negative pairs of graphs with the hamming similarity metric. However, with the help of binarization and balance function, the binary representations still hold an acceptable and more robust performance for the graph matching task. This is due to the fact that the binarized model produces *true* binary representations for the calculation of hamming loss and is designed for the graph matching task specifically



(a) Performance w.r.t bit width of **Graph** Representation



(b) Performance w.r.t bit width of **Node** Representation

Figure 3.6: The performance comparison of graph matching task between original version of GMN and the BGN-GMN with (a) graph representations binarized and (b) node representations binarized

on hamming space.

Parameter Sensitivity Analysis

We compare the performance of binarized and original version GMN to show the effect of dimension for node and graph embedding vectors. We set the number of nodes in each graph $n = 30$ for this comparison. We change the dimension of graph embeddings produced by two models to ensure them to produce the same bit-width embedding vectors and keep the other settings as the same to compare the performance of two models.

The result is included in Figure 3.6a. We can find that the binary graph representations tend to have better performance when they are low bit-width and have similar accuracy when the bit-width for the representations getting larger. The binary representations have more robust performance compared with the baseline model when the dimension of embedding varied.

The node representations' binarization is more important than the graph representations' because the dot product operation is mainly conducted between the node representations which costs plenty of time. The performances of GMN and BGN-GMN are compared under different bit-width for the node embedding vectors by varying the dimensions.

As shown in Figure 3.6b, the result for the pair-wise AUC is similar between the binary and the real-valued node embedding vectors, but BGN-GMN holds a better performance with low bit-width representations. As for the triplet graph accuracy, the binary embedding vector achieves better performance with short code length and similar accuracy as real-valued node embedding with long code length. These results indicate that the binary representations are much better for the comparison between two graphs under low bit-width circumstances. In line with the result of the binary graph embedding vectors, the binary node embedding vectors also have more robust performance compared with the real-valued node representations.

3.4.8 Discussion

The BGN propsoed in this chapter is a general framework which can choose any GNN as the backbone. Our propsoed BGN learns the binary code for both representations of the vertices and the parameters in the network. As illustrated in previous subsections, BGN can achieve high efficiency while sacrificing the accuracy. BGN can be extended by applying the quantify techniques which pro-

duce the low-bit width codes for representations of the vertices and the network parameters instead of the binary code. In that way, we can make a trade-off between the efficiency and accuracy by adjusting the bit-width. We remain this in our future work.

3.5 Conclusion

In this chapter, we present a model focused on the challenging problem of seeking binary representations of network embeddings using a compact neural network structure. We proposed a novel binarized graph embedding method, namely BGN, that has binarized parameters and enables GNNs to learn discrete embedding. The binarized neural network can reduce the memory and time cost of the GNN such that increases the scalability of GNNs. BGN can be naturally integrated into other GNN models to enhance the performance of the model such as graph matching network in terms of the inference time and space consumption. External experiment also illustrates that BGN can increase the time efficiency while holding competitive accuracy.

Chapter 4

Graph of Graphs Models

4.1 Chapter Overview

In this chapter, we present our models design on graph of graphs for graph classification and structured entity interaction prediction tasks. The works mentioned in this chapter are published in [104] and under revision for xxx. The rest of this chapter is organized as follows. Section 4.2 gives the formal problem definitions, input graph structures and other important theoretical definitions. Section 4.3 proposes the graph of graphs neural network designed for structured entity interaction prediction problem. Section 4.4 introduces the model details of the powerful graph neural network which has provably powerful expressive power to capture the graph structure for the representation learning and downstream applications. Section 4.5 analyzes the expressive power of the proposed PGON with careful theoretical proof. Section 4.6 experimentally evaluates the effectiveness of GoGNN and PGON on both graph classification and structured entity interaction prediction tasks. Section 4.7 gives a conclusion of this chapter.

4.2 Preliminaries

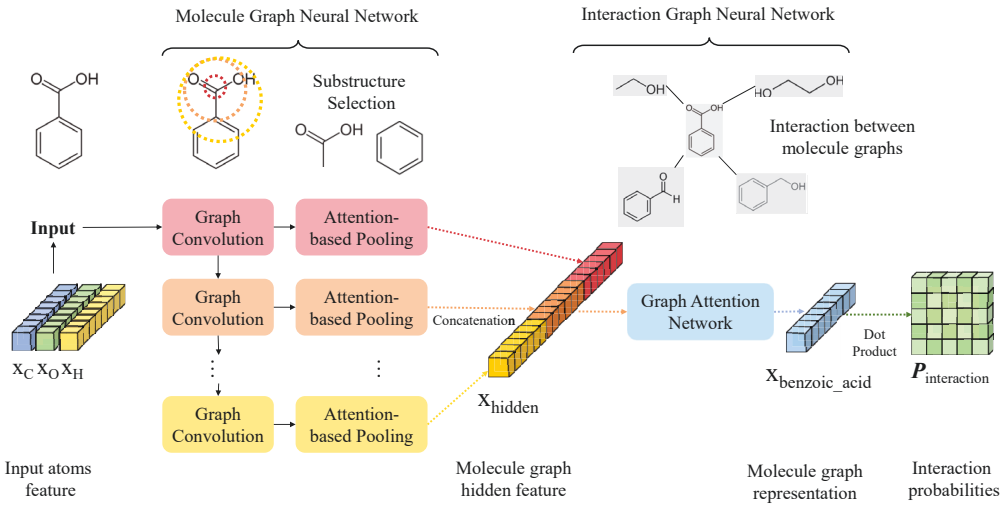


Figure 4.1: Framework of Graph of Graphs Neural Network.

4.2.1 Problem Definition

In this chapter, we focus on two representative applications of graph of graphs: graph classification and graph interaction prediction.

Graph Classification: a molecule graph g_L is called a *labeled graph* if it is labeled by a class vector $\boldsymbol{y}_L \in \{0,1\}^c$, where c denotes the number of classes. Otherwise, a local graph is *unlabeled*. Therefore, we have two subsets of graphs: unlabeled graphs G_U and labeled graphs G_{Label}. The objective of **graph classification** is to determine the class labels of the unlabeled graphs in G_U with given class labels in the labeled graphs G_{Label} and the graph of graphs topological structural information.

Graph Interaction Prediction: Given the local graph $g = (N, E)$, and graph interaction graph $G = (g, I)$, where I indicates the interactions between the

local graphs g, the objective of **graph interaction prediction** is to predict the existence of unseen interactions I_u between the graphs.

There are two main applications of graph interaction prediction: chemical-chemical interaction (CCI) prediction and drug-drug interaction (DDI) prediction. In the CCI graph, there is only one type of interaction, and our goal is to estimate the reaction probability score p_{ij} of given chemical pair (G_i, G_j). As to DDI graph which has multiple types of interactions, we aim to estimate the occurrence probabilities p_{ij}^r of side effect type r with the given triplet (G_i, r, G_j).

4.2.2 Input Graph of Graphs

Overall, the input interaction graph is regarded as graph-of-graphs as follows.

Molecule Graph. In both CCI and DDI prediction tasks, the local graphs are molecule graphs, each of which can be modeled as a heterogeneous graph with multiple types of nodes and edges. In particular, the molecule graph G_M consists of atoms $\{a_i\}$ as nodes, and edges $\{e_{ij}\}$ where e_{ij} denotes the bond between atoms a_i and a_j. Each atom (i.e., node) a is encoded as a vector $\boldsymbol{x}_a$. For each bond (i.e., edge), we assign a weight to the corresponding edge depending on the type of the bond. For example, the bond between the carbon atoms in the ethylene molecule is a double bond. Therefore, the weight of the edge e_{CC} between the carbon atoms is set to 2.

Interaction Graph. The global interaction graph G_I is formed by the molecule graphs and the interactions between them: $G_I = \{N, E_I\}$, where N denotes the node set of G_I which consists of molecule graphs $\{G_M\}$, and E_I denotes the interaction edges between the molecule graphs. Note that, in CCI graph, there is only one type of interaction between two nodes. In DDI graph, there are multiple types of side effects caused by the combination of two drugs. An attribute vector $\boldsymbol{e}^r$ is assigned for each edge e based on the side effect type r.

4.2.3 Theoretical Definitions

Definition 4.2.1. Equivariant Function. *With given matrix $\boldsymbol{X}$ and any permutation matrix $\boldsymbol{P}$, a function f is an equivariant function if $f(\boldsymbol{P}^T\boldsymbol{X}\boldsymbol{P}) = \boldsymbol{P}^T f(\boldsymbol{X})\boldsymbol{P}$.*

Definition 4.2.2. Invariant Function. *With given matrix $\boldsymbol{X}$ and any permutation matrix $\boldsymbol{P}$, a function f is an invariant function if $f(\boldsymbol{P}^T\boldsymbol{X}\boldsymbol{P}) = f(\boldsymbol{X})$.*

4.2.4 The Weisfeiler-Lehman graph isomorphism test.

The Weisfeiler-Lehman (WL) graph isomorphism test is a family of algorithms used to test the graph isomorphism. Given graph $G(V, E, d)$ where $|V| = n$, and $d : V \to \Sigma$ maps each vertex in V to the color set Σ. Two graphs are isomorphic if there exists a bijection $\phi : V \to V'$ that preserves the edge and color attached to the vertices.

There are two families of WL-test algorithms: k-WL and k-Folklore WL (FWL) algorithms. They both produce the canonical form for each graph G and construct the k-tuples of vertices with the corresponding color set, that is mapping $c : V^k \to \Sigma$. $\boldsymbol{C}$ is the tensor representing the coloring of k-tuples. For each k-tuple $v_i = (v_{i_1}, ..., v_{i_k}) \in V^k$, the corresponding color set is denoted as $\boldsymbol{C}_i \in \Sigma$, $i \in \binom{n}{k}$.

The k-WL and k-FWL test algorithms refine the coloring set $\boldsymbol{C}$ in every iteration until the split groups of k-tuples no longer change. In each step, the k-WL and k-FWL algorithms aggregate the color labels from the neighboring k-tuples to update the coloring $\boldsymbol{C}$. The aggregation steps for the algorithms to refine the coloring set of each k-tuple are shown in the following equations:

$$N_j(i) = \{(i_1, ..., i_{j-1}, i', i_{j+1}, ..., i_k) | i' \in [n]\} \tag{4.1}$$

$$N_j^F(i) = \{(j, i_2, ..., i_k), (i_1, j, ..., i_k), ..., (i_1, ..., i_{k-1}, j)\} \tag{4.2}$$

$N_j(i), j \in [k]$ is a set of n k-tuples denoting the j-th neighborhood of the selected tuple i by the k-WL algorithm. $N_j^F(i), j \in [n]$ denotes the j-th neighborhood used by k-FWL algorithm. Then both k-WL and k-FWL algorithms aggregate the coloring set using the bijective mapping from the collection of all possible tuples selected by Equations 4.1 and 4.2.

Previous works [10, 31, 32, 70] show that 1-WL and 2-WL tests have equivalent discrimination power. When $k \geq 2$, the $k+1$-WL test is more powerful than the k-WL test in detecting non-isomorphic graphs.

4.3 Graph of Graphs Neural Network

In this section, we introduce our Graph of Graphs Neural Network model.

4.3.1 Framework of GoGNN

The framework of GoGNN is illustrated in Figure 4.1. GoGNN contains molecule graph neural network which takes the atom features as input and interaction graph neural network which produces the graph representation for the prediction task. The two parts of GoGNN play a synergistic effect on improving the performance. The hidden features learned by molecule-level GNN provide the interaction-level GNN a representative initial input. The feature aggregation on the interaction-level GNN promotes the ability of molecule-level GNN to find key substructure through back-propagation.

4.3.2 Molecule Graph Neural Network

In organic chemistry, functional groups (i.e., substructures) in molecules are responsible for the characteristic chemical reactions between these molecules. For example, the reaction between benzoic acid and ethanol in Figure 1.1 is the esterification between two functional groups -COOH in benzoic acid and -OH in ethanol.

The model could achieve better performance for prediction if the model can identify the functional groups in the molecules and represent the molecule with such functional groups. Therefore, we designed our molecule graph neural network with the combination of multi-resolution architecture [111] which preserves the information of multi-hop substructures and attention-based graph pooling [56, 27] which selects the substructures to represent the molecules.

As proved in previous work [109], one single general graph convolution layer can only aggregate the feature of the node and its immediate neighbors. To obtain features of the multi-scale substructure of the molecule graph, we apply multiple layers of graph convolution operations to the input graphs. The graph convolution operation at l^{th} layer is summarized as follows

$$\begin{aligned} \boldsymbol{M}_{(l+1)} &= GCN_l(\boldsymbol{A}, \boldsymbol{M}_l) \\ GCN_l(\boldsymbol{A}, \boldsymbol{M}_l) &= \sigma(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{M}_l\boldsymbol{W}_l) \end{aligned} \tag{4.3}$$

where $\boldsymbol{M}_l \in \mathbb{R}^{n\times d}$ is the hidden feature matrix for molecule graph at l^{th} layer, $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ is the adjacency matrix with self-connection for molecule graph G_M, $\tilde{\boldsymbol{D}}$ is the diagonal degree matrix of $\tilde{\boldsymbol{A}}$ and $\sigma(\cdot)$ denotes the activation function.

Different from MR-GNN [111] which uses dual graph-state LSTMs on the input of subgraph representations, GoGNN applies graph pooling for learning the graph representation that preserves the substructure information, in order

to reduce the time and space complexity significantly. As shown in Figure 4.1, the self-attention graph pooling layer takes the output of each graph convolution layer as input to select the most representative substructures (functional groups) by learning the self-attention score $\boldsymbol{s}_l \in \mathbb{R}^{n\times 1}$ for molecule graph G_M with n atoms at l^{th} layer

$$\boldsymbol{s}_l = \sigma(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{M}_l\boldsymbol{W}_{att}^{l}) \tag{4.4}$$

where $\boldsymbol{W}_{att}^{l} \in \mathbb{R}^{d\times 1}$ is the attention weight matrix for the pooling layer to obtain the self-attention score. In order to select the most representative substructure, the graph pooling layer calculates the attention score for each atom in the graph and finds the top-$\lceil\gamma n\rceil$ atoms with the highest attention scores. We set a hyperparameter pooling ratio $\gamma \in (0, 1]$ to determine the number of nodes $\lceil\gamma n\rceil$ that are selected to represent the molecule graph

$$\begin{gathered} idx = top(\boldsymbol{s}, \lceil\gamma n\rceil), \boldsymbol{s}_{mask} = \boldsymbol{s}_{idx} \\ \boldsymbol{M}_{sel} = \boldsymbol{M} \odot \boldsymbol{s}_{mask} \end{gathered} \tag{4.5}$$

where *top* is the function that returns the indices of atoms with top $\lceil\gamma n\rceil$ attention scores as in [28]; $\boldsymbol{s}_{mask} \in \{0, 1\}^{n\times 1}$ is the mask vector determined by the attention score; $\odot$ denotes the column-wise product for masking; $\boldsymbol{M}_{sel}$ is the feature matrix of selected atoms in a molecule graph. Afterward, the readout layer, which contains mean and sum pooling, is applied on the embedding of selected atoms $\boldsymbol{M}_{sel}$ to produce the molecule graph hidden feature. After multiple graph convolutional and self-attention graph pooling layers, we got several graph hidden features. Once obtained, we concatenate the outputs of the graph pooling layers as the hidden feature vector $\boldsymbol{x}_{G_M}$ for the molecule graph. Because the hierarchical graph pooling architecture is applied, the graph representation can preserve the multi-hop substructure information effectively.

Hence, GoGNN can identify the function groups which play the key roles in molecule interactions and use these functional groups to represent the molecule graph.

4.3.3 Interaction Graph Neural Network

Most of existing CCI and DDI prediction models train the model with the input of pair of molecule graphs, but ignore the molecule interaction graph. However, the information of interaction graph is crucial for the interaction prediction because it enables the model to capture high-order interaction relationship and enhance the model's ability to capture the representative molecular substructures synergistically.

We have the following observations that motivate us to perform graph neural network on the interaction graph: Firstly, the type of interaction is dependent on the type of involved molecules. As mentioned in Section 4.3.2, esterification is the reaction between -OH in alcohols and -COOH in carboxyl acids. The neighbor aggregation of GNN can gather the neighbor information that helps to summarize the types of chemicals that interact with the selected one. Secondly, it is necessary to assign importance score to the neighbors for molecules in the interaction graph, since the chemical interactions have different significance and frequency. For example, vitamin C has two main properties: reducibility and acidity. Therefore, vitamin C cannot be prescribed with oxidizing drugs like vitamin K1 and alkaline drugs like omeprazole. In an uncommon case, vitamin C reduces the therapeutic effect of inosine because of their complex physical and chemical reactions. Therefore, we apply the graph attention network in order to preserve the frequencies of the chemical reactions and reduce the influence of biased observation of the interaction graph. As for the DDI graph with edge attributes, an edge-aggregation graph neural network is applied.

Graph Attention Network. The attention-based graph neural network [100] is applied on the interaction graph without edge attributes. With the learned molecule hidden feature vector $\boldsymbol{x}_{G_M}$ and interaction graph $G_I = \{N, E_I\}$ as input, molecule graph representations are calculated by the neighbor aggregation on the interaction graph as follows

$$\boldsymbol{x}_{G_i}^{l+1} = \|_{\kappa=1}^{K} \sigma(\sum_{j \in \eta_{G_i}} \alpha_{ij}^{\kappa} \boldsymbol{W}_{\kappa}^{l} \boldsymbol{x}_{G_j}^{l}) \tag{4.6}$$

where K is the number of attention heads, σ is a nonlinearity function, $\boldsymbol{W}_{\kappa}^{l}$ is the weight matrix at κ^{th} attention head in l^{th} layer and η_{G_i} is the set of neighbor molecule graphs of G_i in the interaction graph G_I. Notation α_{ij}^{κ} is the attention coefficient between G_i and G_j which is calculated by the following equation:

$$\alpha_{ij} = \frac{exp(\text{LeakeyRelu}(\boldsymbol{a}[\boldsymbol{W}\boldsymbol{x}_{G_i}] \parallel [\boldsymbol{W}\boldsymbol{x}_{G_j}]))}{\Sigma_{n \in \eta_{G_i}} exp(\text{LeakeyRelu}(\boldsymbol{a}[\boldsymbol{W}\boldsymbol{x}_{G_i}] \parallel [\boldsymbol{W}\boldsymbol{x}_{G_n}]))} \tag{4.7}$$

where $\boldsymbol{a}$ is a learnable attention weight vector and $\parallel$ is the concatenation operation.

Edge Aggregation Network. In DDI graph, each edge has an attribute vector $\boldsymbol{e}_{ij}^{r}$ which is determined by the side effect type r of the drug combination (G_i, G_j). To capture the edge attributes [88], we propose an edge aggregation network that aggregates the neighbor information together with edge attribute:

$$\boldsymbol{x}_{G_i}^{l+1} = \sigma(\boldsymbol{W}^{l} \boldsymbol{x}_{G_i}^{l} + \sum_{r} (\sum_{G_j \in \eta_{G_i}^{r}} \boldsymbol{x}_{G_j}^{l} \cdot h_{\boldsymbol{W}_{\boldsymbol{e}}}(\boldsymbol{e}_{ij}^{r}))) \tag{4.8}$$

where $h_{\boldsymbol{W}_{\boldsymbol{e}}}$ is the MLP layer with linear transformation matrix $\boldsymbol{W}_{\boldsymbol{e}}$ which transforms the edge attribute vector $\boldsymbol{e}_{ij}^{r} \in \mathbb{R}^{h \times 1}$ into a real number $\tau_{ij}^{r} \in \mathbb{R}$. In this way, GoGNN aggregates node's neighbor information together with edge attributes. Different from Decagon [121] which sets side-effect-specific parameters,

GoGNN shares the parameters for all types of side effects in order to improve the robustness and generalization of the model.

4.3.4 GoGNN Model Training

We optimize the parameters with the task-specific loss functions.

Chemical Interaction Prediction. Since there is no edge attribute in the graph, we regard the chemical interaction prediction as a link prediction problem. The dot product of two graph representations is used as the link probability of two graphs:

$$p_{ij} = \sigma(\boldsymbol{x}_{G_i}^T \cdot \boldsymbol{x}_{G_j}) \tag{4.9}$$

where σ is the activation function such as *sigmoid* function that ensures $p_{ij} \in (0, 1)$. To encourage the model to assign higher probabilities to the observed edges than the random non-edges, we follow the previous study and estimate the model through negative sampling. For each positive edge pair (G_i, G_j), a random negative edge (G_i, G_m) is sampled by choosing a molecule graph G_m randomly. We optimize the model using the following cross-entropy loss function

$$\mathcal{L}_{CCI} = \sum_{(G_i,G_j)\in G_{CCI}} -log(p_{ij}) - \mathbb{E}_{m\sim P_j} log(1 - p_{im}) \tag{4.10}$$

Drug Interaction Prediction. The drug-drug interaction prediction task is regarded as a multirelational link prediction problem. Inspired by the loss design in [121], we train the parameters with the following cross-entropy loss function

$$p_{ij}^r = \sigma((\boldsymbol{W}_r \boldsymbol{x}_{G_i})^T \cdot (\boldsymbol{W}_r \boldsymbol{x}_{G_j})) \tag{4.11}$$

$$\mathcal{L}_{ij}^r = -log(p_{ij}^r) - \mathbb{E}_{m\sim P_j^r} log(1 - p_{im}^r) \tag{4.12}$$

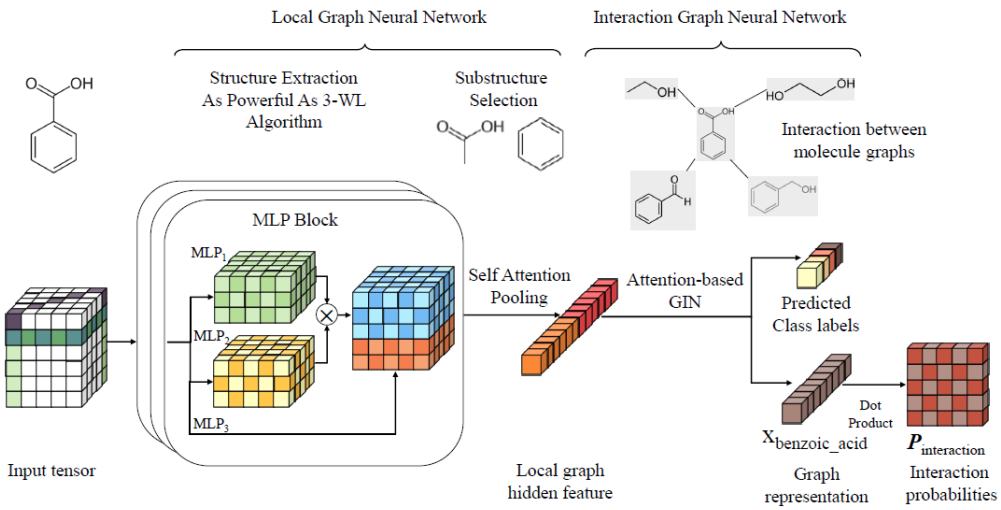


Figure 4.2: Framework of Powerful Graph of Graphs Neural Network.

$$\mathcal{L}_{DDI} = \sum_{(G_i,r,G_j)\in G_{DDI}} \mathcal{L}_{ij}^{r} \tag{4.13}$$

where $\boldsymbol{W}_r$ is the side-effect-specific weight for linear transformation of $\boldsymbol{x}_{G_i}$ w.r.t. the side effect type r. Given observed triplet (G_i, r, G_j), the negative sample is chosen by replacing G_j with randomly selected graph G_m according to sampling distribution P_j^r [75].

4.4 Powerful Graph of Graphs Neural Network

In this section, we introduce details of the powerful graph of graphs neural network (PGON). The framework of the model, graph of graphs learning architecture and the training objectives are discussed in this section.

4.4.1 Framework

PGON is a graph neural network model that takes the graph of graphs structure and feature information as input, and then produces the predicted class labels or predicts the interaction probabilities between the graphs in an end-to-end manner. The framework of PGON is shown in Figure 4.2. The model is formed by two graph neural networks. A local graph neural network whose input is the atom or amino acid features. An interaction graph neural network which eventually produces a representation of graph for down-stream tasks, i.e., graph classification and graph interaction prediction. Local graph neural network learns the hidden features which serve as the initial input for the interaction graph neural network. The hidden feature is representative since we exploit the GNN that has the same expressive power as 3-WL tests. The interaction-level graph neural network promotes the ability of local graph neural network to find the key substructure through training process by the feature aggregation mechanism.

4.4.2 Local Graph Neural Network

The input local graphs are chemical or protein graphs. In these kinds of graphs, the structure of the graph is very important for the analysis task. For example, the reactions between the organic chemical molecules are mainly decided by some specific subgraphs: functional groups. Reactions between organic acid and ethanol are intrinsically controlled by two functional groups, *i.e.,* the carboxy group (-COOH) and hydroxy group (-OH). As for the proteins, the structures of proteins, especially the sequences and structures of amino-acids, are also responsible for the characteristic properties of the proteins. Though currently we cannot model the 3D structure of proteins using graph neural networks, the discriminative GNN model could preserve the plain structures of the proteins which

are the most important for protein properties.

Based on the above motivations, we design the local graph neural network which is discriminative enough to distinguish graph structures. Therefore, we use the graph neural network proposed by Maron et al. [70]. The GNN model has 3-WL discrimination power and low computation complexity. The model is formed by the invariant layer and several MLP blocks:

$$F = m \circ h \circ M_k \circ M_{k-1} \circ ... \circ M_1, \tag{4.14}$$

where m is a transformation layer such as MLP, h is an invariant layer and $M_1, ..., M_k$ are the MLP-based blocks described below. Each block has three MLPs, two of them (MLP_1 and MLP_2) map the tensor dimensions $\mathbb{R}^{n\times n\times d_0} \rightarrow \mathbb{R}^{n\times n\times d_1}$, and the other transforms the tensor dimensions $\mathbb{R}^{n\times n\times d_0} \rightarrow \mathbb{R}^{n\times n\times d_2}$. The detail of the input tensor is introduced in Section 4.5. The tensors produced by the MLP_1 and MLP_2 are in the same dimensions. These two tensors are multiplied to produce the new feature tensor. The computation process for each MLP block is shown in the following equations:

$$\boldsymbol{X}_1 = MLP_1(\boldsymbol{X}) \tag{4.15}$$

$$\boldsymbol{X}_2 = MLP_2(\boldsymbol{X}) \tag{4.16}$$

$$\boldsymbol{X}_3 = MLP_3(\boldsymbol{X}) \tag{4.17}$$

$$\boldsymbol{X}_4 = \boldsymbol{X}_1 \otimes \boldsymbol{X}_2 \tag{4.18}$$

$$\boldsymbol{X}_{out} = \boldsymbol{X}_3 \oplus \boldsymbol{X}_4, \tag{4.19}$$

where $\otimes$ indicates the matrix multiplication between the corresponding matrices: $\boldsymbol{X}_{1(:,:,j)}$ and $\boldsymbol{X}_{2(:,:,j)}$ and $\oplus$ is the concatenation of the tensors. $\boldsymbol{X}_{out}$ is the output feature tensor for every graph after the MLP block. Different from

MR-GNN [111] which uses dual graph-state LSTMs, PGON exploits pooling-based graph model to learn the representation that preserves the substructure information of each graph. Therefore, PGON enjoys lower time and space complexity compared to MR-GNN. In the model of PGON shown in Figure 4.2, the most representative substructures are selected in the self-attention graph pooling layer by learning the self-attention score $\boldsymbol{s} \in \mathbb{R}^{n\times 1}$ for the local graph G_L with n vertices.

$$\boldsymbol{s} = \sigma(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{X}_{feat}\boldsymbol{W}_{att}) \tag{4.20}$$

where $\boldsymbol{W}_{att} \in \mathbb{R}^{d\times 1}$ denotes the weight matrix in the self-attention pooling layer, and $\boldsymbol{X}_{feat}$ denotes the feature vectors of the graph on the diagonal of the output tensor $\boldsymbol{X}_{out_{i,i,:}}$. After the calculation of attention score in the graph pooling layer, the most representative substructure is selected by finding the top-$\lceil\gamma n\rceil$ vertices with the highest attention scores. A hyperparameter $\gamma \in (0, 1]$ to is used as the pooling ratio to determine the number of vertices $\lceil\gamma n\rceil$ that are selected.

$$\begin{aligned} idx = top(\boldsymbol{s}, \lceil\gamma n\rceil), \boldsymbol{s}_{mask} = \boldsymbol{s}_{idx} \\ \boldsymbol{X}_{sel} = \boldsymbol{X} \odot \boldsymbol{s}_{mask} \end{aligned} \tag{4.21}$$

where *top* denotes the function selecting the vertices that have top $\lceil\gamma n\rceil$ attention scores as in [56]; $\boldsymbol{s}_{mask}$ denotes a mask vector decided by the attention score; $\odot$ is the column-wise product for masking; and $\boldsymbol{X}_{sel}$ denotes the feature vectors of the chosen nodes within a local graph. Afterward, the readout layer, which contains sum and mean pooling, is performed on the representations of the chosen nodes $\boldsymbol{X}_{sel}$ to learn the graph-level hidden feature vector $\boldsymbol{x}_{sel}$ for the diagonal elements from the tensor $\boldsymbol{X}_{out}$. As for the off-diagonal elements in the tensor $\boldsymbol{X}_{out}$, max pooling is applied to produce the hidden feature $\boldsymbol{x}_{off} \in \mathbb{R}^{d_{off}}$

for the local graph. The hidden feature $\boldsymbol{x}_{feat}$ is the concatenation of $\boldsymbol{x}_{sel}$ and $\boldsymbol{x}_{off}$. Hence, PGON can identify the substructures which contribute significantly in graph interactions, and represent the local graph using these substructures.

4.4.3 Interaction Graph Neural Network

In the graph analysis tasks, the interactions between the graphs are always ignored. However, it is crucial for graph analysis to obtain the information of the interaction graph because the information enables the model to capture high-order interactivity relationships. Therefore, PGON has a better ability to preserve the characteristic molecule substructures synergistically which benefits both graph classification and graph interaction prediction tasks.

We perform the GNN on the interaction graph based on the following observations: firstly, the type of the involved molecules in an interaction determines the type of interaction. For example, Diels-Alder reaction is a kind of organic chemical reaction between a conjugated diene and a substituted alkene. Specifically, the Diels-Alder reaction is between the two double bonds in the conjugated diene and the double bond in the substituted alkene. Therefore, we use the graph neural network to aggregate the neighbor information, and eventually conclude the types of molecules involved in the interaction graph. Secondly, in order to model the frequency and significance of the interactions, the significance score is assigned to each interaction of the graph. For instance, reducibility and acidity are main properties of vitamin C. Because of the properties, vitamin C can never be taken with procarbazine and sucralfate due to their oxidization and alkalinity respectively. In a much rarer situation, VC reduces the therapeutic effect of inosine due to their multiplex reactions. As a result, an attention-based graph isomorphism network [110] is applied to preserve the significance and importance of the graph interactions, reduce the influence of biased observations

of the interaction graph and maintain the representation power of the model. Specifically, the edge-aggregation GNN is exploited for the DDI graph with edge attributes.

Attention-based GIN. We add the attention coefficients on the original GIN [110] to assign importance values to graph interactions. With the output local graph hidden feature vector $\boldsymbol{x}_{G_L}$ and interaction graph $G_I = \{N, E_I\}$ as the input, the attention-based graph isomorphism network perform the neighbor aggregation as follows:

$$\boldsymbol{x}_{G_i}^k = MLP^k((1+\epsilon^k)\boldsymbol{x}_{G_i}^{k-1} + \sum_{G_j \in \mathcal{N}(G_j)} a_{ij}^k \boldsymbol{x}_{G_j}^{k-1}). \tag{4.22}$$

In the equation, ϵ is a learnable parameter to adjust the importance of the graph representation, and a_{ij}^k is the attention coefficient between two graphs G_i and G_j. The attention coefficient is calculated by the following equation:

$$a_{ij} = \frac{exp(\text{LeakeyRelu}(\boldsymbol{a}[\boldsymbol{W}\boldsymbol{x}_{G_i}] \parallel [\boldsymbol{W}\boldsymbol{x}_{G_j}]))}{\sum_{n \in \eta_{G_i}} exp(\text{LeakeyRelu}(\boldsymbol{a}[\boldsymbol{W}\boldsymbol{x}_{G_i}] \parallel [\boldsymbol{W}\boldsymbol{x}_{G_n}]))} \tag{4.23}$$

where $\boldsymbol{a}$ denotes a learnable vector indicating the attention weights and $\parallel$ denotes the concatenation operation.

Edge Aggregation Network. In the drug-drug interaction graph, each edge has an attribute vector $\boldsymbol{e}_{ij}^r$ which is determined by the side effect type r of the drug combination (G_i, G_j). To capture the edge attributes [88], an edge aggregation GNN is proposed. The network aggregates the neighbor information together with the edge attribute:

$$\boldsymbol{x}_{G_i}^{l+1} = \sigma(\boldsymbol{W}^l \boldsymbol{x}_{G_i}^l + \sum_r (\sum_{G_j \in \eta_{G_i}^r} \boldsymbol{x}_{G_j}^l \cdot h_{\boldsymbol{W_e}}(\boldsymbol{e}_{ij}^r))) \tag{4.24}$$

where $h_{\boldsymbol{W_e}}$ denotes the multi-layer perceptron with matrix $\boldsymbol{W_e}$ which linearly

transforms the edge feature representation $\boldsymbol{e}_{ij}^r \in \mathbb{R}^{h\times 1}$ into a real number $\tau_{ij}^r \in \mathbb{R}$. PGON shares the parameters for all types of side effects which is different from Decagon [121] whose parameters are set for each side effect specifically. Therefore, PGON is the model that has better generalization and robustness.

4.4.4 PGON Model Training

The parameters in the model are optimized by task-specific loss functions.

Graph Classification. We apply MLPs with activations to transfer the learned graph representations $\boldsymbol{x}_{G_i}$ into the labels:

$$y_{G_i} = \sigma(W^l \cdot \sigma(\cdots W^1 \cdot \boldsymbol{x}_{G_i})), \tag{4.25}$$

where σ denotes the activation function such as softmax, and W is the weight matrix in MLP. With the obtained class labels, the cross-entropy is proven to be a good choice to measure the loss of the predicted labels:

$$\mathcal{L}_{class} = \sum_{G_i \in G_{labeled}} \mathcal{J}(\hat{y}_{G_i}, y_{G_i}), \tag{4.26}$$

where $\mathcal{J}$ is the cross-entropy loss and $\hat{y}_{G_i}$ is the ground truth labels of the labeled graphs. The classification model is trained by minimizing the above loss function.

Chemical Interaction Prediction. The CCI prediction is modeled as a link prediction task. We simulate the probability of linkage between two graphs by the dot product of two graph embedding vectors.

$$p_{ij} = \sigma(\boldsymbol{x}_{G_i}^T \cdot \boldsymbol{x}_{G_j}) \tag{4.27}$$

where σ denote the activation function such as *tanh* which constraints $p_{ij} \in (0, 1)$. It is natural that the existing edges should have higher significance scores

compared to the random non-edges. For this purpose, negative sampling is utilized. Given a positive edge pair (G_i, G_j), a random negative edge (G_i, G_m) is sampled by selecting a local graph G_m randomly. The cross-entropy loss function is used for the optimization of the model

$$\mathcal{L}_{CCI} = \sum_{(G_i,G_j)\in G_{CCI}} -log(p_{ij}) - \mathbb{E}_{m\sim P_j} log(1 - p_{im}) \tag{4.28}$$

Drug Interaction Prediction. Because there are many kinds of adverse effects in drug-drug interaction graph. The DDI prediction task is modeled as a multirelational link prediction problem. The following cross-entropy loss function is utilized to optimize the parameters in this task

$$p_{ij}^r = \sigma((\boldsymbol{W}_r \boldsymbol{x}_{G_i})^T \cdot (\boldsymbol{W}_r \boldsymbol{x}_{G_j})) \tag{4.29}$$

$$\mathcal{L}_{ij}^r = -log(p_{ij}^r) - \mathbb{E}_{m\sim P_j^r} log(1 - p_{im}^r) \tag{4.30}$$

$$\mathcal{L}_{DDI} = \sum_{(G_i,r,G_j)\in G_{DDI}} \mathcal{L}_{ij}^r \tag{4.31}$$

where $\boldsymbol{W}_r$ is a weight matrix for the linear transformation of $\boldsymbol{x}_{G_i}$ w.r.t. the side effect type r. For an existing triplet (G_i, r, G_j) denoting two drug graphs G_i and G_j which cause a side effect r, we choose the negative sample G_m randomly according to sampling distribution P_j^r [75] and replace G_j by G_m during the training process.

4.5 Analysis

In this section, we analyze the expressive power of PGON. We prove that our model is more powerful than other graph of graphs neural network.

4.5.1 Framework of PGON

The framework of PGON shown in Equation 4.14 can distinguish the non-isomorphic graphs. This simple structure with multiple MLP blocks can achieve great expressive power equivalents to the 3-WL test. In this subsection, we introduce how PGON implements the 2-FWL graph isomorphism test algorithm which has the equivalent expressive power as the 3-WL test algorithm.

Theorem 1. *Given two graphs $G = (V, E, x)$ and $G' = (V', E', x')$. If these two graphs can be distinguished by the 3-WL isomorphism test, there exists a graph neural network F so that $F(G) \neq F(G')$*

Proof. To prove the framework of the proposed PGON has the 3-WL test expressive power, we show that the model can implement the 2-FWL algorithm which has the equivalent distinguish power as 3-WL isomorphism test algorithm.

With the given $G = (V, E, x)$, the input tensor is built as $\boldsymbol{X} \in \mathcal{R}^{n^2 \times (e+2)}$, where $\boldsymbol{X}_{:,:,:,e+1}$ encodes the adjacency matrix of G and $\boldsymbol{X}_{i,i,1:e}$ encodes the feature vector of each vertex $v_i \in V$, the elements outside the diagonal $\boldsymbol{X}_{j,k,1:e}, j \neq k$ are all zeros, and $\boldsymbol{X}_{:,:,:,e+1}$ is the identity matrix. First, we need to build the 2-FWL initialization that colors the 2-tuples by their isomorphism type. With the given adjacency matrix $\boldsymbol{A} = \boldsymbol{X}_{:,:,:,e+1}$, and input features $\boldsymbol{H} = \boldsymbol{X}_{:,:,:,1:e}$, the tensor $\boldsymbol{C} \in \mathcal{R}^{n^2 \times (4e+1)}$ is constructed using the following colors matrices. $\boldsymbol{A} \cdot \boldsymbol{H}_{:,:,j}$, $(\mathbf{1}\mathbf{1}^T - \boldsymbol{A}) \cdot \boldsymbol{H}_{:,:,j}$, $\boldsymbol{H}_{:,:,j} \cdot \boldsymbol{A}$, $\boldsymbol{H}_{:,:,j} \cdot (\mathbf{1}\mathbf{1}^T - \boldsymbol{A})$, where $j \in [e]$ and $\mathbf{1}\mathbf{1}^T - \boldsymbol{A}$ is the adjacency matrix of the complement graph. The last channel indicates whether

$v_{i_1} = v_{i_2}$.

To follow the 2-FWL update steps, we perform each update step in the following way. As mentioned in Section 4.2.4, when the matrix $\boldsymbol{B} \in \mathcal{R}^{n \times 2a}$ is given, where $\boldsymbol{B}_{j;:} = (\boldsymbol{X}_{j;i_2;:}, \boldsymbol{X}_{i_1;j;:})$. We would like to compute the output tensor $\boldsymbol{X}^{out} \in \mathcal{R}^{n^2 \times b}$, where $\boldsymbol{X}^{out}_{i_1,i_2,:}$ has been proven in [70] that can be calculated by the network based on the MLPs.

$$\boldsymbol{X}^{out}_{i_1,i_2,l} := \sum_{j=1}^{n} m_1(\boldsymbol{X}_{j;i_2;l}) m_2(\boldsymbol{X}_{i_1;j;l}) \tag{4.32}$$

After this, we concatenate X and X^{out} to pair the multiset with the input color of each k-tuple.

The block introduced above could perform the 2-FWL test update. With sufficient update steps we can produce the tensor that can be used to distinguish the graphs. Our final goal is to calculate a representation tensor for each graph that indicates the histogram of its k-tuples' color. In [70], PPGN applies invariant max pooling-based operator by concatenating the max values among the diagonal and off-diagonal elements on $\boldsymbol{X}^{out}$ to produce the graph representations. The expressive power of the representations is proven both theoretically and empirically in [70]. □

4.5.2 Permutation invariance of pooling and GIN operators

Though the operator in PPGN is invariant which can ensure expressive power, we find that simply applying the operator has a limitation in representing the importance of the nodes in the local graph. Therefore, the concatenation of max pooling on the off-diagonal elements and the self attention pooling mentioned in Equation 4.21 on the diagonal elements is used to produce the representations

for the local graphs. Note that when the pooling ratio $\gamma = 1$, i.e., all the diagonal elements are used for the self attention pooling, our pooling operator is same as that in PPGN. Our pooling method can filter out the nodes with low importance in graph analytic tasks by adjust the pooling ratio while preserving expressive power. Then, we apply the attention-based graph isomorphic neural network to further gather information from the neighbors in the interaction graph to enhance the performance of our model. In this subsection, we prove the permutation invariance of self-attention-based pooling and the attention-based graph isomorphic neural network which are used to calculate the graph representations. In the following, we prove the invariance of this operator $F_{GIN}(F_{pool}(\boldsymbol{X}_{out}))$.

Corollary 1. *For two mapping functions f, g, if f and g are invariant, $f \circ g$ is invariant.*

Proof. Given invariant mapping f and g, by definition, we have $f(\boldsymbol{X}) = f(\boldsymbol{P}_1^T \cdot \boldsymbol{X} \cdot \boldsymbol{P}_1)$ and $g(\boldsymbol{X}) = g(\boldsymbol{P}_2^T \cdot \boldsymbol{X} \cdot \boldsymbol{P}_2)$, where $\boldsymbol{P}_i$ denotes the permutation matrix. The following result is immediate.

$$\begin{aligned} f(\boldsymbol{P}_1^T \cdot g(\boldsymbol{P}_2^T \cdot \boldsymbol{X} \cdot \boldsymbol{P}_2) \cdot \boldsymbol{P}_1) &= f(\boldsymbol{P}_1^T \cdot g(\boldsymbol{X}) \cdot \boldsymbol{P}_1) \\ &= f(g(\boldsymbol{X})) = f \circ g(\boldsymbol{X}). \end{aligned} \tag{4.33}$$

□

Therefore, we can prove the permutation invariance of the pooling operation and GIN separately. The proof is also immediate. The pooling operation is based on the max pooling which is not sensitive to the node orders, hence, the representation for each graph is independent of the order of the nodes in the graph. The graph isomorphic neural network in Equation 4.22 is a message passing-based neural network which sums up the representations from the neighboring nodes with attention coefficients. It is obvious that GIN is independent of the order of

nodes. Therefore, we proved the permutation invariance of the pooling operator and attention-based GIN, which eventually infers the permutation invariance of the calculation operator formed by these two parts.

Finally, we have proved that the PGON with the framework in Equation 4.14 has the expressive power equivalent to 2-FWL and 3-WL test.

4.6 Experiment

In this section, we introduce the extensive experiment results that demonstrate the effectiveness and robustness of GoGNN and PGON.

4.6.1 Dataset

We evaluate our models on three real-life chemical interaction and protein interaction graphs. The local graphs in these three datasets are the chemicals or the proteins. The global graphs contain the interactions between the chemicals and the proteins. The datasets used are shown as follows:

- **Proteins** [9] is a benchmark graph dataset where vertices are elements within protein molecules and edges indicate that two nodes are neighbors in the amino-acid sequence or in 3D space.
- **D&D** [23] is a set of structures of enzymes and non-enzymes proteins, where nodes are amino acids, and edges represent spatial closeness between nodes.
- **MUTAG** [20] is a well-known graph classification benchmark dataset. Different from the preprocessed data in [45], we build the interaction graph with given chemicals based on the interaction score in the CCI dataset, and build more detailed molecule graphs using the description in [20].

We build the interaction graphs on the **Proteins** and **D&D** dataset according to the interaction relationship in the PPI dataset. Two proteins are linked in the interaction graph if they are connected or at least have two common neighbors in the PPI network.

To test the performance of our model on chemical-chemical interaction and drug-drug interaction prediction tasks, following datasets are chosen for the experiments:

CCI. The CCI dataset[1] assigns a score from 0 to 999 to describe the interaction probability where a higher score indicates higher interaction probability. According to threshold score, we get two datasets with chemical interaction probability score over 900 and 950: CCI900 and CCI950. CCI900 has 14343 chemicals and 110078 chemical interaction edges, and CCI950 has 7606 chemicals and 34412 chemical interaction edges.

DDI. For the drug-drug interaction prediction problem, DDI dataset[2] and the side effect dataset SE[3] [121] are used. The DDI dataset is proposed by DeepDDI [85] which contains 86 types of side effects, 1704 drugs and 191400 drug interaction edges. SE dataset is the integration of SIDER (Side Effect Resource), OFFSIDES and TWOSIDES database. To familiarize the comparison, we use the preprocessed data used by Decagon [121]. Therefore, the SE dataset contains 645 drugs, 964 types of side effects and 4651131 drug-drug interaction edges. A vector representation $\boldsymbol{se}_r \in \mathcal{R}^{128}$ is assigned to each side effect type produced by pre-trained BERT model [21].

The molecules are transformed from the SMILE strings [106] into graphs by the open-source rdkit [53]. An initial feature vector $\boldsymbol{x}_a \in \mathcal{R}^{32}$ is assigned

[1] `http://stitch.embl.de/download/chemical_chemical.links.detailed.v5.0.tsv.gz`

[2] `https://www.pnas.org/content/suppl/2018/04/14/1803294115.DCSupplemental`

[3] `http://snap.stanford.edu/decagon`

for every atom. The edges in molecule graphs are weighted by the type of the bonds.

4.6.2 Baselines

For the graph classification task, the baseline models include graph neural network based classifiers and GNN-based approaches from the graph of graphs perspective:

- **GIN** [110] is the model that is as expressive as the Weisfeiler-Lehman graph isomorphism test.
- **PPGN** [70] is the simple and scalable GNN model that has a provable 3-WL expressive power. PPGN interleaves applications of standard Multilayer-Perceptron (MLP) applied to the feature dimension and matrix multiplication.
- **ISONN** [73] is a GNN-based model that contains two main components: graph isomorphic feature extraction component and classification component.
- **SEAL-CL** [57] is the neural network on hierarchical graphs for graph classification.
- **GoGNN** [104] is the graph neural network model built on the hierarchical graph structure, which focuses on the entity interaction prediction task.
- **DGCN** [38] is the dual graph convolutional network that learns compound representations from both the compound graphs and the inter-compound network in an end-to-end manner.

For the interaction prediction task, the following state-of-the-art baseline methods are compared:

- **DeepCCI** [49] is the CNN based model for predicting the interactions between the chemicals.
- **DeepDDI** [85] is the model designs a feature called structural similarity profile(SSP) combined with traditional MLP for DDI prediction.
- **MR-GNN** [111] is an end-to-end graph neural network with multi-resolution architecture that produces interaction between pairs of chemical graphs.
- **MLRDA** [14] is the multitask, semi-supervised model for DDI prediction.
- **SEAL-CL** [57] is the neural network on hierarchical graphs for graph classification.
- **DGCN** [38] and **GoGNN** [104].

We used the public code of the baselines and keep the settings of the models the same as discussed in the original papers. We modified the code of GoGNN and DGCN for the classification tasks. SEAL-CL is reimplemented for the interaction prediction task.

The evaluation settings in [57] are used for the classification task, and 10-fold cross validation is performed. The average accuracy is reported. For the CCI and DDI prediction task, the settings in GoGNN [104] are used to familiarize the comparison. We detail settings in the following subsections.

Ablation Study

To investigate how the graph of graphs architecture and dual-attention mechanism improve the performance of the proposed model, we conduct the ablation study on the following variants of GoGNN:

	Proteins	**D&D**	**MUTAG**
GIN	76.28%	79.91%	89.41%
PPGN	77.20%	79.97%	90.55%
ISONN	75.12%	76.28%	88.79%
SEAL-CL	76.89%	80.10%	88.17%
GoGNN	74.62%	77.69%	88.53%
DGCN	75.27%	78.94%	87.91%
PGON	78.87%	81.54%	91.15%

Table 4.1: Accuracy of the graph classification task.

GoGNN-M is the variant which only learns the representations for the molecule-level graphs without the graph convolution on the interaction graph. An MLP layer is applied with the input of molecule-level graph representations for the graph interaction prediction task.

GoGNN-I only conducts graph convolution operation on the chemical interaction graphs. The initial molecule representations are the sum pooling of the atom representations within the molecule.

GoGNN-noPool replaces the self-attention pooling on the molecule graph by the concatenation of conventional mean pooling and sum pooling.

GoGNN-noAttn replaces the attention-based neural network on the interaction graph by a conventional GCN.

4.6.3 Classification Result

Settings. To familiarize the comparison, we divided the dataset on a 90%-10% basis for training and testing respectively. The dimensions for the graph hidden feature and output graph representations are set to 384 and 256 respectively. For the classification tasks, the pooling ratio is set to 1, which means all the node representations are used to produce the graph hidden feature. The average accuracy for classification is reported for evaluation.

Results. As shown in Table 4.1, PGON outperforms the other baseline models on all three datasets. The superior experiment results indicate that the interaction network information integrated in the PGON enhances the classification accuracy significantly compared to the models which only use the graph feature and structural information. Further, on the real-world chemical and biological dataset, the heterogeneous nature of the molecule graphs is usually ignored, and the molecule graphs are always modeled as a homogeneous graph. Our preprocess procedure assigns a simple but effective one-hot feature to each kind of atom in the molecule graphs, which also helps PGON achieve better performance. PGON also outperforms GoGNN and DGCN which shows the improvement in classification accuracy as a result of the higher expressive power of PGON.

4.6.4 CCI Prediction Results

Settings. Following the previous study, we divide the CCI datasets into training and testing set with ratio 9:1, and randomly choose 10% data for validation. The dimensions of molecule graph hidden feature, and the output molecule graph representation are set to 384, 256, respectively. We set the learning rate to 0.01 and the pooling ratio to 0.5. To evaluate the performance, we choose *area under the ROC curve*(AUC) and *average precision score*(AP) as metrics.

Results. As shown in Table 4.2, GoGNN outperforms all the other state-of-the-art baseline methods on the CCI prediction task. The improvement indicates that, compared with the methods that only train the parameters with pair-wise or individual chemical inputs, GoGNN can preserve more useful information on different scales by the feature extraction and aggregation through the graph of graphs. The dual-attention mechanism also helps the model to learn higher quality graph representations by identifying and preserving the importance of molecular substructures and chemical interactions.

	CCI900		**CCI950**	
	AUC	AP	AUC	AP
DeepCCI	0.925	0.918	0.957	0.957
DeepDDI	0.891	0.886	0.916	0.915
MR-GNN	0.927	0.921	0.934	0.924
MLRDA	0.922	0.907	0.959	0.948
SEAL	0.894	0.886	0.941	0.937
GoGNN	0.937	0.932	0.963	0.962
PGON	0.939	0.936	0.968	0.965
GoGNN-M	0.914	0.909	0.938	0.931
GoGNN-I	0.921	0.898	0.929	0.912
GoGNN-noPool	0.931	0.930	0.958	0.954
GoGNN-noAttn	0.909	0.905	0.956	0.948

Table 4.2: Result of chemical-chemical interaction prediction task.

PGON improves the accuracy on the CCI prediction task compared to other baseline methods. The improvement proves that PGON can capture more abundant information on multiple scales by the attribute aggregation and selection in a graph of graphs perspective compared to the baseline models which only optimize the model with molecule graph inputs pair-wisely or individually. Further, the framework that follows the 2-FWL algorithm enhances the model's expressive power, and eventually improve the performance of PGON compared to GoGNN.

4.6.5 DDI Prediction Results

Settings. To familiarize the comparison, we divide the DDI dataset for training, testing, validation with ratio 6:2:2, and divide the SE dataset with ratio 8:1:1. The dimensions of molecule graph hidden feature, and the output molecule graph representation are set to 384, 256, respectively. We set the learning rate to 0.001, pooling ratio $\tau = 0.5$. We choose AUC and *average precision*(AP) for evaluation.

	DDI		SE	
	AUC	AP	AUC	AP
DeepCCI	0.862	0.856	0.819	0.806
DeepDDI	0.915	0.912	0.827	0.809
MR-GNN	0.932	0.922	0.769*	0.752*
MLRDA	0.931	0.926	0.847*	0.825*
Decagon	-	-	0.872	0.832
SEAL	0.925	0.921	N/A	N/A
GoGNN	0.943	0.933	0.930	0.927
PGON	0.946	0.937	N/A	N/A
GoGNN-M	0.905	0.902	0.862	0.817
GoGNN-I	0.922	0.917	0.860	0.834
GoGNN-noPool	0.900	0.891	0.912	0.909
GoGNN-noAttn	0.925	0.921	0.897	0.883

* indicates that the result is the output of the baselines after two weeks' training.
- DDI dataset has no protein data which is required by Decagon

Table 4.3: Result of drug-drug interaction prediction task.

Results. The experiment results for DDI prediction are listed in Table 4.3. The results show that compared with the baseline methods, GoGNN improves the performance with a significantly large margin. GoGNN improves the AUC and AP by 1.18% and 1.19% respectively on DDI dataset, and 6.65% and 11.42% respectively on the SE dataset. The improvement is attributed to the abundant information brought by the graph of graphs architecture and edge-filtered aggregation.

4.6.6 Ablation Results of GoGNN

The ablation experiment results on both tasks are shown in Table 4.2 and Table 4.3. The results prove that the graph of graphs architecture, attention-based pooling, attention-based and edge-filtered aggregation are all effective for the

side effect prediction task. Among all the variants, GoGNN-M and GoGNN-I have the most significant performance gaps between GoGNN, which indicates that the view of graph of graphs contributes the most to helping the model to capture more structural information that improves the prediction accuracy.

4.6.7 Ablation Results of PGON

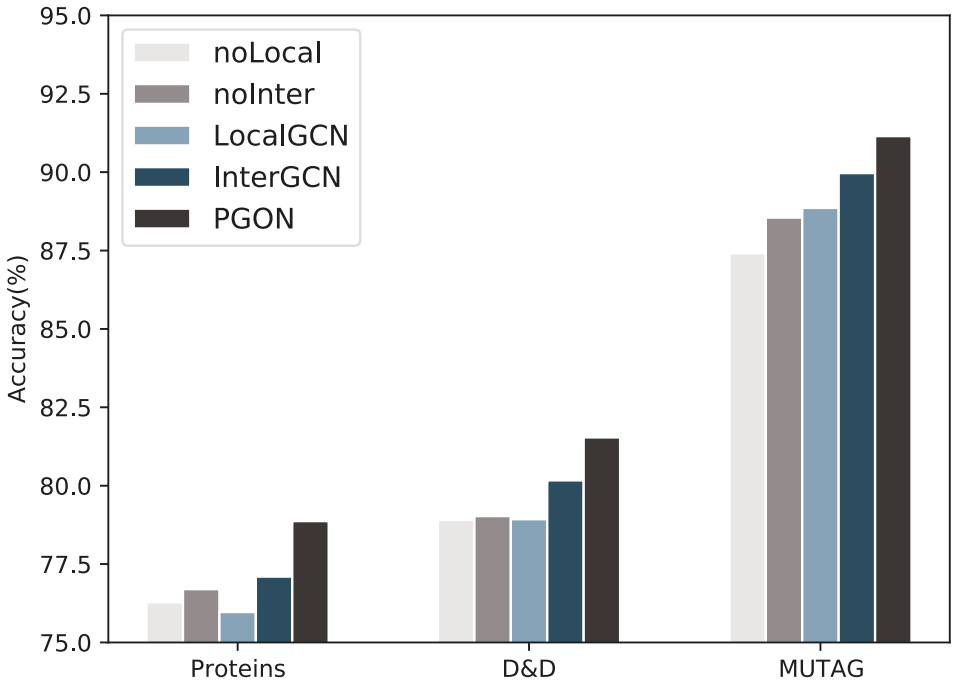


Figure 4.3: Ablation experiment result for graph classification.

In order to determine the effectiveness of graph of graphs framework and the expression power of the model. We conduct the ablation experiments which replace the component of our proposed model with other conventional components. The following variants of the PGON are tested for the ablation experiment.

noLocal is the variant that only applies the attention-based graph isomorphism

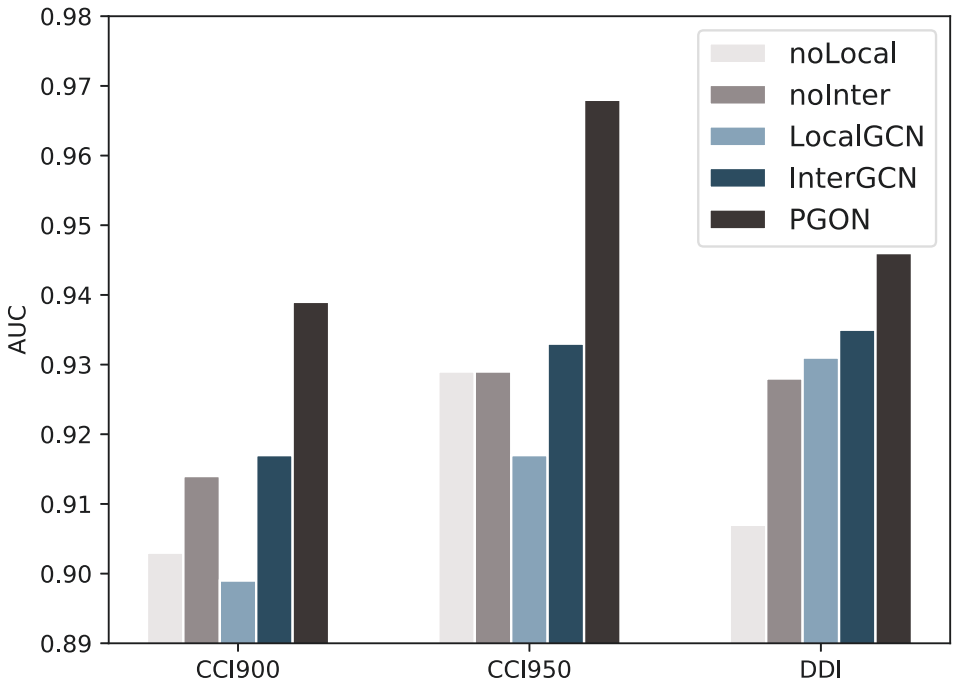


Figure 4.4: Ablation experiment result for graph interaction prediction.

network on the graph interaction network while skipping the representation learning for the local graphs.

noInter is the variant that only learns the representations for local graphs while ignoring the interaction graph between the local graphs.

LocalGCN denotes the variant that replace the local graph neural network with conventional graph convolutional neural network [47].

InterGCN denotes the variant that uses GCN for the message passing between the graphs on the interaction network.

The results are shown in Fig. 4.3 and 4.4, which prove the effectiveness of the graph of graphs architecture and the expressive power of the entire model. Among all the variants, LocalGCN and noLocal have the most significant per-

formance gaps between PGON, which indicates that the expressive power of PGON is the key factor in improving the performance for both classification and prediction tasks.

4.6.8 Parameter Sensitivity Analysis

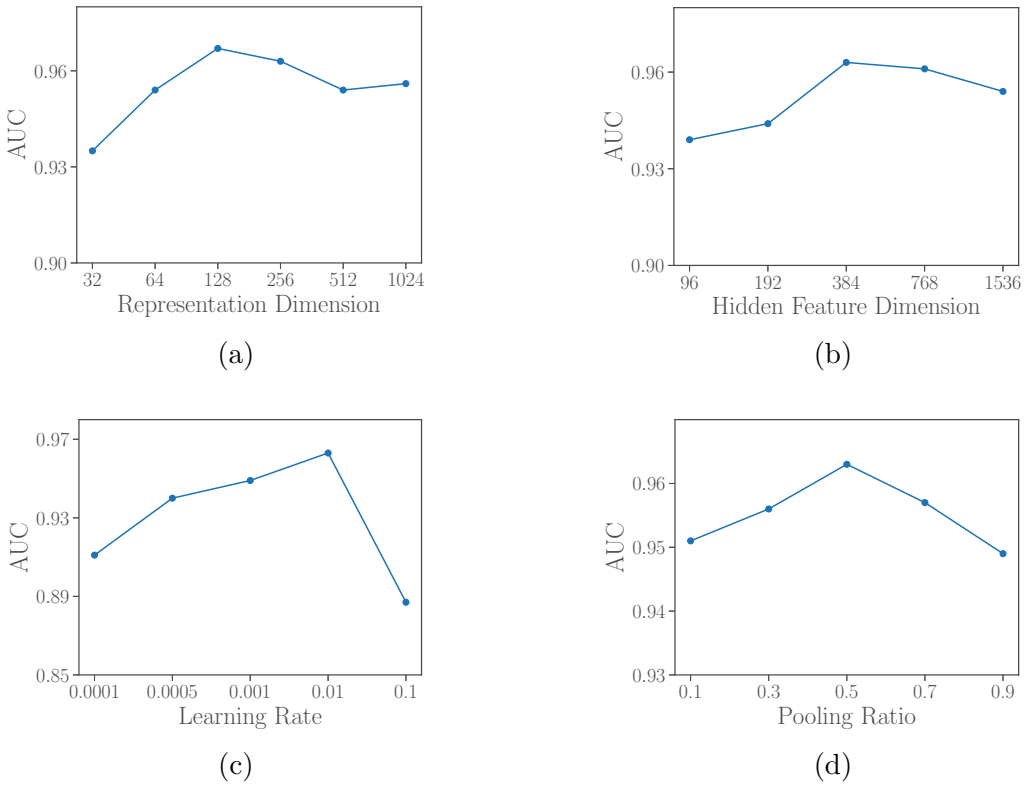


Figure 4.5: Parameter sensitivity experiment results

GoGNN. In this section, we report the impact of the hyper-parameters of GoGNN.

Settings. We conduct the parameter sensitivity experiment on the CCI950 dataset by changing the tested hyper-parameter while keeping other settings the same as mentioned in Section 4.6.4. We test the following hyper-parameters: the dimensions of the output representation and hidden feature, learning rate and

pooling ratio.

Results. As shown in Figure 4.5, overall, the impact of hyperparameter variation is insignificant. Figure 4.5a shows that GoGNN reaches the best performance with representation dimension 128. Figure 4.5b indicates that the salient point for the hidden feature size is 384. As for the learning rate and pooling ratio, the best point appears at 10×10^{-3} and 0.5, respectively.

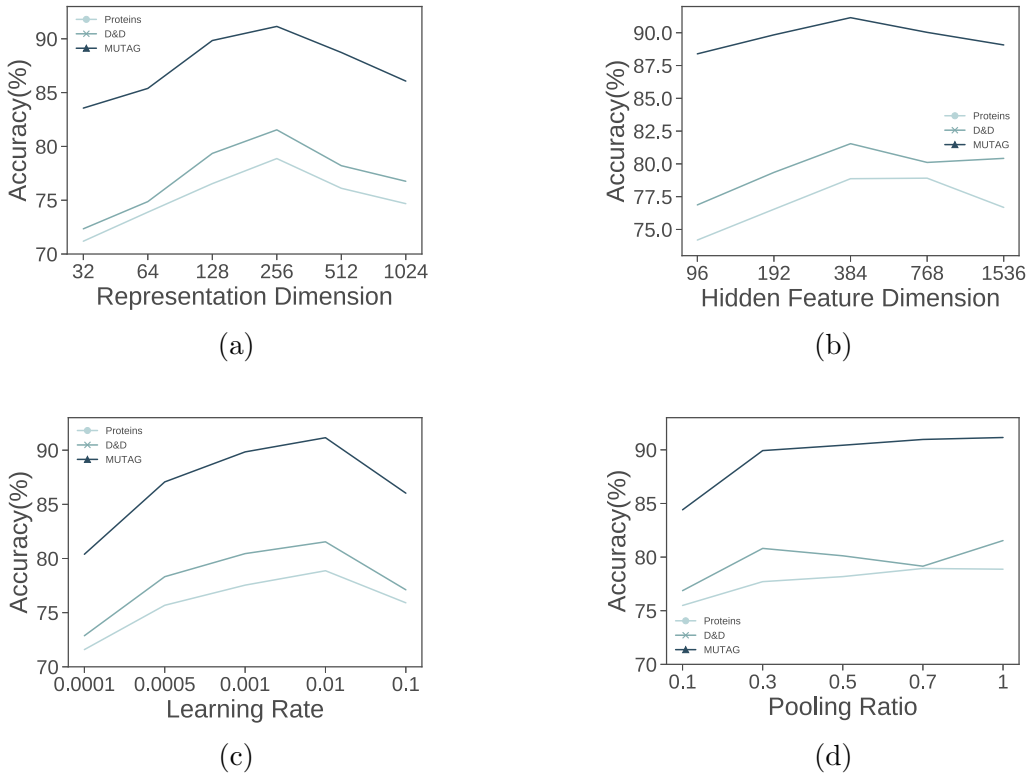
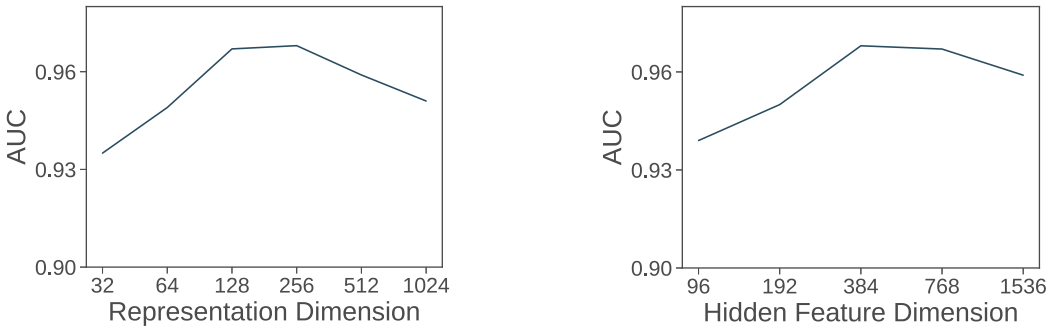


Figure 4.6: Parameter sensitivity experiment results for graph classification

PGON. We also report the sensitivity analysis results of PGON on both graph classification and structured entity interaction prediction tasks.

Parameter sensitivity is analyzed in the experiment. Hidden feature dimensions, final representation dimensions, pooling ratio and learning rate are tested for both graph classification and link prediction tasks to demonstrate how the parameters influence the performance of the model. For the graph classification

task, we test different parameter settings of parameters on all three datasets, while for graph interaction prediction task, the parameters are tested on the CCI950 dataset.



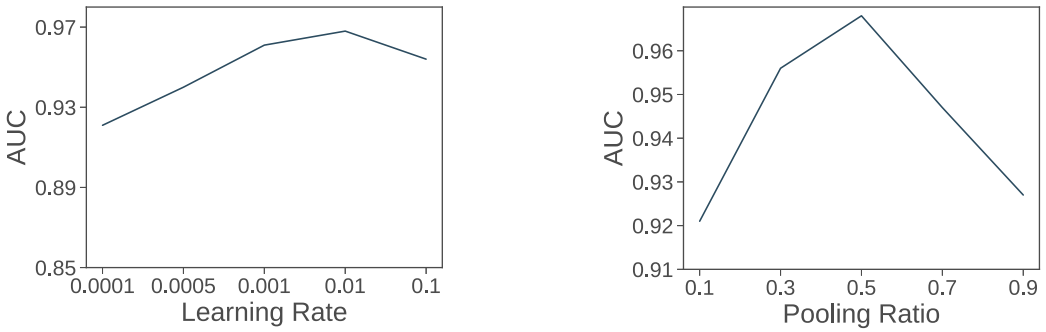


Figure 4.7: Parameter sensitivity experiment results for graph interaction prediction

The results are summarized in Fig. 4.6 and 4.7. Overall, the impact of the hyper-parameter variations is insignificant. The results indicate the salient point for each hyper-parameter. Therefore, the best settings for the hyper-parameters are selected as mentioned in Section 4.6.3, 4.6.4 and 4.6.5.

4.7 Conclusion

In this chapter, we focus on graph classification and structured entity interaction prediction problems. These two problems demand the model to capture the information of the structure of entities and the interactions between entities. However, the previous works represent the entities with insufficient information. To address this limitation, we propose novel models GoGNN which leverages the dual-attention mechanism in the view of graph of graphs to capture the information from both entity graphs and entity interaction graph hierarchically, and PGON that can be applied to both graph classification and graph interaction prediction tasks. PGON is able to preserve the structure and feature information of the graph of graphs, which empirically enhances the performance on link prediction and graph classification. PGON also has the provably great expressive power which helps the model to distinguish the structure of different graphs. The experiments on real-life datasets demonstrate that our model could improve the performance on the graph classification, chemical-chemical interaction prediction and drug-drug interaction prediction tasks.

Chapter 5

Reinforcement Learning based Query Vertex Ordering

5.1 Chapter Overview

In this chapter, we present our novel reinforcement learning based query vertex ordering model for subgraph matching. This work is current in submission. The rest of the chapter is organized as follows: Section 5.2 includes the preliminaries, definitions and related works. Section 5.3 presents the details of our proposed model RL-QVO. We report the experiment results in Section 5.4 and give a conclusion in Section 5.5.

5.2 Background

In this Section, we first introduce notations and the existing general framework for all backtracking-based subgraph matching algorithms in Section 5.2.1. Section 5.2.2 provides the problem statement. Then Section 5.2.3 introduces the details of the state-of-the-art subgraph matching algorithm.

5.2.1 Preliminaries

In this chapter, we focus on the subgraph matching problem on the undirected labeled graph $G = (V, E)$, where V denotes the vertices set, E is a set of edges. There is a label function f_l which maps a vertex $v \in V$ into a label l in the label set L. The query graph, denoted by q, is a potential subgraph of data graph G. q and G share the same label function f_l and the same label set L. The frequently used notations are summarized in the Table 5.1.

Table 5.1: The summary of notations

Notation	Definition
g, q, G	graph, query graph and data graph.
V, E, L	vertex set, edge set and label set.
f_l, f_{iso}	label mapping function and subgraph isomorphism function.
$e(u, v)$, $N(v)$	an edge between node u, v and the neighbor set of v.
$d(u)$	degree of node u.
C, LC	set of candidate vertices and local candidate vertices.
ϕ, $\#_{enum}$	matching order and enumeration number.
$N_+^\phi(v), N_-^\phi(v)$	backward and forward neighbors of v given ϕ.

We give the key formal definition of the subgraph isomorphism as follows:

Definition 5.2.1 (Subgraph Isomorphism)**.** *Given a query graph $q = (V, E)$ and a data graph $G = (V', E')$, a subgraph isomorphism is an injective function f_{iso} from V to V' such that (1) $\forall v \in V, f_l(v) = f_l(f_{iso}(v))$; and (2) $\forall e_{(u,v)} \in E, e_{(f_{iso}(u), f_{iso}(v))} \in E'$.*

Subgraph Matching. The objective of the subgraph matching is searching for all subgraph isomorphisms from query graph q to data graph G. Generally, the existing subgraph matching algorithms can be classified into two categories: backtracking based methods and join based methods. Both directions have been intensively studied in the literature. In this chapter, we focus on the backtracking based subgraph matching methods.

Algorithm 2: Generic Subgraph Matching

Input: Query graph q, data graph G.
Output: Subgraph Match Mapping M from q to G.
`// The filtering method.`
1 $C \leftarrow$ generate candidate vertex sets
`// The ordering method.`
2 $\phi \leftarrow$ generate a matching order based on q, G and C
`// The enumeration procedure, finds the mapping` M `for all query nodes in` q.
3 $M =$ `Enumerate`$(q, G, C, \phi, \{\}, 1)$

As summarized in previous works [95, 54], all backtracking based subgraph matching method can be partitioned into three phases: (1) the *complete candidate vertex set* generation; (2) matching *order* generation; and (3) *matching enumeration*. This general framework (from [95]) is outlined in Algorithm 2.

In more details, **Phase (1)** is to pre-process the graph where various filtering techniques are deployed to generate candidate vertex sets C for all query nodes; this can reduce the search space before the enumeration process begins.

Definition 5.2.2 (Complete Candidate Vertex Set $\mathcal{C}$)**.** *Given q and G, a complete candidate vertex set $C(v)$ of $v \in V(q)$ is a set of data vertices such that for each $u \in V(G)$, if (v, u) exists in a match from q to G, then $u \in C(v)$.*

In **Phase** (2), a match order ϕ is generated to guide the enumeration of the matched subgraphs, which is formally defined as follows.

Definition 5.2.3 (Matching Order)**.** *A matching order ϕ is a permutation (i.e., sequence) of query graph's vertex set $V(q)$.*

Usually, the matching order is generated heuristically based on the label frequency, degree of vertices and other structural and attribute information. For example, QSI introduces an *infrequent-edge first ordering method*, VF2++ uses a *infrequent-label first order* and CFL proposes a *path-based ordering method*.

Following is the formal definition of enumeration procedure in **Phase (3)**, which finds all matches of the query subgraph q in the data graph G.

Definition 5.2.4 (Backward (Forward) Neighbors)**.** *With given order ϕ, the backward (forward) neighbors $N_+^\phi(v)$ ($N_-^\phi(v)$) are the neighbors of v positioned before (after) v in ϕ.*

Definition 5.2.5 (Enumeration Procedure)**.** *An enumeration procedure is performed recursively to find all subgraph matchings f_{iso} with given matching order ϕ and candidate vertex set C.*

5.2.2 Problem Statement

As highlighted in [95], it is critical to choose a good matching order since the enumeration number in the search is significantly influenced by the matching order, which is strongly correlated to the query time. Following is a formal definition of the enumeration number.

Definition 5.2.6 (Enumeration Number)**.** *An enumeration number $\#_{enum}$ is the number of recursive calls of the enumeration procedure to find all subgraph matchings with given q, G, ϕ and C.*

It is cost-prohibitive to find an *optimal* matching order, and existing algorithms resort to heuristic strategies to find a good matching order. Inspired by the great successes of RL techniques in a variety of optimization problems [3], in this chapter, we **aim to apply RL technique to find a matching order** for given query graph q and data graph G to **minimize the enumeration number** in the matching enumeration phase such that the search performance of the subgraph matching can be enhanced.

5.2.3 State-Of-The-Art

In this subsection, we introduce the backtracking search based subgraph matching algorithm Hybrid which is shown to achieve state-of-the-art performance according to the intensive empirical results [95]. Hybrid utilizes the candidate filtering, vertex ordering and enumeration methods of GraphQL [39], RI [8] and QuickSI [91] respectively, which are introduced as follows:

(1) Candidate Set Generation

To limit the size of candidate vertex set for each node in the query, Hybrid utilizes the candidate vertex filtering method in GraphQL [39]. Particularly, Hybrid generates the candidate set with local pruning and global refinement. Local pruning filters out the invalid nodes based on the *profile* of the neighborhood graph of the query vertex v, which is the lexicographic order labels of v and neighbors of v. If the profile of query vertex v is a sub-sequence of that of data vertex u, then u is added to $C(v)$. Afterward, the candidate set $C(v)$ is generated for all vertices $v \in V(q)$. Global refinement prunes $C(v)$ as follows: given $u \in C(v)$, the algorithm first build a bipartite graph B_u^v between $N(u)$ and $N(v)$ by adding $e(u', v')$ where $u' \in N(u)$ and $v' \in N(v)$, if $u' \in C(v')$. Global refinement then checks whether there is a semi-perfect matching in B_u^v, *i.e.,* whether all vertices in $N(v)$ are matched. If not, v will be removed from $C(u)$.

Example 1. *Consider the query graph and data graph in Figure 1.2. Hybrid builds the candidate set with the following steps. The profile of the query vertices* v_2 *is* $\{ A, B, D \}$. *The profiles of* u_2 *and* u_3 *are the same* $\{ A, B, D \}$ *and that of* u_4 *is* $\{ A, B, D, D, D, D \}$. *They all subsume the profile of* v_2, *hence we have* $C(v_2) = \{ u_2, u_3, u_4 \}$. *Similarly, local pruning can build the following candidate sets:* $C(v_1) = \{ u_1 \}$, $C(v_3) = \{ u_5, u_6, u_7 \}$ *and* $C(v_4) = \{ u_{10}, u_{11} \}$. *Global*

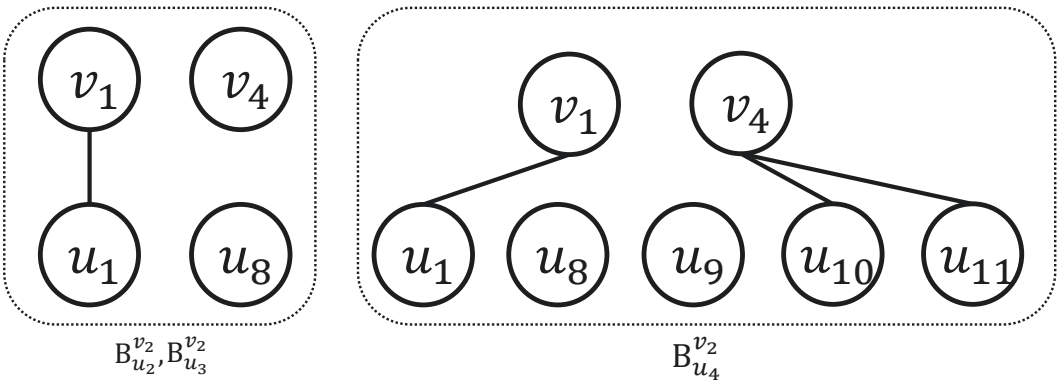


Figure 5.1: Illustrating the construction of bipartite graphs for candidate filtering

refinement is then applied to build a bipartite graphs B_{u2}^{v2}, B_{u3}^{v2} and B_{u4}^{v2} for neighbors of nodes in $C(v_2)$ and v_2 (See Figure 5.1). Since $u_8 \notin C(v_4)$, we cannot find a semi-perfect matching in B_{u2}^{v2} and B_{u3}^{v2}. Hence u_2 and u_3 should be removed from $C(v_2)$. Finally, we can get the candidate set for v_2 after global refinement as $C(v_2) = \{ u_4 \}$.

(2) Matching Order Generation

Query vertex order generation is one of the key factors that reduce the query time. Hybrid exploits RI's order generation method which is the state-of-the-art backtracking-based ordering method [95]. Hybrid generates the matching order only based on the structure of query graph q. Hybrid first selects the query node with the maximum degree $v* = \arg\max_{v \in V(q)} d(v)$ as the starting nodes of order ϕ. We denote the order determined at the t-th step ϕ_t. Then the nodes with the most neighbors in ϕ_t are iteratively selected, *i.e.,* $v* = \arg\max_{v \in V(q) \wedge v \notin \phi_t} |N(v) \cap \phi_t|$. Since ties can easily occur in the arg max function, Hybrid breaks the ties by considering following properties in order of: (1) the maximum value of $|\{v' \in \phi_t \mid \exists v'' \in V(q) \setminus \phi_t, e(v', v'') \in E(q) \wedge e(v, v'') \in E(q)\}|$, *i.e.,* the number of vertices in ϕ_t that have a neighbor outside of ϕ_t and is adjacent with v. The number is denoted as $|v_{neig}|$. (2) the maximum number of neighbors of v that are not in ϕ, and not even adjacent with vertices in ϕ_t, which

is defined as $|v_{unv}| = |\{v' \in N(v) - \phi_t | \forall v'' \in \phi, e(v', v'') \notin E(q)\}|$. If these values are still the same for at least two nodes, Hybrid will choose a node arbitrarily.

Example 2. *We still take the example graphs in Figure 1.2 to illustrate the ordering process. The query vertex with the largest degree should be selected as the first node. However, the degree of all four query nodes are* 2*. According to the tie-break rules,* $|v_{neig}|$ *and* $|v_{unv}|$ *are still same for all vertices because there is no node in* ϕ*, hence the starting vertex is chosen arbitrarily. In this example,* v_4 *is chosen as our starting node. In the next step, we can get* $|N(v_2) \cap \phi_t| = |N(v_3) \cap \phi_t| = 1$ *and we still cannot break the tie according to the rules, therefore, we randomly add* v_2 *to* ϕ*. Similarly, Hybrid still cannot distinguish the priority of* v_1 *and* v_3 *in the next step and should apply the random selection. As a result, the order generated by Hybrid is* $\phi = \{v_4, v_2, v_1, v_3\}$*.*

From above Example 2, we can easily conclude the drawback of Hybrid's ordering method. Since Hybrid only considers the structure of query graph to build the matching order, it ignores the abundant information of the relationship between the query and data graphs. Consequently, Hybrid has to apply random selections in many cases, even selects the starting vertex arbitrarily. This selection cannot guarantee to produce a robust matching order to reduce redundant intermediate results.

(3) Enumeration Procedure.

Hybrid adopts the recursive enumeration procedure which is also widely used in existing backtracking search based methods. In the enumeration process, the algorithm generates the local candidate set for query vertices, and evaluates the matches of query node and corresponding local candidate vertices. The enumeration framework is summarized in Algorithm 3. The function `Enumerate(·)` recursively enumerates all results. M is a set that records all mappings from query vertices to the data graph vertices. Once all the vertices of the query

Algorithm 3: Enumeration Procedure from [95]

Input: Candidate set C, query graph q, data graph G, match order ϕ.
Output: Subgraph Match Mapping M.

```
1  Enumerate(q, G, C, φ, M, i)
   // Start with Enumerate(q, G, C, φ, {}, 1)
2  begin
3      if i > |φ| then
4          Output M, return.
5      v ← select an extendable vertex given φ and M
       // Compute local candidate set
6      LC(v, M) ← ComputeLC(q, G, C, φ, M, v, i)
7      for u ∈ LC(v, M) do
8          if u ∉ M then
9              Add (u, v) to M
10             Enumerate(q, G, C, φ, M, i + 1)
11             Remove (u, v) from M
```

graph are mapped, Line 4 outputs the mapping M. Otherwise, the algorithm selects an extendable vertex which is defined as follows:

Definition 5.2.7 (Extendable Vertices)**.** *Given matching order ϕ and mapping M, extendable vertices $\Gamma(\phi, M)$ are query vertices v such that each $v' \in N_+^\phi(v)$ has been mapped in M but v has not.*

Line 6 shows the computation of the local candidate set after selecting the extendable vertex. The local candidate set is computed following different rules for specific subgraph matching algorithms. Hybrid generates the local candidate set $LC(\cdot)$ by checking whether the vertices in candidate set are connected to the last matched data vertices. Line 7-11 loop over $LC(v, M)$ to find more mappings to extend M, and recursively invoke the `Enumerate(·)` function. Finally, we get the matching from $q(\phi(1:i))$ to G.

Example 3. *In previous Example 1 and Example 2, Hybrid generates the candidate set $C(v_1) = \{ u_1 \}$, $C(v_2) = \{ u_4 \}$, $C(v_3) = \{ u_5, u_6 \}$, $C(v_4) = \{ u_{10}, u_{11} \}$*

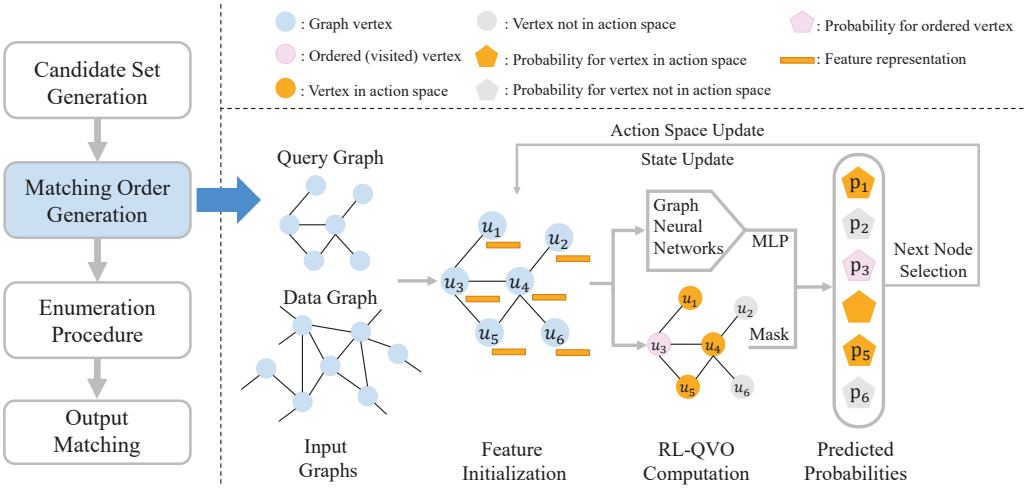


Figure 5.2: Framework of RL-QVO

and the query vertex order $\phi = \{v_4, v_2, v_1, v_3\}$ *according to the corresponding methods. By applying the enumeration procedure in Algorithm 3, the enumeration number should be* 10 *to find all possible matches with the redundant intermediate visit* $\{u_{10}, u_4, u_1, u_6\}$*. If the generated order is* $\{v_1, v_3, v_4, v_2\}$*, the redundant intermediate visit can be skipped and the enumeration number reduces to* 9*.*

Please note that the sizes of query and data graphs in this example are relatively small, the gap between the enumeration numbers may not be very large. In the real applications where the data graph has millions or even billions of vertices, the gap between enumeration numbers can be non-negligibly significant.

In this chapter, We design the RL-based matching order generation method to improve the overall performance. Specifically, we utilize the same candidate set generation and enumeration methods as Hybrid.

5.3 Our Approach

In this section, we present a reinforcement learning based model, namely RL-QVO, to generate high-quality matching orders. Particularly, Section 5.3.1 introduces the motivation of the model. Section 5.3.2 presents the framework of the model. Section 5.3.3 shows how to model the problem as Markov Decision Process. Section 5.3.4 and Section 5.3.5 design the network structures and policy strategies of our model. Section 5.3.6 provides the time and space complexity analysis.

5.3.1 Motivation

The backtracking-based subgraph matching methods explore the search space with a given matching order, which presents several unique challenges that are good fits for reinforcement learning and graph neural network:

First, it is cost-prohibitive to find the optimal matching order for a given query graph q and a data graph G. The existing ordering methods adopt the greedy heuristics to pick nodes that could possibly reduce the search space at current step as early as possible. The greedy heuristics are usually efficient and general so that can be applied on any graph. However, these selections may result in the global *suboptimality* since they follow a predefined rule without adjustment according to graph's distribution. With the help of reinforcement learning technique, our model is able to generate the matching order by considering the long-term rewards multi-steps ahead. Therefore, our proposed RL-QVO has better chance to escape local optimum, and eventually shows higher matching order quality than the greedy methods.

Second, existing ordering methods only utilize the local neighborhood information with fixed priority to produce the matching order, which has a non-

negligible probability to ignore some structural and attribute information within q and G. For example, some ordering methods first select the node with smaller degree, which only considers the first order information. These order methods ignore the high-order information and the label information in the ordering process, which may consequently increase the overall query processing time. Graph neural networks (GNNs) have already been proven to have powerful ability of exacting information for specific graph analysis tasks. With the reinforcement learning framework, RL-QVO does not need to make assumptions on the query or data graph distributions, but regards the ordering process as a black-box computation with features learned by GNNs. As a result, RL-QVO generates the matching order adaptively with regards to different query graphs.

These motivate us to apply the Graph Neural Networks and Reinforcement Learning techniques to better utilize the graph information and find high-quality matching orders by looking multi-steps ahead (i.e., long-term rewards) during the training process. Though the training of the model needs extra overhead compared to existing matching order generation approaches, it can be preprocessed which is a common practice for various indexing techniques in the literature. Moreover, we show that the generation of the matching order during the query processing is very time efficient, and the gain is significantly as demonstrated in our experiments.

5.3.2 Framework

The research focus of this chapter is to design a novel model to generate good matching orders for backtracking based subgraph matching algorithms. Thus, we replace the matching order generation part of the the state-of-the-art algorithm Hybrid with a new model: **Reinforcement Learning Based Query Vertex Ordering Model** (**RL-QVO** for short).

Figure 5.2 illustrates the framework of the RL-QVO. Particularly, RL-QVO regards the matching order generation as a Markov Decision Process (MDP), and the vertices in a matching order are generated sequentially. At the beginning, RL-QVO first generates neighbor information with carefully designed features to produce node representations for each query vertex. With the obtained representations, RL-QVO computes the probability scores for query vertices to guide the selection of the next query node for the matching order. The matching order is output if the last query node is processed. Otherwise, we will repeat this procedure with updated action space and features. More details of the model will be introduced in the following subsections.

5.3.3 Query Vertex Ordering as Markov Decision Process

The ordering process can be naturally formulated as a Markov decision process (MDP). Given ordered nodes ϕ_t and input feature matrix $\boldsymbol{H}^t$ of query graph at step t as a state, reinforcement learning method takes an action from action space determined by the neighbors of the ordered vertices $N(\phi_t) = \{N(v)|\forall v \in \phi_t, N(v) \notin \phi_t\}$ with predicted probabilities to select next node for the order ϕ_{t+1}. After every selection, the feature matrix $\boldsymbol{H}$ is also updated. With the generated order ϕ, the model performs the enumeration procedure and gets the reward based on the reduced number of enumerations $\Delta\#_{enum}$ compared to baselines, entropy of predicted probabilities and rewards for valid predictions. The MDP is formally defined as follows:

State. At step t, the state is defined by the order ϕ_t which contains t vertices and query graph representation matrix $\boldsymbol{H}_q^t$. The whole query graph feature matrix $\boldsymbol{H}_q$ is involve in the state representation in order to familiarize the model with the overall structural and attributed information of the query graph. Since RL-QVO is a machine learning-based model, for each vertex in query graph, we

carefully initialize a feature representation $\boldsymbol{h}_v^{(0)}$.

Feature Representations. As discussed in Section 5.2.1, the statistical heuristics of query graphs, such as degree of query vertices, label frequency, candidate set size and *etc*, play a key role in determining the query vertex order. Inspired by [40], we initialize the representations for all nodes based on all heuristics in order to fully exploit the information and relations hidden in and between query and data graphs. Specifically, we define the initial state as follows:

- We compute the degree of each node. Intuitively, the query node with greater degree will have less matching in the data graph, and eventually has higher priority to be added in the order ϕ. Therefore, we use a scaled degree as one of the initial state value, *i.e.,*

$$\boldsymbol{h}_v^{(0)}(1) = degree(v)/\alpha_{degree}$$

, where α_{degree} is a scaling factor to ensure the computation stability.

- The initial feature also includes the node label information. we simply put the digit-encoded vertex label into the representation.

$$\boldsymbol{h}_v^{(0)}(2) = label(v)$$

- To enable the graph neural network used in RL-QVO to discriminate the order of input node, we directly put the query node id in the initial representation.

$$\boldsymbol{h}_v^{(0)}(3) = id(v)$$

Please note that the query graph usually has small number of nodes, therefore there is no need to perform scaling on the vertex id.

- We further include the data graph related heuristics to build our initial feature representation. The following statistics are commonly used in existing models: the number of nodes in the data graph with less degree than vertex v; the number of nodes in the data graph that have the same label with v and the candidate set size of v. They are all vectorized in the initial representation.

$$\boldsymbol{h}_v^{(0)}(4) = |\{u \in G | d(u) < d(v)\}| / \alpha_d;$$

$$\boldsymbol{h}_v^{(0)}(5) = |\{u \in G | L(u) = L(v)\}| / \alpha_l;$$

$$\boldsymbol{h}_v^{(0)}(6) = |C(v)| / \alpha_c,$$

  where α_d, α_l and α_c are the scaling factors.

- Lastly, we put a trailing indicator at the initial representation. The indicator variable is 0 if the vertex v has not been visited and 1 otherwise, *i.e.,*

$$\boldsymbol{h}_v^t(7) = \mathbb{1}(v \in \mathcal{V}_{visited}^{t-1})$$

  Please note that the indicator variable will change with the update of the state at time step t.

We concatenate all these features to formulate the input representation. This representation could easily incorporate additional heuristic features by appending these features to our initial representation vector. In our model, we only use six primary heuristics introduced above with a trailing indicator and exploit the reinforcement learning and graph neural networks to automatically learn more complicated and informative criterion for query vertex order generation.

Action. The action space is defined by the neighbor vertices set $N(\phi_t) = \{N(v) | \forall v \in \phi_t, N(v) \notin \phi_t\}$ of the ordered vertices at current step t. At every

step t, the action is to select vertex v' from $N(\phi_t)$ according to the predicted probabilities and add v' into the matching order for next step, *i.e.,* $\phi_{t+1} = \phi_t \cup \{v'\}$. Instead of directly selecting the vertex with greatest probability, RL-QVO makes the selection according to the probabilities of vertices in the action space to allow more exploration.

Reward Design. The reward is key factor in reinforcement learning, in this chapter, the reward includes the reduced number of enumeration, validate reward for probability scores and entropy of the probabilities. One immediate reward is the reduced enumeration number $\Delta\#_{enum} = \#_{enum}(\phi) - \#_{enum}(\phi_{base})$, where ϕ is the learned order of RL-based agent and ϕ_{base} is the baseline order produced by existing subgraph matching algorithms. Specifically, according to [95], the ordering method of RI [8] has the best performance. Therefore, we choose the order produced by RI as our baseline order, *i.e.,* $\phi_{base} = \phi_{RI}$, and the baseline algorithm has exactly the same filtering and enumeration methods as our proposed RL-QVO. Considering the varying orders of magnitude of $\Delta\#_{enum}$ with different query graphs, the enumeration reward r_{enum} is defined as $r_{enum} = f_{enum}(\Delta\#_{enum})$, where $f_{enum}(\cdot)$ is a function such as logarithm which reduces the gaps (differences) between enumeration rewards under different query graphs to stabilize the computation. Intuitively, the model is more likely to gain greater enumeration reward r_{enum} on the complex queries which inherently require more rounds of enumeration procedure. Actually, the average enumeration time is usually dominated by the time costs of these complex queries. Therefore, the policy network pays greater importance to the complex queries with the hope to reduce more enumeration numbers. Since the enumeration reward $r_{enum,t}$ at step t cannot be determined until the final order ϕ is obtained, all rewards $r_{enum,t}$ at steps t share the same value as $r_{enum} = f_{enum}(\Delta\#_{enum})$. Meanwhile, this shared reward value enables the policy network to consider the long-term

reward at every step, thus reduces the probability to fall into the local optimum.

We also design the step-wise validate rewards $r_{val,t}$. A small positive reward is assigned if the policy network produces a valid probability distribution, i.e., the vertex with largest probability is in the action space $v' \in N(\phi_t)$. Otherwise, a negative punishment is assigned which is greater than the positive reward in absolute value. Please note that even if the policy network produces invalid probabilities, our model still guarantees to generate a *connected order* ϕ by masking out the vertices that are not in the action space before making selection.

Furthermore, an entropy reward is considered at every step to encourage the model to output action probability distribution with *high* entropy [120]. Hence, the model is more likely to take actions unpredictably, which avoids the agent converging too quickly on a policy that is locally optimal. This entropy reward $r_{h,t}$ is defined as $r_{h,t} = H(P_{\pi_\theta}(\phi_t, N(\phi_t))$, where $H(\cdot)$ is the entropy function, π_θ is a policy network with parameters θ, and $P_{\pi_\theta}(\phi_t, N(\phi_t))$ is the output probability at step t with given order ϕ_t and action space $N(\phi_t)$. Therefore, overall step-wise reward is formulated in the following Equation 5.1:

$$R_t = r_{enum} + \beta_{val} \cdot r_{val,t} + \beta_h \cdot r_{h,t}, \tag{5.1}$$

where β_{val} and β_h are the reward coefficients for validate reward and entropy reward respectively to tune their impact on training process.

In query ordering for subgraph matching task, the starting nodes in the order are far more important than the trailing nodes. Therefore, when calculating the overall rewards for the policy network, we assign a decay factor to the step-wise rewards and formulate the overall rewards as follows:

$$R_{q,\theta} = \sum_{t=1}^{|V(q)|} \gamma^t R_t, \tag{5.2}$$

where $\gamma \in [0, 1]$ is a decay factor which enables the model to consider the importance of nodes in the learned order during training procedure.

5.3.4 RL-QVO Policy Network Architecture

In this section, we introduce the architecture of policy network in our RL-based method RL-QVO, which generates the query vertex order for subgraph matching.

Framework. Framework of RL-QVO is illustrated in Figure 5.2. RL-QVO first computes the vector representations $\boldsymbol{x}_v$ for query vertices $v \in V(q)$ by exploiting graph neural networks. The representation matrix of query vertices $\boldsymbol{X}_q$ is served as the input of a multi-layer perceptron (MLP) to obtain the probability distribution on action space.

Action Space. As introduced before, the action space $AS(t)$ is defined by the neighbor vertices set $N(\phi_t) = \{N(v)|\forall v \in \phi_t, N(v) \notin \phi_t\}$ of the ordered vertices at current step t to ensure the connectivity of generated orders. This constraint applies for most ordering methods in backtracking-based subgraph matching algorithms. In the case that there is only one vertex in the action space, *i.e.,* $|AS(t)| = 1$, RL-QVO directly selects the only candidate as the next node without performing computation.

Policy Network. The policy network is a neural network which learns the rule for solving the target problem. With given state, a policy network returns a probability distribution over the action space. In our case, according to the produced probability distribution, RL-QVO progressively selects the vertex to add into the matching order ϕ. In our model, the policy network includes two main parts: the graph neural network that aggregates and extracts the graph information; and the multi-layer perceptron which finally produces the probability distribution. We introduce the details as follows.

In order to obtain the vector representations for query vertices, we utilize

graph neural network, a well-studied technique which achieves the state-of-the-art performance in graph representation learning. Specifically, the policy network π is parameterized as graph convolutional network (GCN) [47] to embed the query vertices.

The high level idea of graph convolutional neural network is to perform a message passing along edges in the graph for total of $\mathcal{L}$ layers. Therefore, RL-QVO could not only obtain the heuristic information from the initial feature matrices, but also make prediction based on auxiliary structural and high-order neighboring information which are overlooked in conventional subgraph matching algorithms. The aggregation of graph convolutional neural network can be formulated as follows:

$$\boldsymbol{H}^{(l+1)} = \sigma(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}), \tag{5.3}$$

where $\boldsymbol{W}^{(l)}$ and $\boldsymbol{H}^{(l)}$ are the weight and the input feature matrices of l^{th} layer respectively, $\boldsymbol{H}^{(0)}$ is the initial feature representation introduced in Section 5.3.3. $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ is the adjacency matrix with self loops, $\tilde{\boldsymbol{D}}$ is a diagonal matrix where $\tilde{\boldsymbol{D}}_{ii} = \sum_i \tilde{\boldsymbol{A}}_{ij}$, and σ is an activation function such as ReLU. RL-QVO is compatible to any graph neural network. Because GCN could capture the structural, label and neighboring information with simple network structure and efficient computation process, we choose graph convolutional network as the graph representation learning module of RL-QVO. $\mathcal{L}$ layers of GCNs output the representations for all query vertices $\boldsymbol{H}_q^{\mathcal{L}}$ RL-QVO then applies the multi-layer perceptron (MLP) on obtained representation $\boldsymbol{H}_q^{\mathcal{L}}$ to select the next node. Specifically, we use two linear neural layers with mask and normalization operations:

$$\mathbb{P}_{v'}^{(t)} = \pi(\cdot|S^{(t)}) = Softmax(mask_{v' \in AS(t)}(\boldsymbol{W}_2 \cdot \sigma(\boldsymbol{W}_1 \boldsymbol{h}_{v'}^{(t)}))), \tag{5.4}$$

Where $\boldsymbol{h}_{v'}^{(t)}$ is the vector embedding of query vertex v'. The output dimension of the MLP is 1, therefore, we get a real number score as the selection probability for each node. We utilize the mask operation to filter out the probability scores of vertices that are out of the action space, and then apply the softmax function on the candidate query vertices: $Softmax(\boldsymbol{z}) = \frac{e^{\boldsymbol{z}_i}}{\sum_j (e^{\boldsymbol{z}_j})}$ to produce the normalized probabilities. These normalized probabilities for candidate vertices is also used to compute the entropy rewards mentioned in Section 5.3.3. Please note that in Equation 5.4, we omit the bias for simplicity.

5.3.5 Policy Training

In the training phase, given N_T training query graphs $q = \{q_i | i = 1, ..., N_T\}$ and a data graph G, our goal is to maximize the expected rewards of policy network π_θ for the training graphs. The reward of RL-QVO's policy network π_θ with parameters θ at time step t is the summation of rewards for all query graphs in the training batch as follows:

$$r_t(\theta) = \sum_{i=1}^{N_T} R_i(\phi_i; \theta_i)), \tag{5.5}$$

where R_i is the reward for query graph q_i defined in Equation 5.2. We utilize the proximal policy optimization (PPO) [89] to train the policy network. PPO is a policy gradient method which utilizes a sampling policy network that enables the model to update with sampled data in multiple epochs. In our case, the policy network $\pi_{\theta'}$ from the previous epoch (has not been updated) is used as sampling policy network. The objective function is formulated as

$$J_r^{(t)}(\theta) = \sum_{(a_t, s_t)} min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)} r_t(\theta), clip(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)}, 1-\epsilon, 1+\epsilon) r_t(\theta)), \tag{5.6}$$

$$J(\theta) = \sum_{t=1}^{|V(q)|} J_r^{(t)}(\theta) \tag{5.7}$$

where θ' is the parameters of policy network in previous epoch, ϵ is a factor to clip the ratio of $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)}$ which is the probability ratio of action a_t with state s_t computed by current and sampling policy networks. The model replaces the sampling policy network with current policy network after each training epoch.

With the matching order obtained by the policy network, the algorithm executes the enumeration procedure to find all subgraph matches. We adopt the commonly used enumeration procedure summarized in Algorithm 3, which is also utilized in the state-of-the-art baseline method Hybrid.

5.3.6 Complexity Analysis

RL-QVO is built on the neural network framework that enjoys excellent time efficiency in terms of query process. With a given query graph q and data graph G, RL-QVO is only required to perform computation of graph neural networks and MLP on the query graph to produce the probability for node selection at every time step. In order to obtain the matching order for graph with $|V(q)|$ vertices, such computation should be executed for $|V(q)|$ times. Since the time complexity of GNN is $O(|E(q)|)$ [108] and that for MLP is $O(d^2)$ where d is the dimension of representations. As a result, the overall time complexity of RL-QVO for query vertex ordering is $O(|V(q)| \times (|E(q)| + d^2))$. Because the size of query graph q is relatively small, the time complexity of RL-QVO is negligible compared to the time cost of iterative enumeration. In terms of the space complexity, RL-QVO requires fixed space for the parameters of the neural network, which is determined by the input and output vector dimensions of a network. Specifically, the space complexity is $O(\mathcal{L} \times d^2)$, where $\mathcal{L}$ is the number of

neural layers and d is the dimension of representations. Therefore, our proposed RL-QVO remains fixed space requirement with growing sizes of query and data graphs.

5.4 Experiment

This section shows the results of empirical studies.

5.4.1 Experiment Setup

Compared Methods. In order to show the efficiency and effectiveness of our proposed RL-QVO, we conduct the experiments on the following compared methods whose techniques are recommended in [95].

- **QuickSI (QSI)** [91] is a directly enumeration method which does not generate candidate set, but filters out the unpromising vertices during enumeration. QSI uses the *infrequent-edge first ordering method*.
- **RI** [8] is a state space representation based model which generate the query vertex order only based on the structure of query graph q.
- **VF2++** [43] is also a state space representation based model which determines the vertex order by the node label frequency.
- **Hybrid** [95] has a combination of candidate filtering, vertex ordering and enumeration methods of GQL, RI and QSI respectively. This method is proven to have the best performance according to an extensive empirical study [95].
- **RL-QVO** is our proposed backtracking based subgraph matching algorithm. Particularly, we use the reinforcement learning-based query vertex

Table 5.2: Datasets Properties

Dataset	Name	$\|V\|$	$\|E\|$	$\|L\|$	d
Citeseer	*cs*	3,327	4,732	6	1.4
Yeast	*ye*	3,112	12,519	71	8.0
DBLP	*db*	317,080	1,049,866	15	6.6
Youtube	*yt*	1,134,890	2,987,624	25	5.3
Wordnet	*wn*	76,853	120,399	5	3.1
EU2005	*eu*	862,664	16,138,468	40	37.4

ordering model proposed in Section 5.3 to generate the matching order of each individual query. Regarding the candidate set vertex generation and enumeration procedure, we use the corresponding implementations of Hybrid.

All baseline methods are implemented by the authors of [95] in C++ [1]. The machine learning part of RL-QVO is implemented using Pytorch which could be computed on GPUs, while the candidate generation and enumeration methods are adapted from the code of [95] in C++.

Experiment Environment. We conducted the experiments on the servers running RHEL 7.7 system, which have Intel Xeon Gold 6238R 2.2GHz 28cores CPU and NVIDIA Quadro RTX 5000 GPU with 88GB RAM (Six Channel).

Data Graph. We conducted the experiments on six real-life datasets. The number of vertices varies from $3,112$ to $1,134,890$.

We obtain the datasets used by previous works [95]. Six real-life graphs can be classified into five different categories: Citeseer is a citation network, Yeast is a biology network, DBLP and Youtube are social networks, Wordnet is a lexical network and EU2005 is a web network. The detailed properties of the datasets are summarized in Table 5.2.

Query Graph. Following the settings in [95], query graphs are generated for

[1] `https://github.com/RapidsAtHKUST/SubgraphMatching`

Table 5.3: Query sets

Dataset	Query Set	Default
Citeseer, Yeast, Youtube, Wordnet	q_4, q_8, q_{16}	q_{16}
DBLP, EU2005	q_4, q_8	q_8

each data graph by randomly extracting **connected** subgraphs from G. We generate 200 labeled query graphs with $\{4, 8, 16\}$ vertices denoted as q_4, q_8 and q_{16} for Citeseer, Yeast, Youtube and Wordnet datasets, and query graphs with 4 and 8 vertices for DBLP and EU2005. 50% of the generated query graphs are used for training, and the remaining graphs are used for testing. Table 5.3 lists the query sets for each dataset. By default, we report the total time cost results for query graph set with the greatest numbers of vertices, *i.e.,* q_8 for DBLP and EU2005 and q_{16} for Citeseer, Yeast, Youtube and Wordnet.

Experiment settings. In this experiment, RL-QVO has two layers of graph convolutional networks to obtain the representations for the query nodes. After GCN layers, there is a two-layer linear neural network that produces the selection probability for each node. The learning rate is set to 0.01, the output dimension of GCN is set to 64, number of training epochs is 100 for DBLP, Youtube, Wordnet and EU2005, and 300 for Citeseer and Yeast. We also apply a dropout ratio at 0.2 during training, and the discount factor γ is tune in $(0, 1)$. We set a time limit 500 seconds for subgraph matching, if the query process exceeds the time limit during training, we skip this query graph for training to save experimental time. During evaluation, if a compared algorithm cannot finish the query within the time limit, we denote the query graph as a unsolved query and assign the time cost as 500 secs for this query. If a query graph remains unsolved by all compared methods, we would exclude this query graph in computing the average query processing time and enumeration time. We also count the numbers of unsolved queries to compare the capabilities in solving the worse case queries

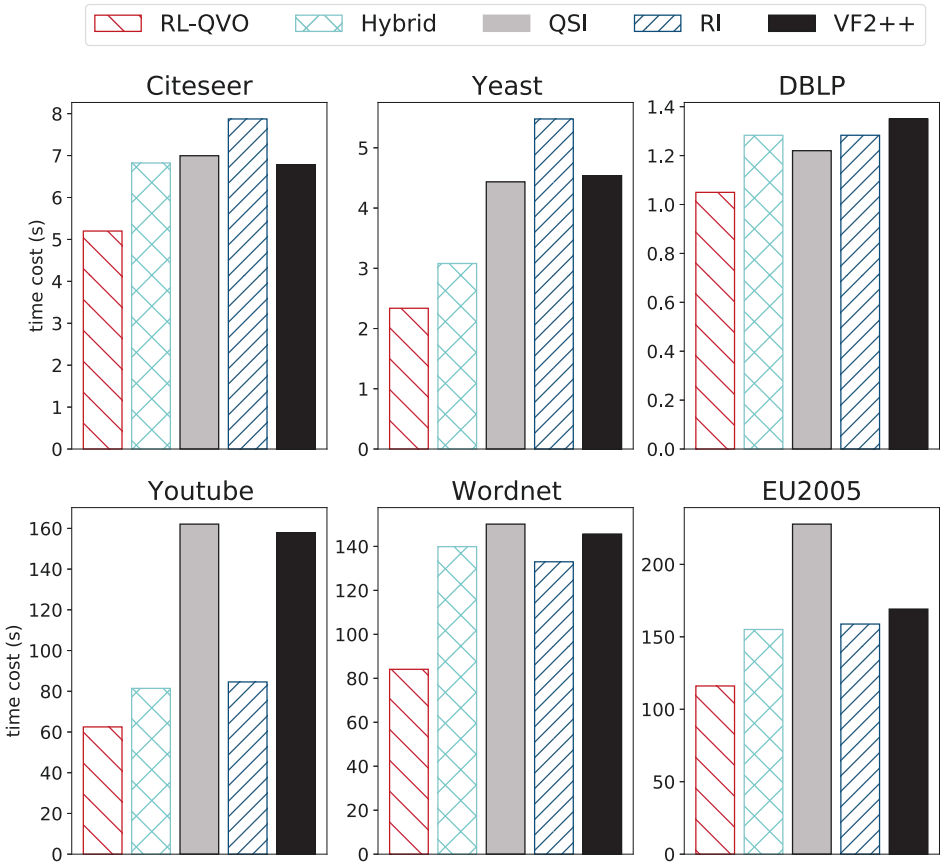


Figure 5.3: Average Query Processing Time Comparison

for all compared methods. The results are shown in Section 5.4.2.

Evaluation Metrics. In this experiment, we compare the time efficiency of compared methods. We use the average query processing time (i.e., the query response time) and enumeration time (i.e., the time used in enumeration procedure) for the comparison. Recall that we say a query is unsolved if it cannot be finished within 500 seconds. We also report the training time and the memory consumption of the our model.

5.4.2 Query Processing Time Comparison

In the first set of experiments, we evaluate the query processing time for all compared subgraph matching algorithms with default settings, where 6 real-life graphs are deployed. Note that the query processing time t includes the filtering methods time cost t_{filter}, the ordering time t_{order} and enumeration time t_{enum}, i.e., $t = t_{filter} + t_{order} + t_{enum}$.

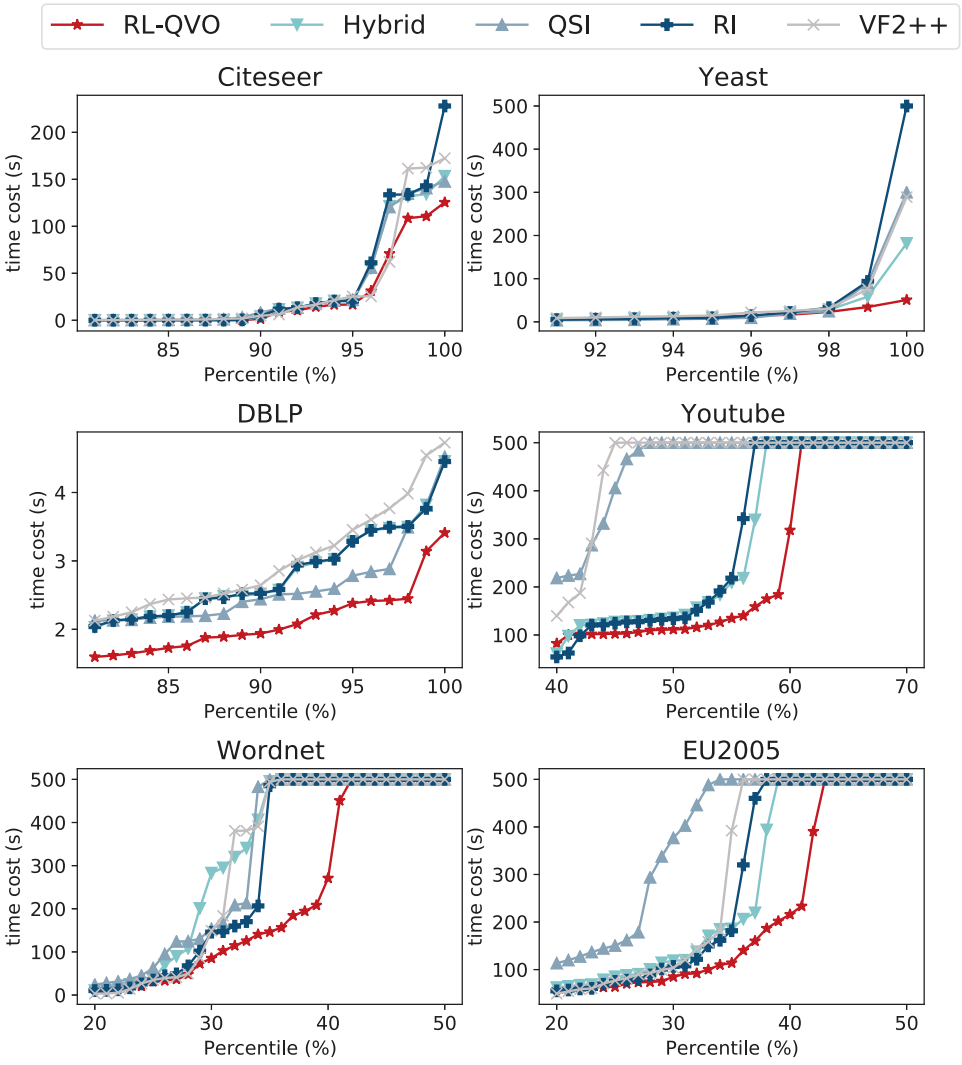


Figure 5.4: Query Processing Time Percentile Comparison

Average Query Processing Time. We first report the average query processing time for the given query graphs, and the results are illustrated in Figure 5.3. It is shown that RL-QVO consistently outperforms the competitors because of the high quality matching order obtained by our advanced model. In the experiments of Section 5.4.3, we further justify this is a significant achievement where the optimal matching order is considered. Our experiments also confirm the claim in [95] which shows that, among the existing algorithms, Hybrid has the best performance on most of the data graphs expect DBLP and Wordnet.

Cumulative Query Processing Time Distribution. To better understand the query processing time distribution, Figure 5.4 details the query processing time comparison by showing a cumulative distribution of the query processing time for all compared methods. The gaps of time cost between RL-QVO and other compared methods grow with percentile, which shows the efficiency of RL-QVO in handling the hard queries. This is a big advantage of RL-QVO compared to other competitors because the response time of queries at high percentile (i.e., hard queries) is critical for the through-output of the system in many industry applications.

Number of Unsolved Queries. Figure 5.4 also demonstrates the number of unsolved query graphs with default query sets for Youtube, Wordnet and EU2005 data graphs. Note that we set the query time of each unsolved query to 500 seconds (preset time limit). It is shown that RL-QVO has much less number of unsolved queries compare to other algorithms.

5.4.3 Enumeration Time Comparison

The enumeration time is the determining factor of the total time cost of subgraph matching process, and also directly reflects the qualities of matching orders generated by different algorithms. We compare the enumeration time of Hybrid,

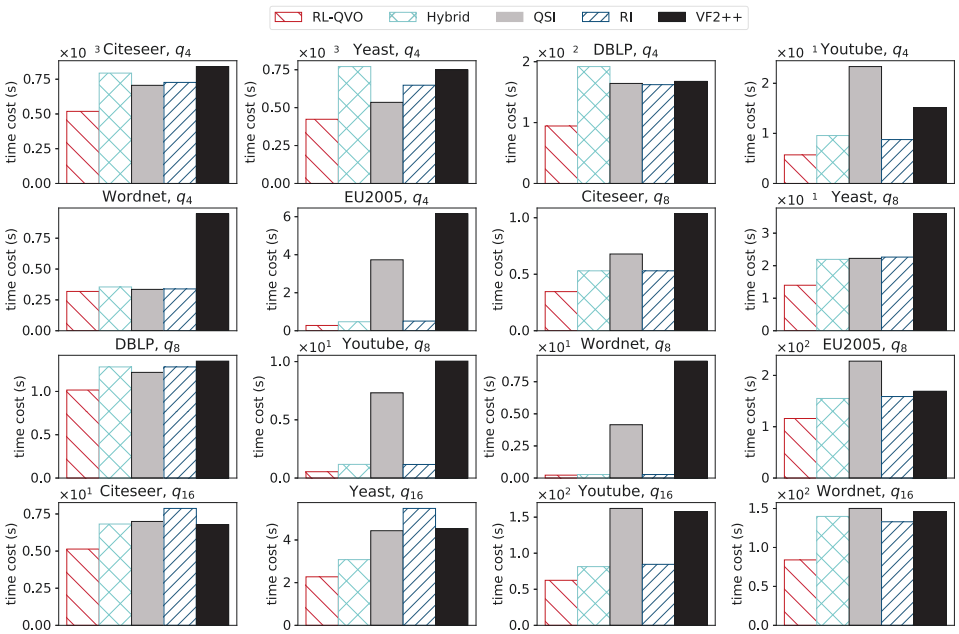


Figure 5.5: Average enumeration time comparison with varying query sizes

QSI, RI and VF2++ with RL-QVO to examine the qualities of the matching orders generated by these methods. Since all these methods utilize the same enumeration methods which are implemented in the same way, the enumeration time costs could directly reflect the quality of the output matching orders.

Average Enumeration Time. The average enumeration time with varying query sizes are shown in Figure 5.5. It is reported that our proposed RL-QVO outperforms all baseline methods on all datasets with all query sizes. Our proposed RL-QVO could improve the enumeration time up to 1.8× compared to the *best* performed baseline method. The performance gaps between RL-QVO and other baseline methods become much more significant with growing query vertex numbers, which indicates that RL-QVO has formidable ability in handling ordering problem in large search space.

Comparison with Optimal Matching Order. To better investigate the goodness and the potential improvement space of the matching orders obtained

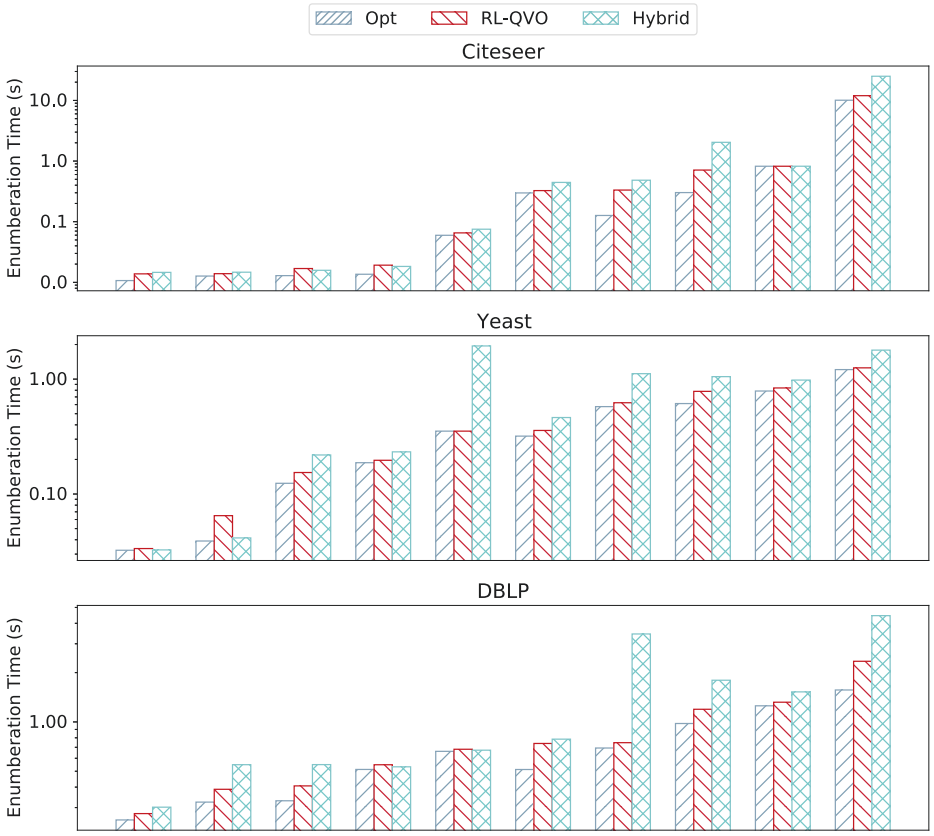


Figure 5.6: Enumeration time spectrum analysis with permutation orders

by the algorithms, we also consider the optimal matching orders in the experiments. As it is time prohibitive to find the optimal matching orders for large size queries and data graphs. we only consider the settings where there are no unsolved queries for RL-QVO and Hybrid. Particularly, we conduct the spectrum analysis on Citeseer, Yeast and DBLP datasets for subgraph matching query with 10 randomly selected query graphs with 8 vertices (q_8). To obtain the *optimal matching order*, we generate the orders of all permutations of the query vertices, and feed them into the subgraph matching algorithm with the same filtering and enumeration methods as RL-QVO and Hybrid.

We compare the enumeration time of RL-QVO and Hybrid to the optimal

Table 5.4: Training Time (in hour)

Datasets	**q4**	**q8**	**q16**
cs	0.51	3.03	11.96
ye	0.54	1.56	12.18
db	1.02	2.53	-
yt	3.67	8.31	78.97
wn	0.75	3.77	30.79
eu	8.68	49.27	-

matching order based algorithm, denoted by **Opt**. Note that all three algorithms use the same filtering and enumeration implementations. The spectrum analysis results are illustrated in Figure 5.6 where the bars indicate the enumeration time costs of the optimal order, RL-QVO's order and Hybrid's order. The results show that compared to the ordering methods of Hybrid, RL-QVO is more likely to generate *near optimal* orders for matching, thus have better overall search performance. Moreover, considering the gaps between *Opt* and Hybrid for these queries in Figure 5.6, we boast that that RL-QVO makes a significant improvement in terms of enumeration time (i.e., the quality of the matching order).

5.4.4 Training Time and Order Inference Time

In this subsection, we evaluate the model training time and matching order inference time.

Training time. The time required to train the proposed model is reported in Table 5.4. The time costs reported are for default settings, *i.e.,* 100 training epochs on 100 training query graphs for all datasets except Citeseer and Yeast which have 300 training epochs. During training procedure, if a training query graph requires more than 500 second to get matching result, we skip this query instance in the following training process to save the training time cost. Because

Table 5.5: Space Evaluation

Dataset	Graph Space	Model Space
Citeseer	112.4 kB	186.2 kB
Yeast	260.8 kB	186.2 kB
DBLP	30.4 MB	186.2 kB
Youtube	89.7 MB	186.2 kB
Wordnet	3.5 MB	186.2 kB
EU2005	437.6 MB	186.2 kB

RL-QVO utilizes the policy gradient descend and PPO training strategy which require the interaction between the agent and environment, the major training time cost comes from the enumeration process.

Matching order inference time. As analyzed in Section 5.3.6, the computational complexity of RL-QVO for query vertex ordering generation is $O(|V(q)| \times (|E(q)| + d^2))$. In our experiments, RL-QVO could produce the matching order within $10ms$ for each query.

5.4.5 Space Evaluation

As discussed in Section 5.3.6, the space complexity for RL-QVO is fixed with growing sizes of query and data graphs. Here, we report the exact space requirements for storing the parameters of the network and corresponding data graphs. The results are demonstrated in Table 5.5. Even for data graph like EU2005 which requires 437.6 MB to store, RL-QVO only needs 186.2 kB to save parameters while achieving outstanding performance in query vertex ordering. In terms of the memory consumption, RL-QVO might require more space for query graphs, however, the memory cost is still dominated by the parameter space of the model, which is relatively small.

5.5 Conclusion

Subgraph matching is a fundamental research topic in database and data mining communities, which is a NP-Complete problem. One important branch of the subgraph matching algorithms follows the backtracking search paradigm, and it is shown that the search performance is heavily affected by the quality of the matching order used in the search. We notice that existing ordering methods with heuristic strategies are lacking the capability to fully exploit the structural and label information to generate high-quality matching order (i.e., query vertex order). In this chapter, we proposed RL-QVO, a reinforcement learning-based model for query vertex ordering. RL-QVO learns the policy to generate the matching order considering both graph structure information and the long-term benefits. Extensive experiments proves the efficiency of RL-QVO.

Chapter 6

EPILOGUE

In this chapter, we summarize the works presented in this thesis and describe possible future research directions. In this thesis, we focus on designing machine learning models for graph analytics. The main contributions of this thesis can be concluded as follows:

- **Efficient and Compact Graph Neural Network.** We design a binarized graph neural network which significantly reduces the both time and efficient complexity of graph neural networks with acceptable accuracy sacrifice. The proposed BGN has binarized parameters and enables GNNs to learn discrete embedding. The binarized neural network can reduce the memory and time cost of the GNN such that increases the scalability of GNNs.

- **Models for Graph of Graphs.** We propose two novel graph neural network models GoGNN and PGON for graph of graphs. By mining the hierarchical structure of graph of graphs, our proposed models could outperform state-of-the-art baselines with regards to the graph classification and structured entity interaction prediction tasks. In particular, GoGNN is

the first work that fully exploits the graph of graphs structure and PGON has the equivalent expressive power as 2-WL test which could capture much detailed graph information.

- **Query Vertex Ordering for Subgraph Matching.** In this thesis, we also propose a reinforcement learning based query vertex ordering method for subgraph matching. Our model optimizes the order generation process of subgraph matching to overcome the bottlenecks of existing methods that can only select the ordered vertex on a greedy search basis.

Future Work and Research Opportunities. There are still some open problems and opportunities that need further research.

- **Designing Machine Learning Models for Graph Queries.** There are numerous other graph analytic tasks that could be optimized by machine learning techniques. For example, we could develop learning-based index for graph queries.

- **Machine Learning Techniques on Join-based Subgraph Matching.** The other important branch to solve the subgraph matching problem is the join-based methods, in which the join order plays a critical role. We can optimize the join order selection by applying the machine learning techniques.

- **Design Graph Neural Networks to Preserve Structural Information.** In this thesis, our models are designed to exploit the graph structure rather than fully capture the graph structural information. However, the structure of the network are usually crucial in various applications. Therefore, it would be interesting and important to develop graph neural networks that are able to fully preserve the graph structural information.

Bibliography

[1] M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn, "Design and implementation of the logicblox system," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, T. K. Sellis, S. B. Davidson, and Z. G. Ives, Eds. ACM, 2015, pp. 1371–1382.

[2] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, "Simgnn: A neural network approach to fast graph similarity computation," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019*, 2019, pp. 384–392.

[3] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 2017, pp. 459–468.

[4] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[5] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, 2020.

[6] B. Bhattarai, H. Liu, and H. H. Huang, "CECI: compact embedding cluster index for scalable subgraph matching," in *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, Eds. ACM, 2019, pp. 1447–1462.

[7] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, "Efficient subgraph matching by postponing cartesian products," in *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, F. Özcan, G. Koutrika, and S. Madden, Eds. ACM, 2016, pp. 1199–1214.

[8] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro, "A subgraph isomorphism algorithm and its application to biochemical data," *BMC bioinformatics*, vol. 14, no. 7, pp. 1–13, 2013.

[9] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H. Kriegel, "Protein function prediction via graph kernels," in *Proceedings Thirteenth International Conference on Intelligent Systems for Molecular Biology 2005, Detroit, MI, USA, 25-29 June 2005*, 2005, pp. 47–56.

[10] J.-Y. Cai, M. Fürer, and N. Immerman, "An optimal lower bound on the number of variables for graph identification," *Combinatorica*, vol. 12, no. 4, pp. 389–410, 1992.

[11] V. Carletti, P. Foggia, A. Saggese, and M. Vento, "Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 804–818, 2017.

[12] V. Carletti, P. Foggia, and M. Vento, "VF2 plus: An improved version of VF2 for biological graphs," in *Graph-Based Representations in Pattern Recognition - 10th IAPR-TC-15 International Workshop, GbRPR 2015, Beijing, China, May 13-15, 2015. Proceedings*, ser. Lecture Notes in Computer Science, C. Liu, B. Luo, W. G. Kropatsch, and J. Cheng, Eds., vol. 9069. Springer, 2015, pp. 168–177.

[13] Z. Chen, L. Chen, S. Villar, and J. Bruna, "Can graph neural networks count substructures?" *arXiv preprint arXiv:2002.04025*, 2020.

[14] X. Chu, Y. Lin, Y. Wang, L. Wang, J. Wang, and J. Gao, "Mlrda: a multi-task semi-supervised learning framework for drug-drug interaction prediction," in *IJCAI*, 2019.

[15] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1367–1372, 2004.

[16] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *NIPS*, 2016.

[17] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 5, pp. 833–852, 2019.

[18] G. D'Agostino and A. Scala, *Networks of networks: the last frontier of complexity*. Springer, 2014, vol. 340.

[19] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 6351–6361.

[20] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv:1810.04805*, 2018.

[22] K. Do, T. Tran, and S. Venkatesh, "Graph transformation policy network for chemical reaction prediction," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, Eds. ACM, 2019, pp. 750–760.

[23] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *Journal of molecular biology*, vol. 330, no. 4, pp. 771–783, 2003.

[24] G. Dong, J. Gao, R. Du, L. Tian, H. E. Stanley, and S. Havlin, "Robustness of network of networks under targeted attack," *Physical Review E*, vol. 87, no. 5, p. 052804, 2013.

[25] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[26] R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin, "LIBLINEAR: A library for large linear classification," *JMLR*, vol. 9, pp. 1871–1874, 2008.

[27] H. Gao and S. Ji, "Graph u-nets," *arXiv preprint arXiv:1905.05178*, 2019.

[28] ——, "Graph u-nets," in *ICML 2019*. PMLR, 2019.

[29] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[30] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 15 554–15 566.

[31] M. Grohe, *Descriptive complexity, canonisation, and definable graph structure theory*. Cambridge University Press, 2017, vol. 47.

[32] M. Grohe and M. Otto, "Pebble games and linear equations," *The Journal of Symbolic Logic*, pp. 797–844, 2015.

[33] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *ACM SIGKDD*, 2016, pp. 855–864.

[34] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017, pp. 1024–1034.

[35] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Eng. Bull.*, vol. 40, no. 3, pp. 52–74, 2017.

[36] M. Han, H. Kim, G. Gu, K. Park, and W. Han, "Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together," in *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, Eds. ACM, 2019, pp. 1429–1446.

[37] W.-S. Han, J. Lee, and J.-H. Lee, "Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 337–348.

[38] S. Harada, H. Akita, M. Tsubaki, Y. Baba, I. Takigawa, Y. Yamanishi, and H. Kashima, "Dual graph convolutional neural network for predicting chemical networks," *BMC bioinformatics*, vol. 21, pp. 1–13, 2020.

[39] H. He and A. K. Singh, "Graphs-at-a-time: query language and access methods for graph databases," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, J. T. Wang, Ed. ACM, 2008, pp. 405–418.

[40] S. Hu, Z. Xiong, M. Qu, X. Yuan, M. Côté, Z. Liu, and J. Tang, "Graph policy network for transferable active learning on graphs," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[41] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, 2018, pp. 4563–4572.

[42] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *NIPS*, 2016, pp. 4107–4115.

[43] A. Jüttner and P. Madarasi, "Vf2++—an improved subgraph isomorphism algorithm," *Discrete Applied Mathematics*, vol. 242, pp. 69–81, 2018.

[44] S. M. Kazemi and D. Poole, "Simple embedding for link prediction in knowledge graphs," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, 2018, pp. 4289–4300.

[45] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann, "Benchmark data sets for graph kernels," 2016. [Online]. Available: http://graphkernels.cs.tu-dortmund.de

[46] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[47] ——, "Semi-supervised classification with graph convolutional networks," in *ICLR 2017*, 2017.

[48] V. Kurin, S. Godil, S. Whiteson, and B. Catanzaro, "Improving sat solver heuristics with graph networks and reinforcement learning," *arXiv preprint arXiv:1909.11830*, 2019.

[49] S. Kwon and S. Yoon, "Deepcci: End-to-end deep learning for chemical-chemical interaction prediction," in *ACM BCB*, 2017.

[50] L. Lai, L. Qin, X. Lin, and L. Chang, "Scalable subgraph enumeration in mapreduce," *Proc. VLDB Endow.*, vol. 8, no. 10, pp. 974–985, 2015.

[51] L. Lai, L. Qin, X. Lin, Y. Zhang, and L. Chang, "Scalable distributed subgraph enumeration," *Proc. VLDB Endow.*, vol. 10, no. 3, pp. 217–228, 2016.

[52] L. Lai, Z. Qing, Z. Yang, X. Jin, Z. Lai, R. Wang, K. Hao, X. Lin, L. Qin, W. Zhang, Y. Zhang, Z. Qian, and J. Zhou, "Distributed subgraph matching on timely dataflow," *Proc. VLDB Endow.*, vol. 12, no. 10, pp. 1099–1112, 2019.

[53] G. Landrum, "Rdkit: A software suite for cheminformatics, computational chemistry, and predictive modeling," 2013.

[54] J. Lee, W. Han, R. Kasperovics, and J. Lee, "An in-depth comparison of subgraph isomorphism algorithms in graph databases," *Proc. VLDB Endow.*, vol. 6, no. 2, pp. 133–144, 2012.

[55] J. B. Lee, R. Rossi, and X. Kong, "Graph classification using structural attention," in *SIGKDD*. ACM, 2018, pp. 1666–1674.

[56] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," *arXiv preprint arXiv:1904.08082*, 2019.

[57] J. Li, Y. Rong, H. Cheng, H. Meng, W. Huang, and J. Huang, "Semi-supervised graph classification: A hierarchical graph perspective," in *WWW 2019*, 2019.

[58] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, 2019, pp. 3835–3845.

[59] ——, "Graph matching networks for learning the similarity of graph structured objects," *arXiv preprint arXiv:1904.12787*, 2019.

[60] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[61] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, pp. 537–546.

[62] D. Lian, Q. Liu, and E. Chen, "Personalized ranking with importance sampling," in *WWW 2020*, 2020, p. 1093–1103.

[63] D. Lian, H. Wang, Z. Liu, J. Lian, E. Chen, and X. Xie, "Lightrec: A memory and search-efficient recommender system," in *WWW 2020*, 2020, p. 695–705. [Online]. Available: https://doi.org/10.1145/3366423.3380151

[64] D. Lian, K. Zheng, V. W. Zheng, Y. Ge, L. Cao, I. W. Tsang, and X. Xie, "High-order proximity preserving information network hashing," in *ACM SIGKDD*, 2018, pp. 1744–1753.

[65] H. Liu, R. Wang, S. Shan, and X. Chen, "Deep supervised hashing for fast image retrieval," in *CVPR*, 2016, pp. 2064–2072.

[66] W. Liu, C. Mu, S. Kumar, and S. Chang, "Discrete graph hashing," in *NIPS*, 2014, pp. 3419–3427.

[67] X. Liu, H. Pan, M. He, Y. Song, X. Jiang, and L. Shang, "Neural subgraph isomorphism counting," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1959–1969.

[68] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, and Y. Qi, "Geniepath: Graph neural networks with adaptive receptive paths," in *AAAI*, vol. 33, 2019, pp. 4424–4431.

[69] Z. Lou, J. You, C. Wen, A. Canedo, J. Leskovec *et al.*, "Neural subgraph matching," *arXiv preprint arXiv:2007.03092*, 2020.

[70] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 2153–2164.

[71] D. Mawhirter, S. Reinehr, C. Holmes, T. Liu, and B. Wu, "Graphzero: A high-performance subgraph matching system," *ACM SIGOPS Operating Systems Review*, vol. 55, no. 1, pp. 21–37, 2021.

[72] J. Menche, A. Sharma, M. Kitsak, S. D. Ghiassian, M. Vidal, J. Loscalzo, and A.-L. Barabási, "Uncovering disease-disease relationships through the incomplete interactome," *Science*, vol. 347, no. 6224, p. 1257601, 2015.

[73] L. Meng and J. Zhang, "Isonn: Isomorphic neural network for graph representation learning and classification," *CoRR*, vol. abs/1907.09495, 2019.

[74] A. Mhedhbi, M. Lissandrini, L. Kuiper, J. Waudby, and G. Szárnyas, "Lsqb: a large-scale subgraph query benchmark," in *Proceedings of the*

4th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA), 2021, pp. 1–11.

[75] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013, pp. 3111–3119.

[76] H. Q. Ngo, "Worst-case optimal join algorithms: Techniques, results, and open problems," in *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, J. V. den Bussche and M. Arenas, Eds. ACM, 2018, pp. 111–124.

[77] J. Ni, H. Tong, W. Fan, and X. Zhang, "Inside the atoms: ranking on a network of networks," in *SIGKDD*, 2014, pp. 1356–1365.

[78] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *ACMSIGKDD*. ACM, 2014, pp. 701–710.

[79] N. Przulj, D. G. Corneil, and I. Jurisica, "Efficient estimation of graphlet frequency distributions in protein-protein interaction networks," *Bioinform.*, vol. 22, no. 8, pp. 974–980, 2006.

[80] J. Qin, Y. Wang, C. Xiao, W. Wang, X. Lin, and Y. Ishikawa, "GPH: similarity search in hamming space," in *IEEE ICDE*, 2018, pp. 29–40.

[81] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *ACM WSDM*, 2018, pp. 459–467.

[82] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[83] C. Ren, L. An, Z. Gu, Y. Wang, and Y. Gao, "Rebalancing the car-sharing system with reinforcement learning," *World Wide Web*, vol. 23, no. 4, pp. 2491–2511, 2020.

[84] E. Rome, P. Langeslag, and A. Usov, "Federated modelling and simulation for critical infrastructure protection," in *Networks of networks: the last frontier of complexity*. Springer, 2014, pp. 225–253.

[85] J. Y. Ryu, H. U. Kim, and S. Y. Lee, "Deep learning improves prediction of drug–drug and drug–food interactions," *PNAS*, vol. 115, no. 18, pp. E4304–E4311, 2018.

[86] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu, "The ubiquity of large graphs and surprising challenges of graph processing," *Proc. VLDB Endow.*, vol. 11, no. 4, pp. 420–431, 2017.

[87] R. Salakhutdinov and G. E. Hinton, "Semantic hashing," *Int. J. Approx. Reasoning*, vol. 50, no. 7, pp. 969–978, 2009.

[88] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *ESWC*, 2018.

[89] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[90] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[91] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verification hardness: an efficient algorithm for testing subgraph isomorphism," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 364–375, 2008.

[92] F. Shen, C. Shen, W. Liu, and H. T. Shen, "Supervised discrete hashing," in *CVPR*, 2015, pp. 37–45.

[93] X. Shen, S. Pan, W. Liu, Y. Ong, and Q. Sun, "Discrete network embedding," in *IJCAI*, 2018, pp. 3549–3555.

[94] T. A. Snijders, P. E. Pattison, G. L. Robins, and M. S. Handcock, "New specifications for exponential random graph models," *Sociological methodology*, vol. 36, no. 1, pp. 99–153, 2006.

[95] S. Sun and Q. Luo, "In-memory subgraph matching: An in-depth study," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1083–1098.

[96] S. Sun, X. Sun, Y. Che, Q. Luo, and B. He, "Rapidmatch: a holistic approach to subgraph query processing," *Proceedings of the VLDB Endowment*, vol. 14, no. 2, pp. 176–188, 2020.

[97] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, "Efficient subgraph matching on billion node graphs," *Proc. VLDB Endow.*, vol. 5, no. 9, pp. 788–799, 2012.

[98] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[99] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 5998–6008.

[100] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[101] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[102] C. Wang, S. Pan, R. Hu, G. Long, J. Jiang, and C. Zhang, "Attributed graph clustering: A deep attentional embedding approach," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 2019, pp. 3670–3676.

[103] H. Wang, D. Lian, Y. Zhang, L. Qin, X. He, Y. Lin, and X. Lin, "Binarized graph neural network," *World Wide Web*, pp. 1–24, 2021.

[104] H. Wang, D. Lian, Y. Zhang, L. Qin, and X. Lin, "Gognn: Graph of graphs neural network for predicting structured entity interactions," *arXiv preprint arXiv:2005.05537*, 2020.

[105] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, "A survey on learning to hash," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 769–790, 2018.

[106] D. Weininger, A. Weininger, and J. L. Weininger, "Smiles. 2. algorithm for generation of unique smiles notation," *J CHEM INF COMP SCI*, vol. 29, no. 2, pp. 97–101, 1989.

[107] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[108] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, 2020.

[109] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[110] ——, "How powerful are graph neural networks?" in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[111] N. Xu, P. Wang, L. Chen, J. Tao, and J. Zhao, "Mr-gnn: Multi-resolution and dual graph neural network for predicting structured entity interactions," *arXiv:1905.09558*, 2019.

[112] X. Yan, P. S. Yu, and J. Han, "Graph indexing: A frequent structure-based approach," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, 2004, pp. 335–346.

[113] H. Yang, S. Pan, P. Zhang, L. Chen, D. Lian, and C. Zhang, "Binarized attributed network embedding," in *IEEE ICDM*, 2018, pp. 1476–1481.

[114] J. You, B. Liu, Z. Ying, V. S. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Ad-*

vances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 6412–6422.

[115] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, 2018, pp. 5171–5181.

[116] ——, "Link prediction based on graph neural networks," in *NIPS*, 2018, pp. 5165–5175.

[117] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *AAAI*, 2018.

[118] X. Zhang, H. Liu, Q. Li, and X. Wu, "Attributed graph clustering via adaptive graph convolution," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 2019, pp. 4327–4333.

[119] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *CoRR*, vol. abs/1606.06160, 2016.

[120] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *Aaai*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[121] M. Zitnik, M. Agrawal, and J. Leskovec, "Modeling polypharmacy side effects with graph convolutional networks," *Bioinformatics*, vol. 34, no. 13, pp. i457–i466, 2018.