Subgraph Classification through Neighborhood Pooling

by

Shweta Ann Jacob

A thesis submitted to the School of
Graduate and Postdoctoral Studies in
partial fulfillment of the requirements for
the degree of

Master of Science in Computer Science

Faculty of Science
University of Ontario Institute of Technology (Ontario Tech University)
Oshawa, Ontario, Canada

June 2023

Thesis Examination Information

Submitted by: **Shweta Ann Jacob**

Master of Science in Computer Science

Thesis title: Subgraph Classification through Neighborhood Pooling

An oral defense of this thesis took place on June 6, 2023 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Shahram S. Heydari
Research Supervisor	Dr. Amirali Salehi-Abari
Examining Committee Member	Dr. Julie Thorpe
Thesis Examiner	Dr. Ying Zhu

The above committee determined that the thesis is in acceptable form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

Abstract

Subgraph classification is an emerging field in graph representation learning where the task is to classify a group of nodes (i.e., a subgraph) within a graph. Graph neural networks (GNNs) are the de facto solution for node, link, and graph-level tasks but fail to perform well on subgraph classification tasks. Even GNNs tailored for graph classification are not directly transferable to subgraph classification as they ignore the external topology of the subgraph, thus failing to capture how the subgraph is located within the larger graph. Existing models for subgraph classification address this shortcoming through labeling tricks or multiple message-passing channels, both of which impose a computation burden and are not scalable to large graphs. To address the scalability issue while maintaining generalization, we propose *Stochastic Subgraph Neighborhood Pooling (SSNP)*, which jointly aggregates the subgraph and its neighborhood (i.e., external topology) information without any computationally expensive operations such as labeling tricks. To improve scalability and generalization further, we also propose a simple data augmentation pre-processing step for *SSNP* that creates multiple sparse views of the subgraph neighborhood. We show that our model is more expressive than plain GNNs, without using any labeling tricks. Our extensive experiments demonstrate that our models outperform current state-of-the-art methods (with a margin of up to 2%) while being up to 3× faster in training.

Author's Declaration

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

Shweta Ann Jacob

Statement of Contributions

I hereby certify that I have been the primary contributor of this thesis by developing the algorithms, implementing them, and designing the experiments. I have also written most content of this thesis. However, some texts of this thesis are borrowed from a currently-under-review preprint [38] that is coauthored by my thesis supervisor Dr. Amirali Salehi-Abari, a collaborator, and me.

Acknowledgments

I would like to express my gratitude to my supervisor, Dr. Amirali Salehi-Abari for his guidance and critical insights that have helped in shaping my research. I would like to especially thank my parents, Appa and Amma, for their support and words of encouragement throughout this journey. I am incredibly grateful for their unconditional love. I would also like to thank my sister, Shilpa, for always being my confidante. I would also like to thank Paul for always being there for me and supporting me every step of the way wholeheartedly. Lastly, I would like to thank everyone who has supported me from near and far.

Contents

List of Figures

List of Tables

List of Abbreviations

BFS ..... Breadth-First Search

DE ..... Distance Encoding

DFS ..... Depth-First Search

DRNL ..... Double-Radius Node Labeling

FN ..... False Negative

FP ..... False Positive

GIC ..... Graph InfoClust

GCL ..... Graph Contrastive Learning

GCN ..... Graph Convolutional Network

GNN ..... Graph Neural Network

MLP ..... Multilayer Perceptron

MPGNN ..... Message Passing Graph Neural Network

OV ..... Online Views

POV ..... Pre-processed Online Views

PV ..... Pre-processed Views

RPE ..... Relative Position Encoding

S2N ..... Subgraph-To-Node

SGRL ..... Subgraph Representation Learning

SSNP ..... Stochastic Subgraph Neighborhood Pooling

TP ..... True Positive

WL ..... Weisfeiler-Lehman

List of Notations

h ..... Length of random walk

k ..... Number of random walks

σ ..... Activation function

$\mathbf{h}$ ..... Hidden representation of node

$\mathbf{q}_s$ ..... Output representation of subgraph S

$\mathbf{W}$ ..... Learnable weight matrix

$\oplus$ ..... Concatenation operator

G ..... Base graph

V ..... Set of vertices

E ..... Set of edges

S ..... Subgraph

V_S ..... Set of subgraph vertices

E_S ..... Set of subgraph edges

V_S ..... Set of subgraph neighborhood vertices

$\mathbf{X}$ ..... Initial node feature matrix

$\mathbb{S}_m$ ..... Set of subgraphs in mini-batch m

$\mathbf{A}$ ..... Adjacency matrix

L ..... Number of convolution layers

L_m ..... Max zero-one labels of nodes in the mini-batch m

l ..... l-th convolution layer

$\mathbb{R}$ ..... Set of real numbers

$\mathbf{Z}$ ..... Learnt node embedding matrix

$N(u)$ ..... Neighborhood node set of node u

$N^k(u)$ ..... Sampled neighborhood node set of node u

p ..... In-out hyperparameter of node2vec

q ..... Return hyperparameter of node2vec

$\mathcal{A}$ ..... Set of anchor patches

d ..... Initial node feature dimensionality

Chapter 1

Introduction

In this chapter, we discuss the importance of the subgraph classification task, shortcoming of current solutions and the need for scalable models to address these shortcomings. We also highlight the contributions of the thesis followed by its organization, summary, and impact.

1.1 Motivation

Graph-structured data is prevalent in many domains such as social networks, biological networks (e.g., protein-interaction networks), or technological networks (e.g., information networks or computer networks). Structural properties of graph data have been exploited for drug repurposing/discovery [39, 55], recommender systems [65,77,86], group decision making [64,66], medical diagnosis [6], peer assessment [56], anomaly detection [4,35] and many more. Graph representation learning [30] has continuously progressed in recent years with the advent of more expressive graph neural networks (GNNs) [31, 42, 73, 79, 82], focusing on various downstream tasks such as node classification [17, 21, 26, 100], link prediction [11, 27, 32, 52, 68, 91, 94], and graph

classification [7, 23, 45, 61, 87, 95].

Subgraph classification is an emerging problem in graph representation learning where one intends to predict the properties associated with a group of nodes (i.e., a *subgraph*) of the larger observed *base* graph [5, 75]. Subgraph classification finds application in various domains such as finding toxic (or violence-inciting) communities in social networks, drug discovery, group recommendation, diagnosis of rare diseases, and many others. As subgraphs may contain any number of nodes ranging from one node to all nodes of the base graph, typical downstream tasks (e.g., node classification, link prediction, or graph classification) can be considered as specific instances of subgraph classification.

Subgraph classification, as a more general problem, requires solutions that can learn, combine, and contrast topological properties and the connectivity between the nodes within and outside the subgraph. Learning these complex intra-connectivity and inter-connectivity patterns of the subgraph and the base graph renders this problem challenging. As a result, existing GNN models that perform well on node classification, link prediction, and graph classification does not perform well on subgraph classification [75]. Also, learning solely on segregated subgraphs that ignore the topology of the base graph is shown to be ineffective [75], thus underpinning the importance of the global topology of the base graphs for the subgraph classification task. Recent state-of-the-art work (e.g., GLASS [75] and SubGNN [5]) alleviates this shortcoming of the lack of global topology information through the use of labeling tricks [75] or artificially-crafted message passing channels [5].

While GLASS [75] and SubGNN [5] enhance the expressiveness of subgraph embeddings, their deployed approaches of labeling tricks and additional artificially-crafted message passing channels are computationally intensive, especially when dealing with larger (sub)graphs. In some cases, these computational bottlenecks have

made these approaches require some careful hyperparameter tuning. For example, the performance of the max-zero-one labeling trick in GLASS is sensitive to the batch size and therefore, requires extensive and careful hyperparameter tuning of the batch size. To overcome the computational overhead of GLASS and SubGNN, it is essential to devise a model that can learn the interactions between subgraph nodes and the external nodes without any computationally-costly subgraph-level operations.

1.2 Contribution

This thesis introduces a simpler computationally efficient model for subgraph classification that does not use any labeling trick or artificially fabricated computationally expensive message-passing channels. Operating on the original graph, our model does not require any subgraph extractions. We first utilize transformation layers on the node features of all nodes in the base graph for dimensionality reduction and node feature smoothing/refinement. The transformation layers can be message-passing layers such as GCN [42], GraphSAGE [31], GIN [82], or a simple graph structure-agnostic model such as MLP. Then, for each subgraph, we aggregate the node features of the subgraph and its neighborhood through our proposed *Stochastic Subgraph Neighborhood Pooling (SSNP)* to generate the subgraph embedding, and consequently the subgraph classification output. The addition of subgraph neighborhood information in our pooling function enhances the expressiveness of subgraph embeddings by capturing their external topology within a base graph. We show that our model is more expressive than a *plain GNN* (i.e., a graph neural network such as GCN [42] without any labeling tricks). To prevent neighborhood explosion for large graphs and keep computation under control, our *SSNP* uses random walks to sample the neighborhood of each subgraph. As a data augmentation strategy, our neighborhood sampling

method can be conducted multiple times in a pre-processing stage to create multiple sparse views of the subgraph neighborhood. We conduct comprehensive experiments on real-world datasets to show the performance and scalability of our model against various baselines including the current state-of-the-art GLASS [75]. In all datasets (except one), our model outperforms others with a gain of up to 2% while having a speedup of up to 3× compared to GLASS. Experimental results on real-world datasets demonstrate our model is effective, yet simpler than existing models and computationally efficient. Moreover, the utilization of subgraph neighborhoods in the pooling layer enhances the power of the subgraph representations without the requirement for any labeling trick.

In summary, the main contributions of our thesis are:

- We propose a simpler model for subgraph classification that uses subgraphs and its neighborhood in our *Stochastic Subgraph Neighborhood Pooling (SSNP)* to generate the final subgraph embedding used for subgraph classification.

- We show that our model with *SSNP* is more expressive than a plain GNN in distinguishing non-isomorphic subgraphs.

- We propose a data augmentation strategy that creates multiple views of subgraph neighborhoods.

- Through extensive experiments on real-world datasets, we show that *SSNP* is scalable and outperforms baselines on 3 out of the 4 datasets. Our multi-view hyperparameter sensitivity analyses and experiments on different stochastic pooling strategies further show that our POV variant is effective in generalization with even a small number of views.

1.3 Thesis Organization

The rest of the Thesis is organized as follows. Chapter **2** provides background on graph representation learning, its impact, and its applications. We then discuss various downstream tasks in graph representation learning such as node classification, link prediction and graph classification followed by subgraph classification. In the later section, we draw focus to shallow encoders and other prominent works in Graph Neural Networks such as GCN and GraphSAGE. We also discuss how GCNs can be adapted for subgraph classification. We review state-of-the-art works in subgraph classification such as SubGNN and GLASS. We conclude the chapter with discussions on data augmentations in graphs and graph contrastive learning. In Chapter **3**, we review shallow encoders and MPGNNs. We later on discuss subgraph representation learning approaches followed by previous subgraph classification works. We then discuss scalable SGRLs followed by scalability through sampling. Chapter **4** discusses the subgraph classification problem statement and our Stochastic Subgraph Neighborhood Pooling model. We provide examples of transformation layers in our model and discuss about the expressive power of our model. In the later section, we discuss various multi-view sampling strategies for subgraph neighborhoods. In Chapter **5**, we perform various experiments to show the advantage of using our Stochastic Subgraph Neighborhood Pooling model and its computational advantage. We also provide some hyperparameter sensitivity analysis. In Chapter **6**, we conclude our work with a brief summary of our model and its empirical results followed by possible future research directions.

1.4 Summary and Impact

Subgraph Classification is a new vertical in graph representation learning that focuses on learning the properties of a group of nodes given a large graph. Existing solutions for node, link, or graph-level tasks are not suited for subgraph classification due to the distinctive nature of subgraphs. Subgraphs have internal as well as external topology which is crucial for its classification. To learn this complex topology, previous work in subgraph classification such as SubGNN [5] and GLASS [75] uses subgraph specific message-passing channels or labeling tricks which are computationally costly.

We propose a model that uses subgraph neighborhood information during pooling along with subgraph information. The addition of subgraph neighborhood information is simple and low-cost while increasing the expressiveness power of the underlying graph neural network. We term our pooling function that uses subgraph and its neighborhood as *Stochastic Subgraph Neighborhood Pooling (SSNP)*. To allow for faster computation and avoid noise, we use random-walk sampled subgraph neighborhoods. As random walks create some stochasticity in the model, we use a data augmentation strategy to create multiple views of the random-walk sampled subgraph neighborhoods. We propose three sampling strategies such as Online Views, Preprocessed Views and Pre-processed Online Views.

We conduct experiments on four real-world datasets using our pooling function *SSNP* combined with various transformation layers such as MLP, GCN [42], and Nested Network [69]. We compare our model against various baselines including previous state-of-the-art SubGNN [5] and GLASS [75]. On 3 out of the 4 datasets, our model outperforms all the baselines whereas on the other one, our model performance is comparable to the baselines. We perform another set of experiments to evaluate the scalability of our model in comparison to GLASS. On all the datasets, our model is

much faster than GLASS while being better or comparable in terms of performance. We further conduct hyperparameter sensitivity experiments to understand the effect of the random walk and multi-view parameters introduced in our model. With small random walk length and number of views, our model is able to perform well without being computationally demanding. The experiments show that our POV sampling variant provides the best performance while being computationally-light. The stochasticity introduced by the number of views per epoch parameter allows for generalization of the model without overfitting.

In conclusion, our novel *SSNP* with simple transformation layer along with subgraph neighborhood pooling performs well on subgraph classification task while being scalable (due to node-level operations irrespective of target subgraph) and thereby overcoming the shortcomings of previous works (which require target subgraph specific operations). Additionally, our data augmentation technique allows the model to generalize well without impacting the computation dramatically. It is the position of this thesis that *SSNP* can be adapted to other downstream tasks and can work with large-scale graphs.

Chapter 2

Background

This chapter reviews some essential topics that this thesis is built upon. We first review graph representation learning and its applications. Next, we review the various downstream tasks on graphs. We also discuss two prominent works in graph neural networks, GCN [42] and GraphSAGE [31] and their message aggregation and update formulations. We then review two previous subgraph classification approaches, SubGNN [5] and GLASS [75]. We conclude the chapter with discussions on data augmentations in graphs and graph contrastive learning.

2.1 Graph Representation Learning

Many domains use graph-structured data and therefore, the applications of graph representation learning are widespread. Over the years, the field of graph representation learning has advanced through the use of various graph neural network architectures that focus on various aspects such as performance and scalability. Graph learning has applications in numerous domains and industries ranging from polypharmacy to social networks. In social networks, graph learning can be used for making friend

recommendations [16], community detection, predicting the type of interaction [46], etc. Graph learning also finds applications in natural language processing [18], drug design, software mining [44], modeling polypharmacy side effects, molecular property prediction, anomaly detection [53], etc. Graphs are defined as a set of nodes and edges. As a result, we can operate on graphs at the node-level, link-level, or graph-level.

In the coming subsections, we discuss the three prominent downstream tasks in graph representation learning: node classification, link prediction and graph classification and various graph neural network architectures that target specific tasks or all tasks in general. We then shift our focus to a new task in graph representation learning called subgraph classification and discuss how it is related to other tasks.

2.1.1 Node Classification

In node classification, we wish to predict specific labels (that indicate certain properties) of the nodes. For example, if we consider a social network where the nodes represent users and links represent friendships, the node classification task is to classify each user as a member of a specific political party. Graph Neural Networks such as GCN [42], GraphSAGE [31] learns node representations that can be used for node classification. DropEdge [62] prevents oversmoothing and overfitting in graph neural networks by randomly removing some edges from the graph. DropEdge can be considered a data augmentation technique where edge perturbation happens at each training epoch. Additionally, DropEdge allows for faster aggregations and creates robust representations for node classification. An example for node classification is given in Figure 2.1. Each node is given a label (indicated by its color) and the task is to predict the labels of the unlabeled nodes.

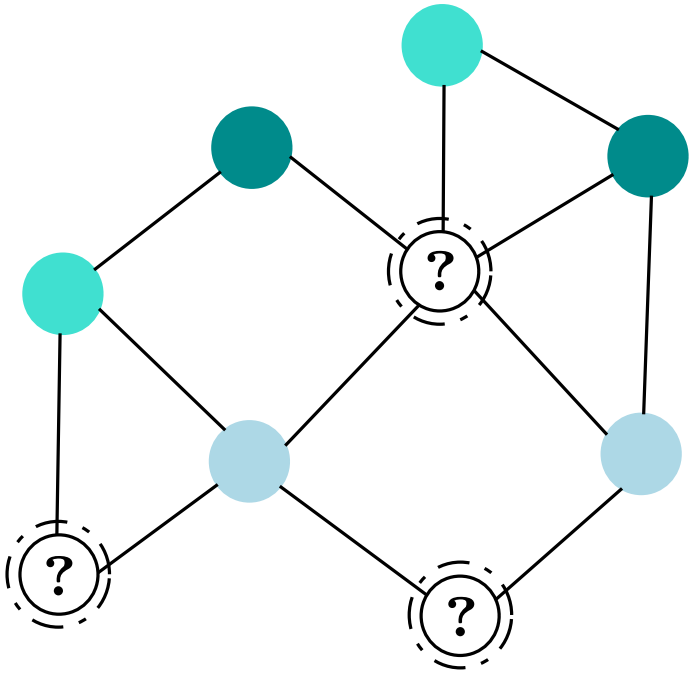


Figure 2.1: Node Classification Example.

2.1.2 Link Prediction

Link prediction also known as relation prediction involves the prediction of absence/existence of a link between two nodes. Node classification works maybe extended to work for downstream tasks such as link prediction. However, link prediction requires modeling interactions between pairs of nodes which is a shortcoming of node-level GNNs. Link prediction is also used for graph completion tasks. For example, in a citation network where nodes represent scientists and links represent co-authorships between two scientists, the link prediction task may be to predict if two scientists may collaborate in the future. Recent works such as PinSage [86] and Decagon [102] are used for making recommendations and predicting drug side effects which are specific cases of link prediction. Other SGRL methods such as SEAL [94] and DE-GNN [48] use enclosing subgraphs along with labeling tricks for link prediction. Labeling trick [97] refers to any labeling function such as double-radius node labeling (where distance of

each node to the target pair of nodes is captured) or zero-one labeling (where target nodes are assigned a label of 1 and all other nodes are assigned a label of 0) that helps the underlying graph neural network to understand the structure of the nodes in the graph. Labeling tricks primarily satisfy two conditions: identifying the target nodes within a node set and permutation equivariance (i.e., the labeling function should assign similar labels to a set of nodes irrespective of their order in the set). DEAL [32] uses two separate encoders: one for attribute and one for structure and aligns the two encoded embeddings using alignment losses for link prediction. An example for link prediction is given in Figure 2.2. The blue-colored nodes are the target nodes and the task is to predict if the two nodes will interact in the future.

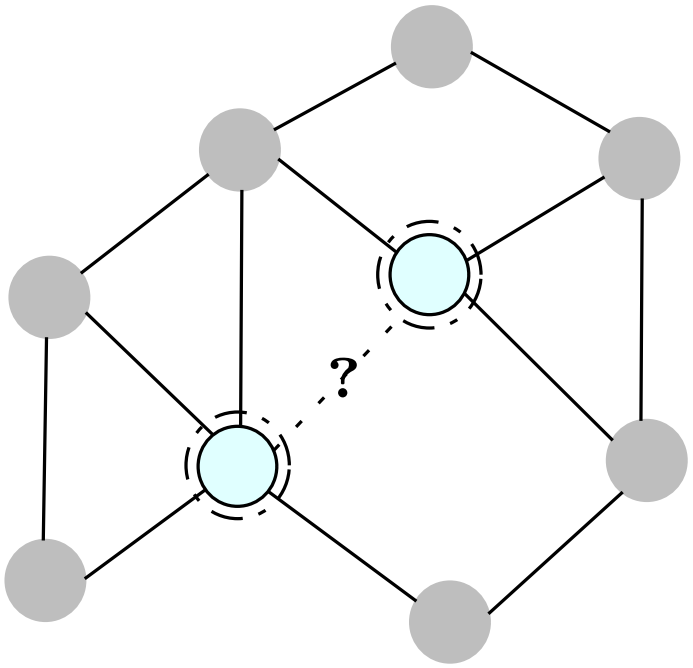


Figure 2.2: Link Prediction Example.

2.1.3 Graph Classification

Graph classification involves predictions for a group of nodes. The task may be to classify a group of proteins as enzymes or non-enzymes or predicting properties of a molecule (which is represented using a graph). GCN [42] and GraphSAGE [31] can be extended to graph classification by adding a graph pooling layer to transform the node-level outputs to a graph-level representation. Works such as [28, 80] focus on graph classification tasks. Other graph neural networks focus on creating better pooling layers to convert node-level outputs to a graph-level output. SAGPool [45] and ASAP [61] uses attention based pooling layer to create graph representations. DiffPool [87] and MinCutPool [7] are clustering based pooling approaches used in graph classification. DGCNN [95] is another prominent work in graph classification. An example for graph classification is given in Figure 2.3. Each graph has a label (indicated by the color of its nodes) and the task is to predict the label of the unseen graph.

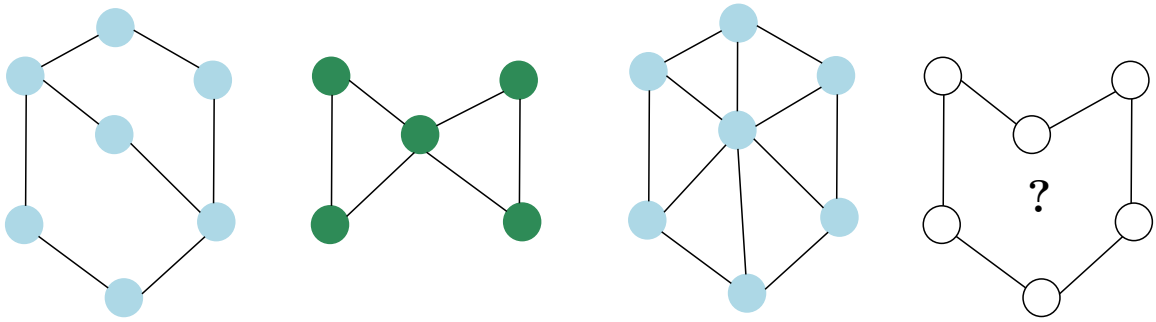


Figure 2.3: Graph Classification Example.

2.1.4 Subgraph Classification

Subgraph Classification is a new downstream task in graph representation learning. In general, all the downstream tasks vary in the number of nodes used in the prediction

task. As a result, nodes, links and graphs can be considered as specific instances of subgraphs with a single node, pair of nodes, isolated group of nodes, respectively. Therefore, subgraphs can be considered as an abstract data type for nodes, links and graphs and subgraph classification can be considered a more general problem. Existing works on node classification, link prediction and graph classification cannot be directly extended to subgraph classification as subgraphs have both internal as well as external topology. Subgraph classification solutions however can be generalized to work for other downstream tasks. Few works such as GLASS [75] and SubGNN [5] has focused on the subgraph classification task. Such works use labeling tricks or subgraph specific channels to model the properties important to subgraphs. However, the use of additional labels and message-passing channels creates an overhead and simple models are necessary to enable learning on large-scale graphs. An example for subgraph classification is given in Figure 2.4. Each target subgraph in the graph has a label (shown by blue and green color) and the task is to predict the label of the unlabeled subgraph.

2.2 Shallow Encoders

One of the earliest works to create node embeddings were through the use of shallow encoders such as DeepWalk [60] and node2vec [29]. DeepWalk [60] learns latent node representations through the use of random walk sequences. DeepWalk uses the random walk sequences to construct the node embeddings similar to using words to construct sentences in natural language processing. The random walk sequences allows the model to learn the local structure around a node. To obtain node embeddings, node2vec [29], an extension of DeepWalk, uses random walk sequences controlled by two different hyperparameters. The two hyperparameters p and q provides a trade-off

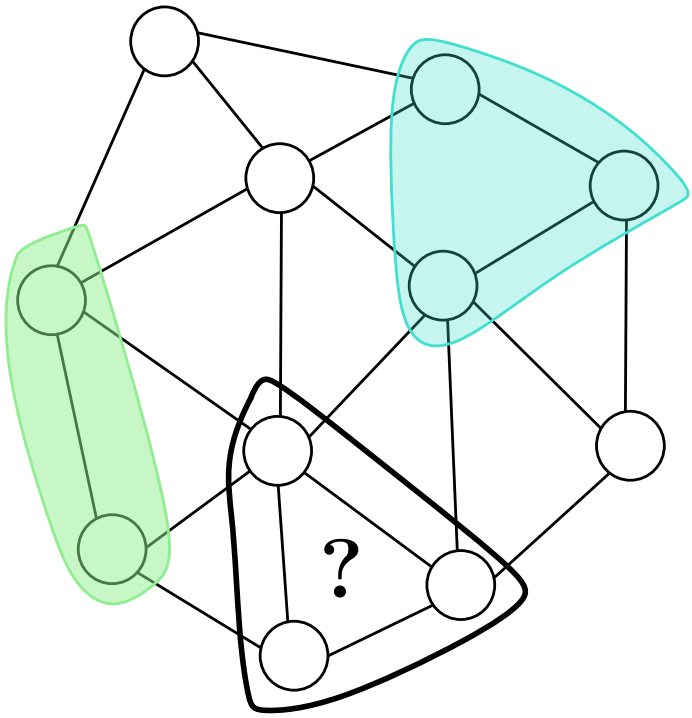


Figure 2.4: Subgraph Classification Example.

between local (i.e., breadth-first) and global (i.e., depth-first) neighborhood exploration. More specifically, p and q controls the return probability and inward-outward exploration, respectively. Due to the reliance of node2vec on structure, it is predominantly used as a pretraining method for graphs without any features. However, the lack of use of features makes it inapplicable to inductive settings.

2.3 Message Passing Graph Neural Networks

Message Passing Graph Neural Networks (MPGNNs) converts hidden representations of nodes to learnt embeddings through the use of multiple message-passing iterations. Each message-passing iteration is specified using two functions: AGGREGATE and UPDATE. The AGGREGATE function aggregates the neighborhood information of each node whereas the UPDATE function updates the representation of each node

by combining its previous hidden representation and the aggregated neighborhood information. Each round of message-passing allows the nodes to contain information from nodes that are 1-hop away. After h rounds of message-passing, each node has information from nodes that are upto h-hops away. The final representation after h rounds of message-passing can then be used in any downstream task.

In the coming subsections, we discuss two prominent message-passing graph neural network architectures, GCN and GraphSAGE followed by a discussion on how MPGNNs can be applied to the subgraph classification task.

2.3.1 Graph Convolutional Networks

Graph Convolutional Network (GCN) [42] is one of the first message-passing graph neural network (MPGNN) that uses messages from neighboring nodes to update each node's representation. The message aggregation and update step in GCN can be summarized as:

$$\mathbf{h}_u^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{v \in N^+(u)} \mathbf{h}_v^{(l-1)}\right), \tag{2.1}$$

where $\mathbf{h}_u^{(l)}$ is the hidden representation of node u at layer l, $\mathbf{W}^{(l)}$ is a learnable weight matrix, and $N^+(u)$ contains the neighbors of u and itself. σ represents an activation function such as ReLU. In GCN, the AGGREGATE and UPDATE function is combined to a single step. Node representations are obtained through summation over the messages from the neighboring nodes and the node itself. By stacking h layers of convolution, we can learn the h-hop neighborhood of each node. Since GCNs use the entire neighborhood to make node feature updates (i.e., aggregation), it can become costly especially when dealing with nodes with large neighborhoods. In addition to

this, GCN cannot be applied to inductive settings thereby, limiting its application. However, GCN can be used in transductive settings for various downstream tasks such as node classification and link prediction.

2.3.2 GraphSAGE

To allow for inductive inference and computation speedup, GraphSAGE [31] samples the neighborhood of each node when making feature updates. The formulation of GraphSAGE differs from GCN in three ways. The UPDATE function in GraphSAGE is concatenation of the node and its neighborhood representations rather than combining it to one representation. GraphSAGE allows for different AGGREGATE function for the neighboring node messages such as mean or LSTM. Furthermore, the neighborhood of each node is sampled. The message aggregation and update step in GraphSAGE can therefore be summarized as:

$$\mathbf{h}_{N^k(u)}^{(l)} = AGGREGATE(\{\mathbf{h}_v^{(l-1)}, v \in N^k(u)\}), \tag{2.2}$$

$$\mathbf{h}_u^{(l)} = \sigma\left(\mathbf{W}^{(l)}.CONCAT(\mathbf{h}_u^{(l-1)}, \mathbf{h}_{N^k(u)}^{(l)})\right), \tag{2.3}$$

where $\mathbf{h}_{N^k(u)}^{(l)}$ is the messages from the sampled neighborhood $N^k(u)$ of node u at layer l, k is the number of nodes sampled from the neighborhood at each layer, $\mathbf{h}_u^{(l)}$ is the hidden representation of node u at layer l and $\mathbf{W}^{(l)}$ is a learnable weight matrix. The AGGREGATE function of GraphSAGE, given by Equation 2.2, aggregates the sampled neighborhood of each node through simple mean aggregation or using an LSTM. The UPDATE function of GraphSAGE given by Equation 2.3 concatenates the representation of each node with its aggregated neighborhood. Using sampled neighbor-

hood in AGGREGATE reduces the computational overhead, especially when dealing with nodes that have large neighborhoods. Furthermore, GraphSAGE addresses the shortcoming of GCN by allowing inference on unseen nodes or graphs. As a result, GraphSAGE finds application in large-scale graphs and inductive settings.

2.3.3 Subgraph Classification by MPGNN

An l-layered GNN can be adapted for the subgraph classification task by applying a subgraph pooling function on the node representations. If $\mathbf{X}$ represents the initial node features of the graph G, then the output of the l-layer GNN can be represented as:

$$\mathbf{Z} = f_{GNN}(G, \mathbf{X}) \tag{2.4}$$

The subgraph representation for S denoted by $\mathbf{q}_s$ can then be obtained using:

$$\mathbf{q}_s = pool(\mathbf{Z}, G, S) \tag{2.5}$$

where *pool* is a permutation invariant function such as sum pooling, max pooling or mean pooling. The final subgraph class can be obtained by passing $\mathbf{q}_s$ to a classifier.

2.4 SubGNN

SubGNN [5] is one of the first works in subgraph classification. SubGNN is a graph neural network model that takes into account the subgraph-level properties with respect to neighborhood, structure, and position when creating node level representations which is missing in traditional GNNs. For this purpose, separate message

passing channels are used for modeling the different structural properties that are key to subgraphs. SubGNN uses property-specific anchor patches to achieve the same. Figure 2.5 shows the SubGNN architecture.

Anchor patches are subgraphs sampled from the base graph that help in understanding how each target subgraph is located within the base graph. Each anchor patch is created to account for the properties of a subgraph such as position, neighborhood and structure within the subgraph and between the subgraph and the external nodes, respectively. For internal and border position, a node from the subgraph and a group of nodes from the base graph are used as anchor patches. For internal and border neighborhood, nodes from the subgraph and nodes from the h-hop subgraph neighborhood are used as anchor patches. For internal and border structure, triangular random walks are used to sample anchor patches.

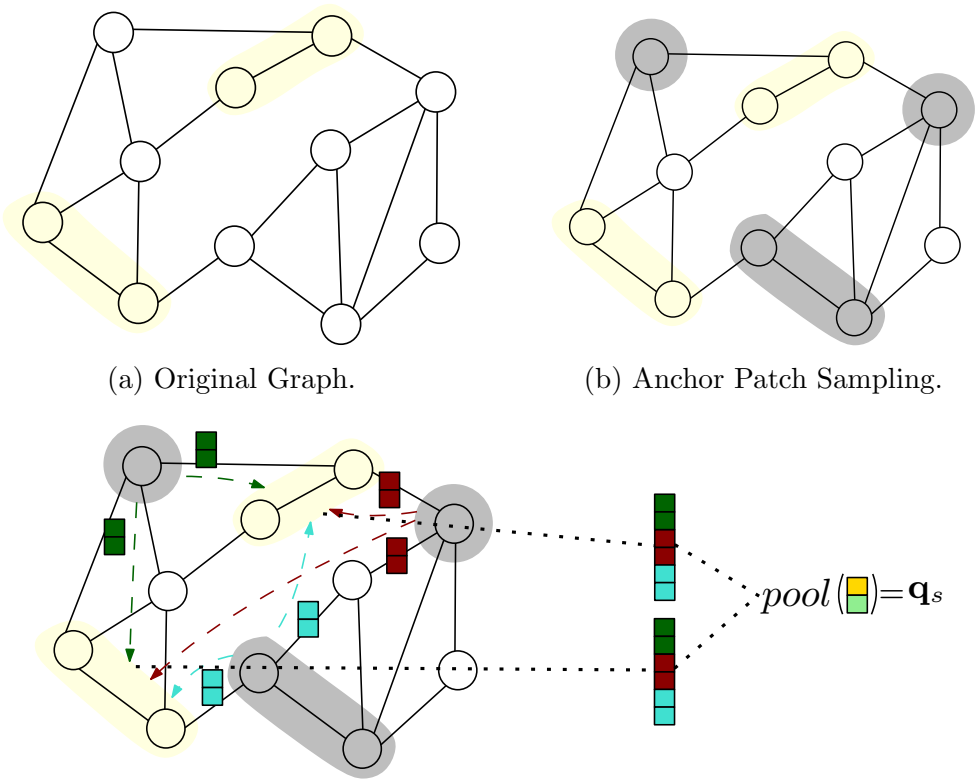
The property-specific anchor patches $\mathcal{A}$ are then used to propagate messages to each of the component in the subgraph as shown in Equation 2.6.

$$\mathbf{Z}_P = AGG_M(G, \mathbf{X}, \mathcal{A}, S) \tag{2.6}$$

The messages are propagated through a similarity function based on shortest path distance between anchor patches and the subgraph components. Once the property-aware subgraph component embeddings are available, a channel aggregation function AGG_C is used to obtain component representations for position, structure and neighborhood as shown in Equation 2.7.

$$\mathbf{Z}_C = AGG_C(\mathbf{Z}_P, S) \tag{2.7}$$

Another aggregation function AGG_S is used to combine the subgraph component



(a) Original Graph.

(b) Anchor Patch Sampling.

(c) Message Passing followed by Pooling.

Figure 2.5: SubGNN architecture. (a) The task is to learn a representation for a subgraph (shaded in yellow) in the original graph. (b) SubGNN samples property-specific anchor patches (shaded in grey) from the base graph. (c) SubGNN propagates messages from the anchor patches to the subgraph components followed by multiple aggregations to obtain the subgraph component representations. *pool* combines the different component representations to obtain the subgraph representation $\mathbf{q}_s$.

representations as shown in Equation 2.8.

$$\mathbf{Z}_S = AGG_S(\mathbf{Z}_C, S) \tag{2.8}$$

Similarly, an aggregation function AGG_L is used to aggregate the outputs from different layers. Finally, a *pool* function is used to obtain the subgraph embedding $\mathbf{q}_s$ as shown below.

$$\mathbf{q}_s = pool(\mathbf{Z_S}). \tag{2.9}$$

The final representation is then fed into a classifier. The use of specific anchor patches and message passing channels ensure that the subgraph embeddings contain the necessary topological information.

However, the separate message passing channels (up to six) and the sampling of anchor patches creates a computation overhead which can increase as the (sub)graph size increases. Moreover, as the graph size increases, there might be a need to increase the anchor patch sizes in order to capture the topology of the graph.

2.5 GLASS

Plain-GNNs fail to work on subgraph classification tasks due to their inability to distinguish internal and external nodes of a subgraph during message passing which is crucial to learn different subgraph properties. To address this, GLASS [75], current state-of-the-art model for subgraph classification, differentiate nodes within and outside the subgraph with the use of labeling tricks. GLASS extends SEAL [94] and overcomes its computational bottleneck by applying labeling to the entire graph without the need for subgraph extractions. GLASS uses a simple zero-one labeling trick [97] on nodes with a full GNN. Zero-one labeling trick [97] assigns all the target nodes a label 1 and the remaining nodes in the sub(graph) a label 0.

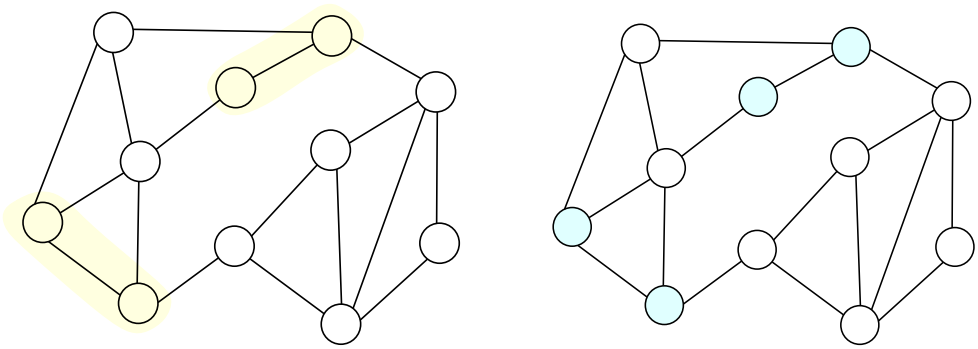
The zero-one labeling trick allows GLASS to encode the six topological properties of a subgraph such as internal position, border position, internal neighborhood, border neighborhood, internal structure and border structure. Furthermore, GLASS uses a node-level message passing framework in comparison to SubGNN which uses subgraph-level message passing to learn these properties.

To enable mini-batching of multiple subgraphs, the labels of the different subgraphs in the mini-batch are combined to create a new label. To avoid labeling

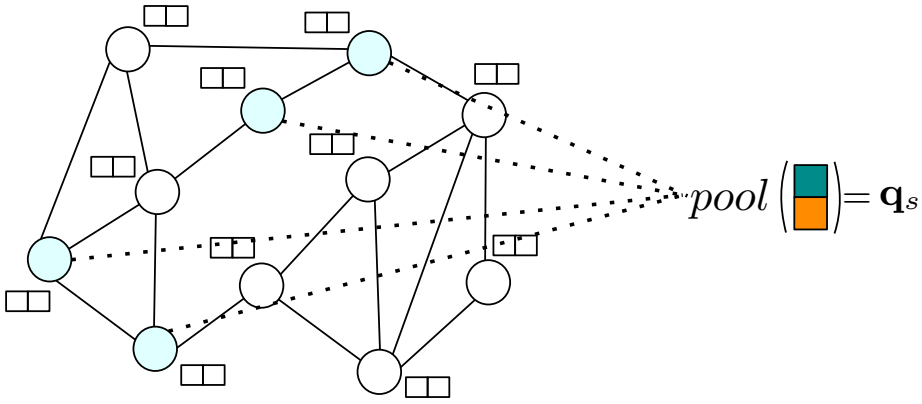
inconsistencies when different subgraphs are in a mini-batch, a novel max zero-one labeling scheme is introduced. The max zero-one labeling takes the maximum of the different labels created for each node by all the subgraphs in the mini-batch. The max zero-one labeling is an approximation of the zero-one labeling trick. However, max zero-one is as effective as zero-one labeling if the subgraphs in the mini-batch are located far away from each other. If the subgraphs are located in different parts of the base graph, it reduces the interference of labels from multiple subgraphs. Furthermore, if the subgraphs in the mini-batches have significant overlap, GLASS reduces to a plain GNN. Using this simple but powerful max zero-one labeling enhances the performance of any graph neural network and also beats the previous state-of-the-art SubGNN [5] both in terms of accuracy and speed. Although previous works using zero-one and other labeling tricks [97] exist, they focus on learning subgraph representations for the downstream task of link prediction [47, 94]. As such, a direct application of labeling tricks to subgraph classification did not exist before the introduction of GLASS. While max zero-one labeling trick removes the need for subgraph extractions, the labeling is dependent on the number of subgraphs in each mini-batch and the graph needs to be relabeled whenever a mini-batch is created.

The architecture of GLASS is summarized in Figure 2.6. The task is to learn a representation for a subgraph (shaded in yellow) in the original graph (see Figure 2.6a). GLASS labels the nodes within the subgraph with the label 1 (indicated by blue nodes in Figure 2.6b) and all other nodes with the label 0. The labeled graph then undergoes message passing followed by pooling of subgraph nodes using a permutation invariant function (i.e., *pool*) (as seen in Figure 2.6c) to obtain the subgraph representation $\mathbf{q}_s$. The final subgraph representation is then passed through a classifier to get the class probabilities.

GLASS follows a Network in Network architecture [69] for message passing. The



(a) Original Graph. (b) Node Labeling.



(c) Message Passing followed by Pooling.

Figure 2.6: GLASS framework.

node labels for each mini-batch is created using a labeling function given by:

$$L_m = f_L(G, \mathbb{S}_m) \tag{2.10}$$

where f_L denotes the max zero-one labeling function, G denotes the original graph, $\mathbb{S}_m$ denotes the set of subgraphs in the mini-batch m and L_m denotes the final node labels of the mini-batch m. GLASS uses the labels L_m during the message passing as shown in Equation 2.11 to learn the node representations for nodes with labels 0 and 1 in the mini-batch using separate layers.

$$\mathbf{Z_m} = f_T(G, \mathbf{X}, L_m) \tag{2.11}$$

where f_T is a transformation function with message passing layers and $\mathbf{Z_m}$ is the learnt representation of all nodes in the mini-batch m. The outputs of these layers are then mixed to obtain the final node embeddings. The usage of separate layers enable the model to learn different parameters for nodes with different labels without the need to explicitly augment the labels to the node features. Furthermore, such an approach using two separate functions and mixing works better in preventing the smoothing of labels in the graph. Additionally, GLASS uses normalization layers such as GraphNorm [12] for generalization and faster convergence. The final subgraph representation $\mathbf{q}_s$ for $S \in \mathbb{S}_m$ is given by:

$$\mathbf{q}_s = pool(\mathbf{Z_m}, S). \tag{2.12}$$

2.6 Data Augmentation in Graphs

Data augmentation increases the number of labeled samples in the training data by creating multiple copies of the existing samples/data. Data augmentation is particularly helpful when there is only limited data available to train a model. Data augmentation techniques can also increase the generalizability of the model and avoid overfitting by allowing the model to learn more data points. Data augmentation in graph data [98] is achieved by using techniques such as node dropping and edge perturbation. Similar techniques can be used to create augmented subgraphs. GraphCL [90] analyses the impact of different data augmentation techniques in various domains. GAUG [99] proposes new data augmentation technique of adding missing edges and removing noisy edges from the graph for better performance on node classification. Other works [101] focus on node and edge-level data augmentation strategies that adapt to the underlying graph structure. While data augmentation can increase the

number of training instances for more effective training, finding a suitable data augmentation approach can be challenging and requires careful analysis of the underlying dataset.

2.7 Graph Contrastive Learning

Graph Contrastive Learning is a self-supervised learning technique that jointly learns on multiple data points. Graph contrastive learning (GCL) uses specific loss functions that allow similar samples to be closer in the representation space and dissimilar samples to be far away in the representation space. GCL often uses data augmentation techniques to create positive pairs for the contrastive learning process. GraphCL [90], GAUG [99], JOAO [89], AD-GCL [71], InfoGCL [81] are recent graph contrastive learning frameworks that uses augmented samples to create positive and negative pairs for contrastive learning. Each of the contrastive learning method differs in the underlying data augmentation strategy used and the contrastive loss term formulation.

Chapter 3

Related Work

In this chapter, we discuss related work relevant to our line of research. We first review recent work on GNNs that have been successful in other downstream tasks, and then discuss prominent works in subgraph classification. We then discuss scalability of SGRLs followed by scalability through various sampling techniques. In each of the section, we provide explanation as to how each of these works are used in our model.

3.1 Shallow Embedding Methods

The early work in graph representation learning was *shallow encoders* such as DeepWalk [60] and node2vec [29]. DeepWalk uses random walks to understand the neighborhood around each node and encodes the sequence of random walks as node representations. An extension of Deepwalk, node2vec [29] introduced two hyperparameters to control the trade-off between breadth-first and depth-first exploration for random walks. The two hyperparameters offers flexibility to the model in learning each node's neighborhood thereby allowing the model to learn more expressive representations. However, as shallow embedding approaches focus on the relative positioning of the

nodes, they do not use any node features (e.g., user characteristics in a social network).

Relation to This Thesis. Our proposed *SSNP* can use shallow embedding methods such as DeepWalk and node2vec to create node embeddings used in the pooling function. Furthermore, the hyperparameters such as number of random walks and random walk length used to create random-walk induced subgraph neighborhoods can be considered similar to node2vec parameters, p and q that offers trade-off between local and global structure.

3.2 Message Passing Graph Neural Networks

Due to the inapplicability of shallow embedding methods in inductive settings and their negligence of nodal features, message-passing graph neural networks (MPGNNs) [10,22] were introduced and popularized. GCN [42], one of the most popular MPGNN models, iteratively updates nodal representations by aggregating messages (through summation) from its neighbors. However, GCN suffers from the exploding neighborhood problem with a high number of node feature updates. Moreover, GCN is inherently transductive. To overcome this, GraphSAGE [31] used a neighborhood sampling method during message passing that allowed for the model to be inductive as well as scalable. Furthermore, GraphSAGE allows for different aggregation functions for its neighbors such as mean and LSTM. Additionally, the updated node representation is a concatenation of the node and its neighborhood representation. GAT [73] is another MPGNN that improves the message passing scheme by computing different weights or attention for node feature aggregation in a neighborhood. GIN [82] enhances the expressiveness of graph neural networks by applying a multi-layer perceptron to the nodes after message passing. JK-net [83] enhances the representation

power of GNNs by concatenating the intermediate layer representations with the final output representation.

Recently, Graph Neural Networks that uses the concept of anchor distances [47, 57, 88] have been proposed to make the underlying model position-aware. The use of anchor distances helps in understanding how the nodes are positioned with respect to each other and their structure in the base graph. P-GNN [88] randomly selects anchor node sets from the base graph and uses distances from each node to the anchor sets for weighted aggregation of the messages. DLGNN [47] uses a GNN architecture to get the representation of a pair of nodes through concatenation of their individual representations along with a distance information vector that contains distances to sampled anchor nodes from the graph.

Graph InfoClust (GIC) [54] improves the quality of node representations created by a graph neural network using mutual information maximization of the node representation and graph-level as well as cluster-level summaries. This allows the node representations to be similar with nodes that belong to the same cluster and thereby getting better representations. N-GCN [1] uses a network of GCN modules where each module uses a different power of the adjacency matrix, thereby learning different statistics from the varying graph scales. MixHop [2] extends N-GCN by allowing for neighborhood mixing at each layer and learning differences in features with different graph scales.

Relation to This Thesis. Our model can use any of the MPGNNs as a transformation layer to generate the node embeddings used for generating the subgraph and its neighborhood embeddings. By using *SSNP*, the expressive power of the subgraph representations created by the underlying MPGNN can be improved.

3.3 Subgraph Representation Learning

Despite their success on the node and graph classification tasks, MPGNNs fail to uniquely capture pairwise nodal interactions [13, 74]. To circumvent this, SEAL [94] converts link prediction to a graph classification problem (see Figure 3.1) by extracting enclosing subgraphs around each pair of target nodes, which are then used to predict the existence/absence of the link. Using enclosing subgraphs in SEAL allows the model to learn/approximate various heuristics in the network. To understand the structure of the enclosing subgraph, SEAL injects distance information as nodal features of the subgraph nodes using double-radius node labeling (based on distances of nodes in a subgraph to the target nodes). Labeling tricks (e.g., double-radius) are shown to allow the underlying model to learn the dependency between the target nodes in their neighborhood subgraphs [97]. Subgraph representation learning approaches (SGRLs), by using the enclosing subgraphs around the target pair and labeling tricks, have enhanced the expressive power of MPGNNs for link prediction [11, 48, 58, 94]. Such methods works at a subgraph-level (i.e., operates on groups of nodes) rather than at a node-level as seen in plain GNNs. DE-GNN is an extension of SEAL [94] and uses Distance Encoding (DE) as additional node features in the learning paradigm. Distance Encoding helps GNN to understand the overall structure of the graph as well as the position of nodes. DE-GNN can be applied to a variety of tasks such as node classification, link prediction and triangle prediction. Shortest path distance and landing probabilities are used as distance encoding methods. mLink [11] extends SEAL [94] by transforming each enclosing subgraph to multiple scales and using each of this subgraph for link prediction. WalkPool [58] uses *walk profiles* as additional information for each node in the enclosing subgraph. The random-walk profiles capture the reachability of different nodes in the subgraph

and allow for encoding of higher order structural properties. The success of SGRLS even has extended to other downstream tasks. For example, shaDow-GNN [92] extract K-hop subgraphs around each node and operate on them for node classification. Similarly, NGNN [96] aggregates the node features in the K-hop rooted subgraphs around each node to increase the expressiveness of representations for graph classification. Recently, I^2-GNNs [37] extended NGNN by using both labeling tricks and subgraph-level information to improve graph classification and cycle counting in graphs. I^2-GNNs does this by labeling the root node and one of its neighbors in the K-hop subgraph before message passing, thereby increasing the nodal representational power of MPGNNs.

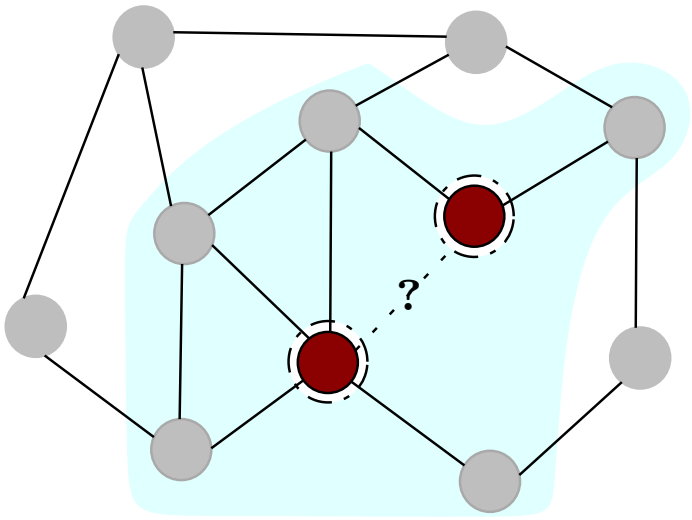


Figure 3.1: An example of SGRL framework SEAL. An enclosing subgraph is extracted for each pair of nodes, thereby converting the problem to graph classification from link prediction.

Relation to This Thesis *SSNP* builds upon the idea that enclosing subgraphs around a pair of nodes can provide more information and result in better representations. By including the subgraph neighborhood information at the pooling layer, we can increase the expressiveness of the underlying model without being computationally expensive.

3.4 Subgraph Classification

Subgraph classification [5, 75] is an emerging problem, which extends subgraph representation learning. SubGNN [5] samples anchor patches from the base graph and propagates messages between anchors to the subgraph in multiple channels to learn the internal and external topologies of subgraphs. The anchor patches are sampled to encode properties such as neighborhood, position, and structure of a subgraph in the base graph. While SubGNN learns the different topological properties of the subgraphs, sampling of channel-specific anchor patches followed by propagation is computationally expensive. The state-of-the-art GLASS [75] uses the zero-one labeling trick [97] to differentiate between the internal and external nodes of a subgraph and thereby encode various topological properties of the subgraph. GLASS further modifies zero-one labeling to max zero-one labeling to enable mini-batch training. Sub2Vec [3], as a subgraph embedding model, is deployed for community detection and graph classification; however, it can be adapted for the subgraph classification task. Sub2Vec samples random walks from a node within each subgraph to learn its structure and neighborhood. This information is then fed into Paragraph2Vec [43] to create the final subgraph embeddings. Some recent work on subgraph classification includes PADEL [49] and Subgraph-To-Node (S2N) [40]. PADEL uses data augmentation and contrastive learning techniques along with position encodings of nodes during message passing. The position encodings are obtained using cosine phase position encoding. These position encodings are then used in the pooling function along with node features to learn the structure. Subgraph-To-Node (S2N) [40] framework removes the complexity and overhead associated with extracting different subgraph properties in SubGNN [5] by using a subgraph-to-node translation. S2N translates the subgraphs to nodes to coarsen the base graph, and casts the subgraph classifica-

tion task to a node classification task. S2N has the advantage of using plain GNNs to perform subgraph classification compared to SubGNN. Furthermore, the translation functions used in the framework are also flexible. Other works use a multi-view augmentation framework [67] that can be beneficial for subgraph tasks. By using the original subgraph and multiple augmented views, different subgraph representation vectors can be created for the same original subgraph. Multiple views of the original subgraph are created using strategies such as node dropping and edge dropping. Each view including the original subgraph is then passed to a subgraph specific model to get their embeddings. The individual embeddings of each view of the subgraphs are then pooled to get one final subgraph embedding. The authors use GLASS [75] for the subgraph embedding task and feeds the final subgraph embedding to a MLP to perform subgraph classification. [63] adds invariant linear layers between GLASS layers to improve their performance.

Relation to This Thesis. We include majority of these works as our baselines to measure the performance and scalability of our model. We consider GLASS [75] as our best baseline in terms of performance. We do not include PADEL [49] and S2N [40] as our baselines as these methods do not have results on all the datasets available.

3.5 Scalability of SGRLs

SGRLs are computationally demanding due to the extraction of subgraphs around target nodes, applying labeling tricks, and running GNNs on each subgraph. To address this computational bottleneck, a recent line of research has emerged in scalable SGRLs. SaGNN [76] enhances the expressiveness of a graph neural network by aggregating node representations in the rooted subgraph around each node in the base

graph to make the model subgraph-aware. SaGNN does not use the subgraphs for message passing but only at the aggregation step. ScaLed [50] extends SEAL by sparsifying the enclosing subgraphs around the target nodes (see Figure 3.2) to reduce the computational overhead associated with large subgraph sizes. The random-walk-induced subgraphs approximate the enclosing subgraphs without substantial performance compromises. Two hyperparameters are used to control the number and length of random walks to create the induced subgraph. By setting the random-walk hyperparameters to an appropriate value allows for comparable performance with substantial computational gains. SUREL [85], similar to ScaLed, uses pre-computed random walks around each pair of nodes to approximate the subgraph, however, does not use MPGNNs. Furthermore, SUREL does not use random-walk induced subgraphs rather only the random walk sequences. Additionally, relative positions of the nodes that occur in the random walks are recorded using vectors called relative position encoding (RPE). To provide maximum overlap and reusability between query nodes in the training set, query nodes are sampled using BFS to create mini-batches. ELPH/BUDDY [13] uses computationally-light algorithms such as MinHashing [9] and HyperLogLog [25, 34] to derive subgraph sketches for approximating the neighborhood overlap and unions around target nodes for faster message-passing without explicit subgraph extractions. S3GRL [51] models speed up the training and inference of SGRL methods by simplifying the underlying GNN message-passing and aggregation steps. S3GRL does this by removing the non-linearity in-between graph convolutions, thus allowing precomputation of the subgraph-level message passing, and consequently faster training and inference. Additionally, S3GRL uses multiple subgraph operators in conjunction to learn the different properties of the enclosed subgraphs.

Relation to This Thesis. We use the findings of works such as SaGNN [76],

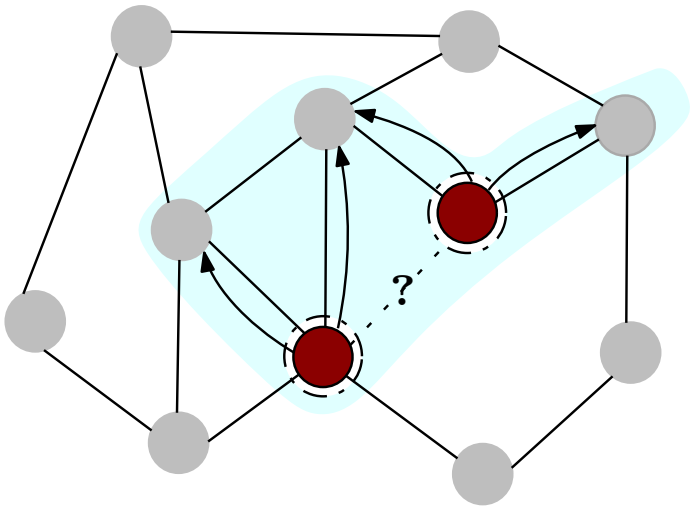


Figure 3.2: An example of scalable SGRL framework ScaLed. ScaLed samples the enclosing subgraphs using multiple random walks (specified by two hyperparameters) starting from the target pair of nodes.

ScaLed [50] and SUREL [85] to use random-walk induced subgraph neighborhoods in our $\text{pool}_{\text{SSNP}}$. Additionally, our model with *SSNP* is an alternative to expensive SGRL based methods while being more expressive than a plain GNN and subgraph-aware.

3.6 Scalability by Sampling

A number of graph neural network methods cannot be applied to large-scale graphs due to huge resource requirements (due to neighborhood explosion problem). As a result, many GNNs were proposed over the years that provide scalability through various sampling strategies such as node-wise sampling [15,31], graph sampling [19,93] and layer-wise sampling [14,36,103]. Such sampling techniques (shown in Figure 3.3) provide scalability through the reduced memory requirements as well as faster training times.

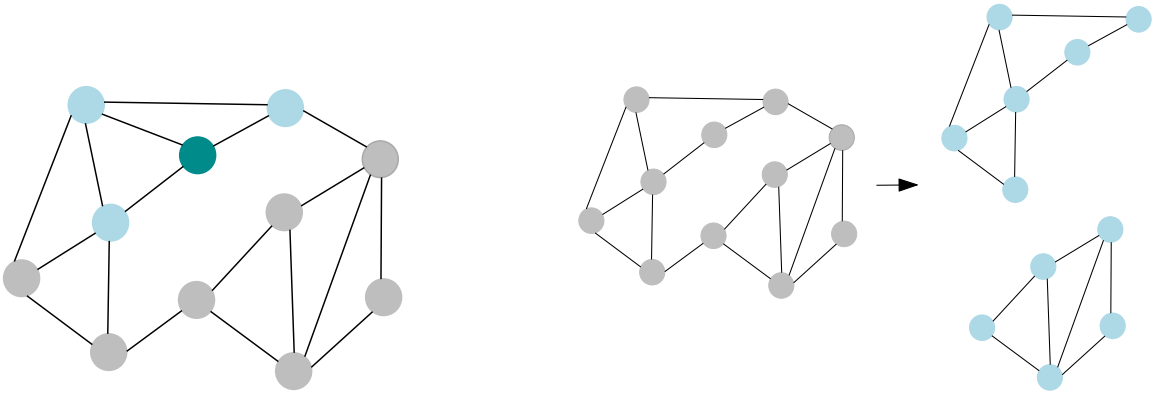
In node-wise sampling, the nodes in the base graph are sampled. GraphSAGE [31]

provides scalability by sampling nodes during the neighborhood aggregation process allowing for faster training time while being inductive. VR-GCN [15] performs sampling of neighborhood nodes with the goal of reducing variance during sampling. VR-GCN uses the historical node embeddings to compute the control variate and decide the nodes to be sampled.

To further alleviate the neighborhood explosion problem, layer-wise sampling is used. In layer-wise sampling, the nodes are sampled for each convolution layer in the graph neural network. FastGCN [14] is an extension of the traditional graph convolution network where layer wise sampling of the vertices in the graph is performed to allow for inductive learning and reduced computation complexity. While GraphSAGE performs neighbor sampling at each layer, FastGCN performs layer independent sampling of the nodes itself, which reduces the total complexity of the model considerably. An importance sampling is used to decide the training nodes. However, since it is a layer independent sampling there exists the issue of having disconnected nodes in the training graph. LADIES [103] overcomes the problem of disconnected nodes between layers by proposing a layer-dependent node sampling technique. ASGCN [36] is another layer-wise sampling technique where the node sampling in each layer is dependent on the next layers.

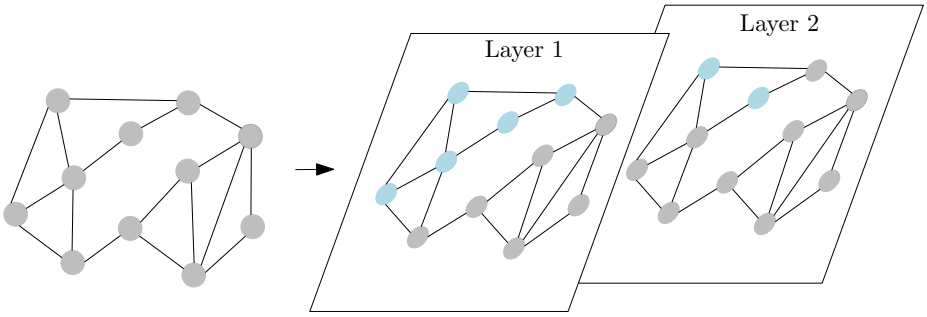
In graph sampling, subgraphs from the base graph are sampled for model learning and inference. Cluster-GCN [19] proposes an efficient training algorithm that can work with deeper GCNs without higher memory requirements. Since each node is most influenced by its neighbors within the same cluster, it is important to preserve the links within a cluster more than the links between different clusters. Cluster-GCN uses this idea to create mini batches. Ensuring that all nodes in a cluster belong to the same mini-batch improves the memory consumption. Different clusters are used in each mini-batch to ensure a better learning for the model. In addition

to this, the clusters in mini-batches vary in each epoch to allow for generalization. GraphSAINT [93] is another graph-wise sampling technique that samples subgraphs from the original graph for training. GraphSAINT proposes three different subgraph sampling strategies: node, edge and random walk sampling.



(a) Node-wise sampling.

(b) Graph-wise sampling.



(c) Layer-wise sampling.

Figure 3.3: Different sampling techniques used in Graph Neural Networks for scalability. The sampled node set is shown in blue.

Relation to This Thesis. Our random-walk induced subgraph can be considered a node sampling technique that allows for faster training and filters noisy neighborhood nodes. In addition to this, *SSNP* can work with graph neural networks that use any of the above mentioned sampling techniques for further scalability.

Chapter 4

Approach

In this chapter, we discuss the required preliminaries followed by the problem statement for subgraph classification. We then introduce Stochastic Subgraph Neighborhood Pooling and its various components such as transformation layers, subgraph neighborhood pooling and subgraph neighborhood sampling strategies.

4.1 Preliminaries

Let $G = (V, E)$ represent a simple, undirected graph where $V = \{1, \ldots, n\}$ is the set of nodes (e.g., users, scientists, articles, proteins, etc.), and $E \subseteq V \times V$ represents the edge set (e.g., friendships, collaborations, citations, interactions, etc.). We sometimes represent G by the adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ where $a_{ij} = 1$ if an edge exists between nodes i and j, and 0 otherwise. We also assume each node $i \in V$ possesses a d-dimensional feature $\mathbf{x}_i \in \mathbb{R}^d$ (e.g., user information, research profile, keywords, protein characteristics). We sometimes stack all nodal features, row-by-row in the feature matrix $\mathbf{X}$ whose i-th row contains $\mathbf{x}_i$. We consider a subgraph $S = (V_S, E_S)$ in base graph G where $V_S \subseteq V$ and $E_S \subseteq (V_S \times V_S) \cap E$.

4.2 Problem Statement

The goal is to learn a mapping function $f(G, \mathbf{X}, S)$ which takes the base graph G, its node feature matrix $\mathbf{X}$, and a subgraph S as an input, and outputs the subgraph class label $y \in \{1, \ldots, C\}$, where C is the number of classes. The class labels of the subgraph could represent the toxic friendship communities, cellular functions (e.g., metabolism, development, etc), or metabolic/neurological disorders.

4.3 Stochastic Subgraph Neighborhood Pooling (*SSNP*)

We first discuss the various components of our proposed solution for subgraph classification. We then detail an important part of this solution, our proposed Stochastic Subgraph Neighborhood Pooling (*SSNP*).

Our proposed solution for subgraph classification is depicted in Figure 4.1. The initial node features $\mathbf{X}$ are transformed to learned embeddings $\mathbf{Z}$ through the use of a transformation function f_T:

$$\mathbf{Z} = f_T(G, \mathbf{X}) \tag{4.1}$$

The transformation function f_T can be multi-layers of graph convolutions (with message passing) for feature smoothing or a simple multi-layer perceptron (without any explicit message passing) for dimensionality reduction. We have considered three different types of transformation layers: Nested Network convolution [69], GCN convolution [42], and Multi-Layer Perceptron (MLP). Nested Network convolution and GCN convolution are message-passing layers whereas MLP is a graph-agnostic transformation method (see more details in Section 4.3.1). After obtaining node embeddings $\mathbf{Z}$, our proposed $\text{pool}_{\text{SSNP}}$ function is used to aggregate the target subgraph's internal

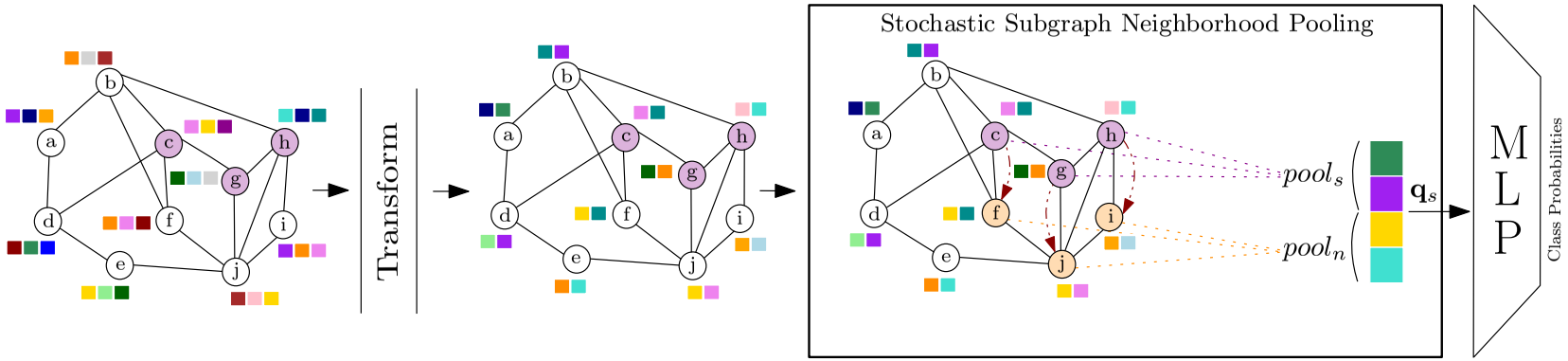


Figure 4.1: Architecture of our model. Subgraph nodes are shaded in purple. The initial node features are transformed using transformation layers such as Nested Network convolutions, GCN convolutions, or MLP. The stochastic subgraph neighborhood pooling $\text{pool}_{\text{SSNP}}$ is applied in multiple steps. The subgraph neighborhood nodes (shaded in brown) are sampled by rooted random walks (red dashed arrows). The subgraph and its sampled neighborhood are separately pooled by $pool_s$ and $pool_n$, which are simple graph pooling operators (e.g., mean, sum, etc.). The pooling outputs are concatenated to form the subgraph representation $\mathbf{q}_s$, which is passed to an MLP for generating class probabilities.

and external topological properties into a latent subgraph representation:

$$\mathbf{q}_s = \text{pool}_{\text{SSNP}}(\mathbf{Z}, G, S) \tag{4.2}$$

This subgraph representation $\mathbf{q}_s$ is fed to an MLP to output class probabilities for the subgraph classification task. The MLP, in addition to giving the class probabilities, learns how to mix the pooled subgraph and its neighborhood representations. Our proposed solution does not require computationally-expensive labeling tricks (as opposed to GLASS [75]), or artificially-crafted message passing channels (as opposed to SubGNN [5]). This computational reduction is achieved by applying transformation on the base graph (rather than on subgraphs) and our proposed SSNP function. Detailed information on transformation layers and our proposed $\text{pool}_{\text{SSNP}}$ function follows.

4.3.1 Transformation Layer

In addition to deploying MLP as a transformation function, we have considered two graph convolution layers. We discuss their formulations in this section.

Nested Network Convolution. Our Nested Network (NN) convolution follows a Network in Network architecture [69] as a way of deepening a GNN model by adding multiple non-linear layers within a convolution layer to increase model capacity while preventing overfitting and oversmoothing. The first step of the NN convolution layer is to transform the current layer's node embeddings $\mathbf{h}_u^{(l-1)}$ using one linear layer with an activation function σ:

$$\hat{\mathbf{h}}_u^{(1)} = \sigma\left(\mathbf{W}_1^{(l)}\mathbf{h}_u^{(l-1)}\right) \tag{4.3}$$

Following this, we perform simple message passing with summation aggregation followed by graph normalization and dropout:

$$\hat{\mathbf{h}}_u^{(2)} = f_{GD}\left(\sum_{v \in N^+(u)} \hat{\mathbf{h}}_v^{(1)}\right) \tag{4.4}$$

where $N^+(u)$ contains the neighbors of u and the node u itself and f_{GD} is a sequential function of graph normalization followed by dropout. The recently updated representation $\hat{\mathbf{h}}_u^{(2)}$ is then concatenated with the original layer's input representation $\mathbf{h}_u^{(l-1)}$ (similar to residual connections [33, 84]) to be linearly transformed to the output representation of the layer:

$$\mathbf{h}_u^{(l)} = \mathbf{W}_2^{(l)}\left(\hat{\mathbf{h}}_u^{(2)} \oplus \mathbf{h}_u^{(l-1)}\right) \tag{4.5}$$

Equations 4.3, 4.4 and 4.5 constitute a single layer of convolution in our NN model with two learnable weight matrices $\mathbf{W}_1^{(l)}$ and $\mathbf{W}_2^{(l)}$.

GCN Convolution: Our implemented GCN convolution layers exactly follows GCN [42]. Neighborhood features are aggregated through message-passing by

$$\mathbf{h}_u^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{v \in N^+(u)} \mathbf{h}_v^{(l-1)}\right), \tag{4.6}$$

where $\mathbf{W}^{(l)}$ is a learnable weight matrix, and $N^+(u)$ contains the neighbors of u and itself.

4.3.2 Subgraph Neighborhood Pooling and Variants

Our proposed pooling is built based on the idea that the representations of subgraphs and their neighborhoods are both important for capturing the internal and external topology of subgraphs. We first define the h-hop subgraph neighborhood as:

Definition 1 (h-hop Subgraph Neighborhood) *Given the base graph $G = (V, E)$ and its subgraph $S = (V_S, E_S)$, the h-hop subgraph neighborhood $N_S^{(h)}$ is the induced subgraph created from the node set $\{j \in V_N | min_{i \in S} d(i, j) \leq h\}$, where $d(i, j)$ is the geodesic distance between node i and j, and $V_N = V \setminus V_S$ are nodes of G that do not belong to S.*

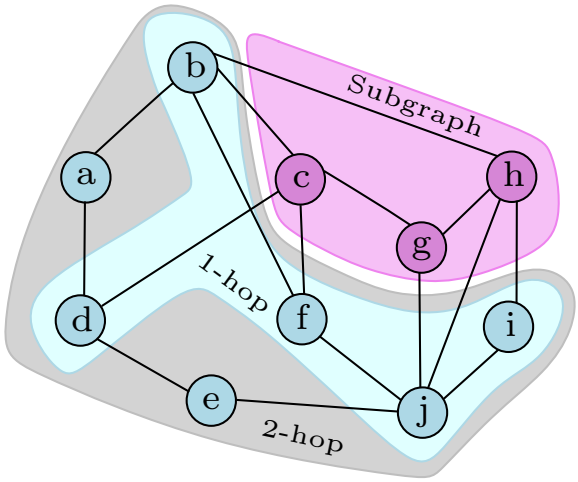


Figure 4.2: h-hop subgraph neighborhood.

In simple words, the h-hop subgraph neighborhood is the subgraph of G whose nodes do not belong to S and are within a distance of h to at least one of the nodes of S. An example of 1-hop and 2-hop subgraph neighborhood is shown in Figure 4.2. Our h-hop subgraph neighborhood can be viewed as an extension (or generalization) of the enclosing subgraphs for pair of nodes [94] but with two distinctions: (i) the h-hop neighborhood is defined for any subgraph size (rather than just a pair of nodes)

and (ii) the subgraph S is excluded from its neighborhood subgraph. Given this h-hop subgraph neighborhood definition, we first consider a simple *subgraph neighborhood pooling*:

$$\text{pool}_{\text{SNP}}(\mathbf{Z}, G, S, h) = \text{pool}_s(\mathbf{Z}_S, S) \oplus \text{pool}_n\left(\mathbf{Z}_N, N_S^{(h)}\right), \tag{4.7}$$

where $\mathbf{Z}_S$ and $\mathbf{Z}_N$ denote the matrix node embeddings of the subgraph S and its neighborhood $N_S^{(h)}$. Here, $\oplus$ is the concatenation operator, and $pool_s$ and $pool_n$ can be any order invariant graph pooling function (e.g., sum, mean, max, size, or SortPooling [95]). The main idea here is simple: treat the subgraph and its neighborhood as two separate graphs, then pool their information, and then concatenate their representations to capture both the internal and external topology of the subgraph. Current subgraph representation learning models (e.g., GLASS, SubGNN) only use $pool_s$, while ignoring the rich information of the neighborhood subgraph.

However, consuming the complete subgraph neighborhoods is computationally problematic as the subgraph neighborhoods can become extremely large and dense with many uninformative and noisy nodes, thus hindering the model's learning capability and slowing down the running time. To overcome this limitation, we define h-hop sparsified subgraph neighborhood:

Definition 2 (h-hop Sparsified Subgraph Neighborhood) *Given the base graph $G = (V, E)$ and subgraph $S = (V_S, E_S)$, we define the h-hop sparsified subgraph neighborhood $\hat{N}_S^{(h,k)}$, as the subgraph induced from the nodes in $\hat{V}_S^{(h,k)} \in \{W_S^{(h,k)} \setminus V_S\}$, where $W_S^{(h,k)}$ is the set of nodes visited by k many h-length random-walk(s) from the nodes in V_S.*

Compared to the exact subgraph neighborhood which can get extremely large, the size of the sparse subgraph neighborhoods is bounded by hk, which is the product of the length and number of random walks. The rooted random walks allow sampling

"important" external nodes to a subgraph (similar to rooted PageRank [8]), which encapsulates information on the border structure and neighborhood. The randomness in the neighborhood subgraph also adds some regularization effect to the training of the model (similar to what was observed in ScaLed [50]). Our h-hop sparsified subgraph neighborhood has a resemblance with random-walk sampled enclosing subgraphs [50], but differs in two ways: the neighborhood does not include the original subgraph, and the neighborhood is defined over arbitrary-sized subgraphs (rather than pair of nodes). Given the computational and learning advantages of specified neighborhood subgraphs, we introduce *stochastic subgraph neighborhood pooling (SSNP)* by a slight modification of Eq. 4.7:

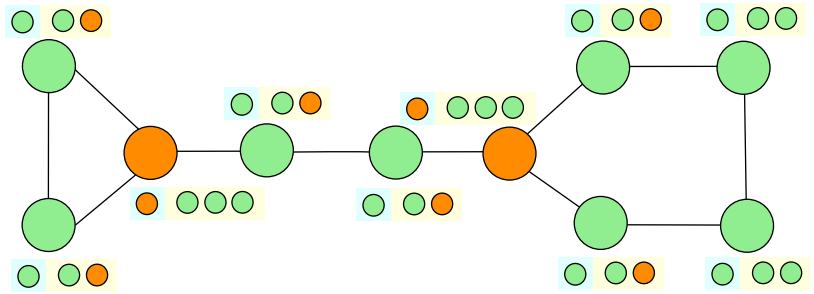
$$\text{pool}_{\text{SSNP}}(\mathbf{Z}, G, S, h, k) = \text{pool}_s(\mathbf{Z}_S, S) \oplus \text{pool}_n\left(\mathbf{Z}_N, \hat{N}_S^{(h,k)}\right), \tag{4.8}$$

where $\mathbf{Z}_S$ and $\mathbf{Z}_N$ denote the matrix node embeddings of the subgraph S and its sparsified neighborhood $\hat{N}_S^{(h,k)}$ by k-many h-length random walks.

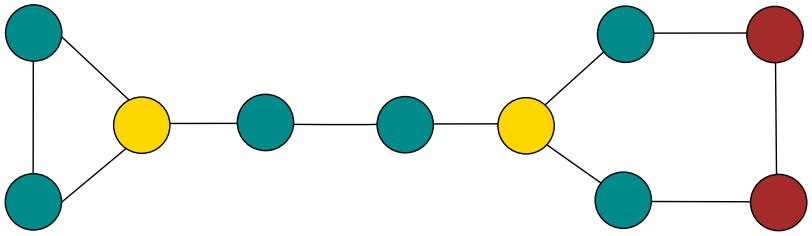
4.4 Expressiveness of Subgraph Neighborhood Pooling

Weisfeiler-Lehman (WL) algorithm [78] has been used for graph isomorphism testing (i.e., to test whether two graphs are isomorphic) and thereby, evaluate the expressive power of various models. In the WL algorithm, each node in the graph is given an initial color. In each iteration of coloring, a node and its neighboring nodes' colors are aggregated to form a multi-set. This multi-set is updated to a new color using a hash function. After multiple rounds of coloring, if two graphs have different node colors, then the two graphs are non-isomorphic. Graph Neural Networks such as GCN [42]

and GraphSAGE [31] use a similar neighborhood aggregation mechanism to obtain node embeddings (which is equivalent to node colors) and therefore, are as powerful as 1-WL algorithms (that aggregate neighborhood node colors within 1-hop of each node). An example of 1-WL coloring in GCN is shown in Figure 4.3.



(a) 1-WL color initialization and neighborhood multi-sets.

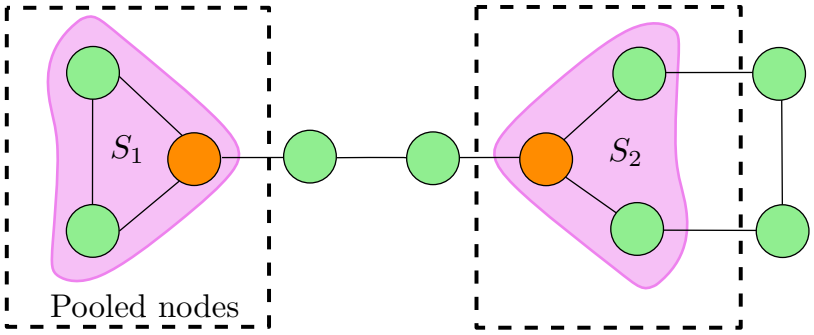


(b) Node colors after 1 iteration

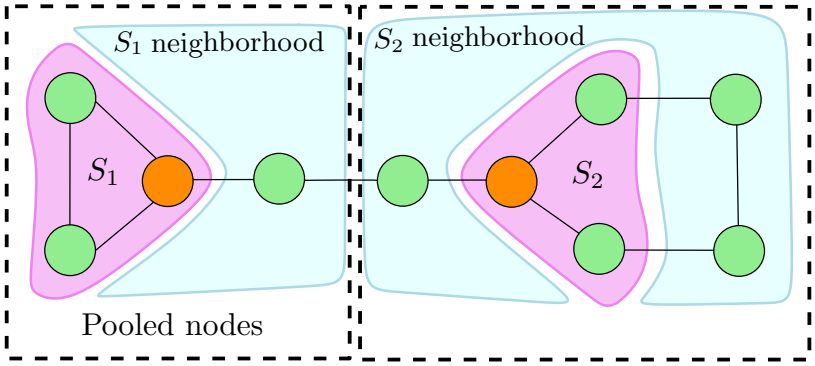
Figure 4.3: 1-WL coloring in a plain-GNN such as GCN.

A plain GNN (which only pools subgraph embeddings without its neighbors) is also as powerful as 1-WL algorithm. However, our model with $\text{pool}_{\text{SSNP}}$ is more expressive than a plain GNN in the absence of distinguishing node features. Figure 4.4 shows an example of two subgraphs that are distinguishable under our model, but not under the plain GNN. After one iteration of 1-WL coloring/MPGNN followed by subgraph pooling (shown by pink shaded area) in Figure 4.4a, S_1 and S_2 has the same representation (the nodes involved in the pooling step are in dotted boxes). However, after one iteration of 1-WL followed by subgraph neighborhood pooling including both subgraph pooling and neighborhood pooling shown by pink and blue shaded area, respectively in Figure 4.4b, S_1 and S_2 have different representations. As shown in

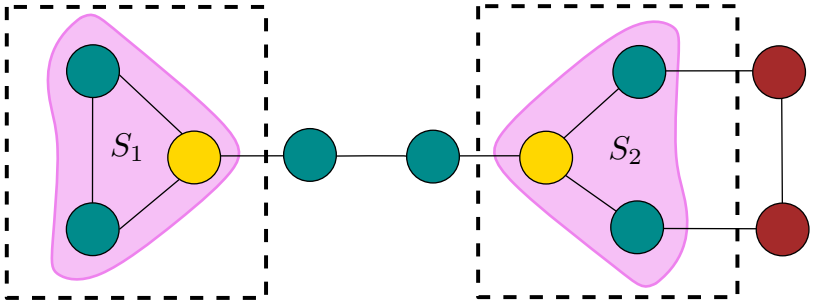
Figure 4.4c, after two 1-WL iterations followed by subgraph pooling, S_1 and S_2 still are not distinguishable. Similar to Figure 4.4b, in Figure 4.4d, after two 1-WL iterations followed by subgraph neighborhood pooling, S_1 and S_2 have different representations. This additional expressiveness is just an outcome of simple low-cost neighborhood pooling. Our model with *SSNP* as a result is more powerful in distinguishing non-isomorphic subgraphs with unique neighborhoods in comparison to plain GNNs. In the worst case (i.e., in the absence of unique subgraph neighborhoods), our model ignores the subgraph neighborhood information and degrades to a plain GNN and is thereby at least as powerful as 1-WL algorithm.



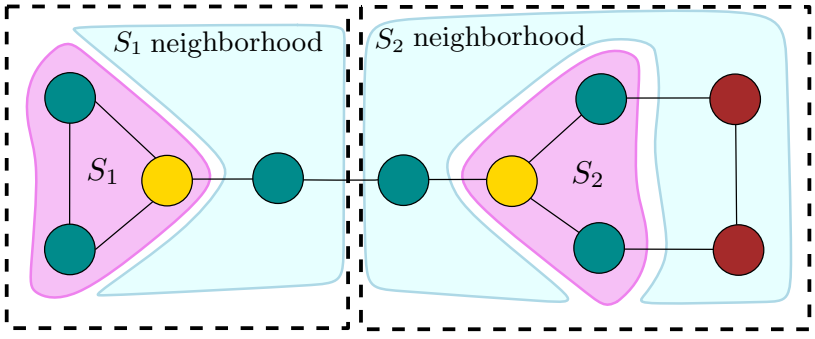
(a) Subgraph Pooling, Iter. 1



(b) Subgraph Neighborhood Pooling, Iter. 1



(c) Subgraph Pooling, Iter. 2



(d) Subgraph Neighborhood Pooling, Iter. 2

Figure 4.4: Comparison of subgraph pooling vs subgraph neighborhood pooling for MPGNNs on distinguishing two non-isomorphic subgraphs S_1 and S_2 without any distinguishing node features.

4.4.1 Subgraph Neighborhood Sampling Strategies

Random walks are effective in approximating and sparsifying subgraphs around a node [50, 85]. However, the sampling of the sparsified subgraph neighborhood in each training epoch might introduce undesirable instability and stochasticity in gradient computations and optimization procedures. To account for this instability as well as manage the sampling overhead, we introduce and distinguish three different stochastic subgraph neighborhood sampling strategies.

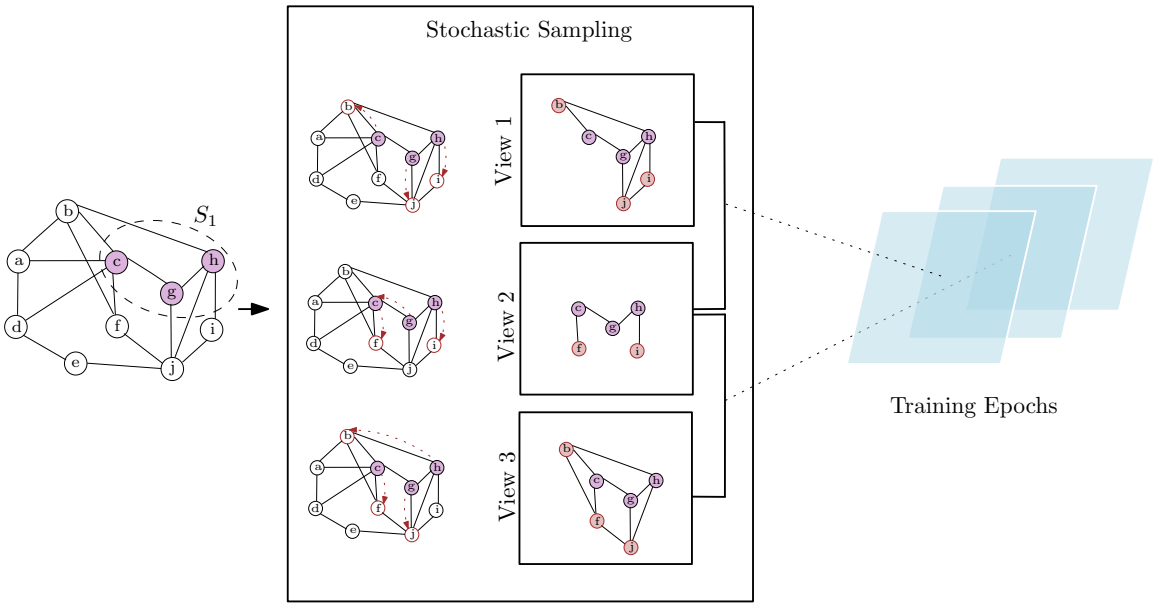


Figure 4.5: An example of POV when n_v=3 and n_{ve}=2.

Online Stochastic Views (OV): The h-hop sparsified subgraph neighborhood is sampled in each epoch. This stochasticity over training intends to add implicit regularization to the model but might have undesirable outcomes of gradient instability. Also, the epoch-level sampling adds computational overhead to the training. This computational overhead is due to sampling potentially redundant sparsified subgraph neighborhoods as many times as the total number of epochs.

Pre-processed Stochastic Views (PV): To overcome the additional overhead cre-

ated by sampling during training, we propose *pre-processed stochastic views (PV)* for which a fixed number n_v of sparsified subgraph neighborhood is sampled for each subgraph before training (i.e., during preprocessing). These sampled neighborhood subgraphs can be viewed as data augmentation that provides n_v views of the subgraph neighborhood. Similar to other data augmentation strategies, PV improves the generalization of our model and makes it more robust to noise and overfitting.[1]. However, the dataset size and training time grows linearly with the number of views n_v.

Pre-processed Online Stochastic Views (POV): To reduce the training time on the augmented datasets, we propose *pre-processed online stochastic views (POV)* that leverages both the pre-processed and online subgraph neighborhood sampling method. In the pre-processing stage similar to PV, POV creates n_v multiple sparsified subgraph neighborhoods (i.e., multiple views) for each subgraph. But, during each epoch of training, for each subgraph only n_{ve} of the precomputed views are randomly sampled for training. POV allows data augmentation with multiple views while keeping the number of training instances per epoch independent of the number of views n_v. To do so, we have introduced the number of views per epoch n_{ve}.[2] Figure 4.5 shows an example of POV when there is a total of 3 views for each neighborhood and each epoch uses 2 views for training.

[1]The impact of multi-view augmentations on subgraphs has also been studied recently [49, 67]. However, our augmentation techniques create multiple views of the subgraph neighborhoods rather than subgraphs.

[2]Unlike contrastive learning methods, our model does not jointly learn from the different subgraph neighborhood views. As a result, our model is much faster than contrastive learning models.

Chapter 5

Experiments

We compare our model with *SSNP* and its variants against different subgraph classification baselines on four real-world datasets to evaluate our model in terms of performance and scalability. Our experiments are created to answer the following questions: **(Q1)** How does our *SSNP* model compare in terms of F1-score with respect to the baselines ? **(Q2)** How does our *SSNP* model compare to GLASS in terms of running time ? **(Q3)** How does the different hyperparameters in *SSNP* such as h (length of random walk), n_v (number of views) and n_{ve} (number of views per epoch) impact the model ? **(Q4)** How does the different variants of stochastic sampling strategies affect our model ?

5.1 Datasets

We perform experiments on four publicly-available real-world datasets that have been the main subject of study in other subgraph classification works [5, 75]. The dataset statistics are available in Table 5.1. In the ppi-bp dataset, the goal is to predict the cellular function of a group of genes, whereas, in hpo-metab we wish to predict

	ppi-bp	hpo-metab	hpo-neuro	em-user
Number of nodes	17080	14587	14587	57333
Number of edges	316951	3238174	3238174	4573417
Number of subgraphs	1591	2400	4000	324
Number of classes	6	6	10	2
Multi-label	No	No	Yes	No
Avg. number of nodes in subgraphs	10.2	14.4	14.8	155.4
Avg. density of subgraphs	0.216	0.757	0.767	0.010

Table 5.1: Statistics of all real-world datasets.

the metabolic disease corresponding to a group of phenotypes. The classification task in hpo-neuro is to predict the neurological disease corresponding to a group of phenotypes. In em-user, we wish to predict the gender of the user given the workout history subgraph. We follow the same dataset split as GLASS [75]: 80/10/10 for train, validation, and test splits.

5.2 Evaluation Metric

We use F1-score as the measure of efficacy of all models, where F1-score is measured as:

$$\textit{F1-score} = \frac{2 \times (\textit{precision} \times \textit{recall})}{(\textit{precision} + \textit{recall})}. \tag{5.1}$$

Precision is defined as

$$\textit{Precision} = \frac{TP}{TP + FP}. \tag{5.2}$$

Recall is defined as

$$\textit{Recall} = \frac{TP}{TP + FN}. \tag{5.3}$$

Figure 5.1 shows the confusion matrix of predicted vs true labels. True Positive (TP) indicates the number of samples correctly predicted as positive. False Positive (FP) indicates the number of samples falsely predicted as positive. False Negative

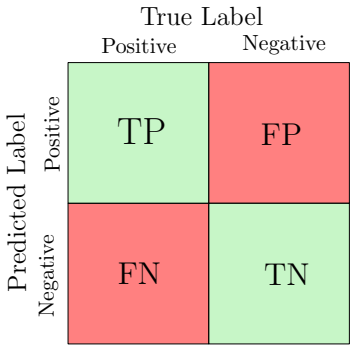


Figure 5.1: Confusion Matrix.

(FN) indicates the number of samples falsely predicted as negative.

5.3 Baselines

We consider the GLASS model [75] as our state-of-the-art baseline. Other baselines include SubGNN [5], graph-agnostic MLP, and GBDT (gradient-boosted decision trees), GNN-plain, Sub2Vec [3], and GNN-seg (learning on segregated subgraphs) [75]. All the baseline results, except for GLASS, are taken from [75]. The GLASS model is rerun by us to capture the timing values and verify that our setup is identical to the setup of reported results.[1]

5.4 Experimental Setup

For GLASS, we use the best-performing reported hyperparameters to reproduce their results. GLASS uses the Network in Network Architecture [69] to obtain the subgraph embeddings. For our model, we set the transformation layers/functions to either MLP, Nested Network (NN) [69], or Graph Convolution Network (GCN) [42], and the corresponding models are called *SSNP*-MLP, *SSNP*-NN and *SSNP*-GCN,

[1]We rerun GLASS using the code available at https://github.com/Xi-yuanWang/GLASS

respectively. We use the ELU activation [20] for all transformation layers. We always set the number of walks per node $k = 1$, and let the pooling method for the subgraph and neighborhood be the same (i.e., $\text{pool}_s = \text{pool}_n$). Unless noted otherwise, we use the POV for creating subgraph neighborhood views, where we set the number of views $n_v = 20$ and the number of views per epoch $n_{ve} = 5$. The other hyperparameters are searched over validation datasets to maximize micro-F1 scores. The search spaces are $\text{pool}_s \in \{sum, size\}$, length of walks $h \in \{1, 5\}$, and the number of transformation layers $\in \{1, 2, 3\}$. Similar to GLASS, we set the learning rate for ppi-bp to 0.0005 and hpo-neuro to 0.002 whereas, for both hpo-metab and em-user, we set it to 0.001. Our model, similar to GLASS and SubGNN, uses pre-trained 64-dimensional nodal features as the initial features for all datasets. We use Adam optimizer [41] paired with ReduceLROnPlateau learning rate scheduler, which reduces the learning rate on plateauing validation dataset loss values. We set dropout [70] to 0.5 for all models. We use a single-layer MLP to output the class probabilities and always use the cross-entropy loss in our model. Our models with NN and GCN transformation layers are trained for a maximum of 300 epochs in each run with a warm-up of 50 epochs for ppi-bp, hpo-metab and hpo-neuro and warm-up of 10 epochs for em-user. We set patience to 50 epochs for hpo-metab and hpo-neuro and 20 for em-user. Our models with the MLP transformation layer are run for 100 epochs. Our model is implemented in PyTorch Geometric [24] and PyTorch [59].[2] Our results are reported with an average F1-score over 10 runs with different random seeds.

Model	ppi-bp	hpo-metab	hpo-neuro	em-user
MLP	0.445±0.003	0.386±0.011	0.404±0.006	0.524±0.019
GBDT	0.446±0.000	0.404±0.000	0.513±0.000	0.694±0.000
GNN-plain	0.613±0.009	0.597±0.012	0.668±0.007	0.847±0.021
Sub2Vec	0.388±0.001	0.472±0.010	0.618±0.003	0.779±0.013
GNN-seg	0.361±0.008	0.542±0.009	0.647±0.001	0.725±0.003
SubGNN	0.599±0.008	0.537±0.008	0.644±0.006	0.816±0.013
GLASS	0.618±0.006	0.598±0.014	0.675±0.007	0.884±0.008
SSNP-MLP	0.591±0.006	0.571±0.006	0.669±0.004	0.853±0.012
SSNP-GCN	0.607±0.005	0.553±0.011	0.667±0.003	0.843±0.014
SSNP-NN	0.636±0.007	0.587±0.010	0.682±0.004	0.888±0.005

Table 5.2: The mean micro-F1 scores (average of 10 runs) with standard error for all models. The top 3 models are indicated by First, Second, and Third.

5.5 Results: F1 Score and Runtime

To answer **Q1**, we perform experiments on all baselines and compare our *SSNP* model with different transformation layers. To answer **Q2**, we compare *SSNP* with GLASS in terms of pre-processing, training, inference and runtimes.

Table 5.2 shows the mean micro-F1 results for all datasets. On ppi-bp, hpo-neuro, and em-user, our *SSNP*-NN model outperforms all others with a gain of 0.018, 0.011, and 0.004, respectively. For hpo-metab, *SSNP*-NN ranks third with a small margin of 0.011 compared to GLASS ranked first. This relatively low performance could be attributed to the fact that subgraphs in hpo-metab are dense and therefore, do not need external topological information or the neighborhood information adds noise to the model. This is also supported by the good results for GNN-seg on hpo-metab, which only uses target subgraphs for message passing. Moreover, on dense subgraph datasets such as hpo-metab and hpo-neuro, GNN-plain does not have considerable boost in their performance in comparison to GNN-seg. Although,

[2]Our code is available at https://github.com/shweta-jacob/SSNP. We run our experiments on servers with 50 CPUs, 377GB RAM, and 11GB GPUs.

	ppi-bp			
Model	Preproc.	Training	Inference	Runtime
SSNP-NN	*8.94±0.54*	0.38±0.02	0.02±0.00	129.35±3.27
SSNP-GCN	8.89±0.71	*0.42±0.02*	*0.03±0.00*	*142.38±3.85*
SSNP-MLP	**8.79±0.63**	**0.06±0.02**	**0.00±0.00**	**16.00±0.94**
GLASS	3.93±0.10	0.78±0.02	0.05±0.00	207.99±24.76
Speedup	0.44/0.45	1.86/13	1.67/25	1.46/13

	hpo-metab			
Model	Preproc.	Training	Inference	Runtime
SSNP-NN	25.20±0.84	0.73±0.02	0.05±0.001	159.56±18.86
SSNP-GCN	*26.13±1.53*	*0.94±0.03*	*0.06±0.00*	*209.20±43.15*
SSNP-MLP	**24.81±0.75**	**0.10±0.02**	**0.00±0.00**	**35.00±1.72**
GLASS	15.99±0.88	2.15±0.03	0.13±0.00	239.48±33.22
Speedup	0.61/0.64	2.29/21.5	2.17/43.33	1.14/6.84

	hpo-neuro			
Model	Preproc.	Training	Inference	Runtime
SSNP-NN	*29.67±1.54*	1.27±0.03	0.05±0.00	202.28±26.01
SSNP-GCN	**28.14±0.81**	*1.58±0.05*	*0.06±0.00*	*344.14±44.14*
SSNP-MLP	28.37±1.13	**0.21±0.01**	**0.01±0.00**	**50.00±1.05**
GLASS	16.56±0.84	4.20±0.04	0.25±0.00	511.54±94.40
Speedup	0.56/0.59	2.66/20	4.17/25	1.49/10.23

	em-user			
Model	Preproc.	Training	Inference	Runtime
SSNP-NN	*27.93±1.41*	*3.00±0.04*	*0.08±0.00*	*156.81±32.10*
SSNP-GCN	27.62±0.91	1.61±0.04	*0.08±0.00*	108.30±18.62
SSNP-MLP	**27.52±1.54**	**0.16±0.01**	**0.00±0.00**	**44.00±1.71**
GLASS	25.11±1.61	4.93±0.04	0.56±0.00	212.28±23.51
Speedup	0.90/0.91	1.64/30.81	7/140	1.35/4.82

Table 5.3: Our model vs GLASS: dataset preparation time, training time per epoch, inference time per epoch in seconds and average runtime (mean over 10 runs). The min/max speedup is the ratio of time taken by GLASS to the time of the slowest-/fastest *SSNP* model (in italics/bold). The runtimes are rounded to two decimal places; but, the speedups are computed from actual runtimes.

SSNP-NN performs better than other baselines on hpo-neuro which has dense subgraphs similar to hpo-metab, the task in hpo-neuro (i.e., multi-label classification) is inherently different from the task in hpo-metab (i.e., multi-class classification). Moreover, hpo-neuro contains around twice the subgraphs as hpo-metab which could attribute to the better performance of our model on hpo-neuro. Surprisingly, both *SSNP*-NN and *SSNP*-GCN outperform SubGNN across all the datasets. Even, our simplest model *SSNP*-MLP (even without message passing) outperforms SubGNN in all datasets except for ppi-bp for which it has a comparable result. *SSNP*-MLP also appears to be relatively competitive by being ranked third in hpo-neuro and em-user. All these results indicate that our models with simple transformation layers but the expressive pooling function of *SSNP* can easily outperform more complicated and computationally intensive models.

Our results in Table 5.2 also provide strong evidence in demonstrating how effective neighborhood pooling (and information) is for subgraph classification. The key difference between GNN-plain and *SSNP*-NN is the pooling of neighborhood subgraphs in *SSNP*-NN as both use NN architecture. Similarly, *SSNP*-MLP surpasses MLP by a significant margin too.

The average of dataset preparation time, training time per epoch, inference time per epoch, and total runtime are captured in Table 5.3. Our models for all datasets require at most twice the preprocessing times of GLASS due to the sampling of multiple views of the neighborhood subgraphs.[3] However, in return, the training and inference times are 1.5-137× faster depending on the model variations and datasets. Our best-performing *SSNP*-NN has a training speedup of 1.5-3.3× (min. for em-user and max. for hpo-neuro) and an inference speedup of 2.5-7× (min. for ppi-bp and

[3]One can easily reduce the preparation time by tweaking the total number of views created for each subgraph.

max. for em-user). Notably, our *SSNP*-MLP is the fastest with maximum training and inference (resp.) speedups of 30× and 140× (resp.) in em-user. Cross-examining Tables 5.2 and 5.3, we can observe that *SSNP*-MLP vs. GLASS has a speedup of 13-140× (for both training and inference) with a small negative gain of 0.006–0.031 in F1-score. Similarly, we see a runtime speedup of 4.8-13× (min. for em-user and max. for ppi-bp) with *SSNP*-MLP. These results suggest that our simple models outperform all baselines or were comparable while being multiple magnitudes faster than the current state-of-the-art baselines.

5.6 Multi-view Hyperparameter Analyses

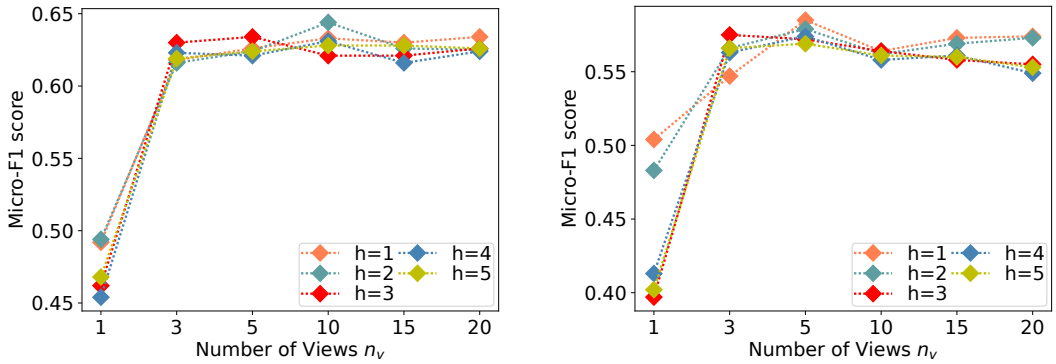
To answer **Q3**, we conduct a sensitivity analysis on the four datasets using the *SSNP*-NN model by varying the length of random walk h, total number of views n_v and number of views per epoch n_{ve}.

We first study the effect of the number of views n_v in the PV variant of our *SSNP*-NN model. For this analysis, we fix $k = 1$ and $h \in \{1, 2, 3, 4, 5\}$ for all datasets, while changing $n_v \in \{1, 3, 5, 10, 15, 20\}$. As shown in Figure 5.2, the F1 score for all datasets sharply increases from 1 to 3 for all values of h and then stabilizes. For hpo-metab, we observe a slight downgrade for a relatively large number of views (e.g., 15 or 20) when h is between 3 and 5 whereas the F1 score of em-user achieves its highest score on 20 views with h=5. For hpo-neuro, as the number of views increases from 10 to 15, for all values of h, we observe a decrease in the F1-score. These results suggest that the number of views should be a few (e.g., $n_v = 3$ or $n_v = 5$), but not so high (e.g, $n_v = 20$) to perform consistently over all the datasets.

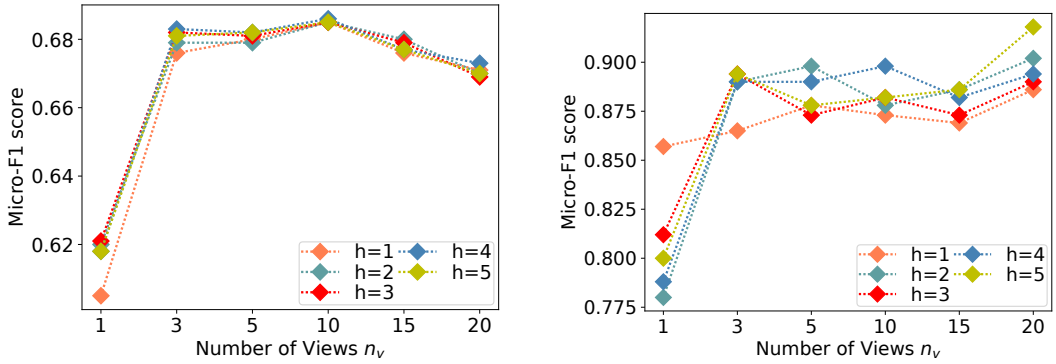
We further our analyses by studying the effect of the number of views per epoch n_{ve} in the POV variant for a fixed number of views $n_v = 10$. For this analysis, we fix

$k = 1$ and $h \in \{1, 2, 3, 4, 5\}$ for all datasets. We change $n_{ve} \in \{1, 2, 4, 6, 8, 10\}$ Figures 5.3a, 5.4a, 5.5a and 5.6a shows that the F1 score increases with n_{ve}, but it has a diminishing return pattern. However, an increase in n_{ve} directly increases the time taken for training (see Figure 5.3b, 5.4b, 5.5b, 5.6b) and thereby increases the total runtime. As it can be seen from Subplot 5.3a, when the number of views is 10 and h=1, 2, we get the largest F1 score. Moreover, setting h to 2 and 10 gives the best performance on ppi-bp. However, this combination does not give the best results for other datasets. In em-user, setting h to 3 and n_{ve} to 10 offers the best performance but incurs additional training overhead (see Figure 5.6b). In hpo-metab and hpo-neuro, setting h to 1 gives good results and setting h to 5 gives comparable or worse performance (see Figure 5.4a, 5.5a). Surprisingly, $n_{ve} = 4$ offers almost the same F1 score as what $n_{ve} = 10$ can offer, while requiring considerably less computation time. We believe this performance is primarily due to accessing large enough augmented training data and the regularization offered through the stochasticity of sampled views per epoch. In almost all the datasets, setting h to 1 gives better or comparable results. Cross-examination of the micro-F1 scores and training times suggest that setting n_{ve} to 2 or 4 with a small h value offers a good F1-score with manageable computational overhead.

In summary, *SSNP* with small random walk length and number of views per epoch can provide comparable or better results while maintaining a light computational load and allowing for faster training and inference.

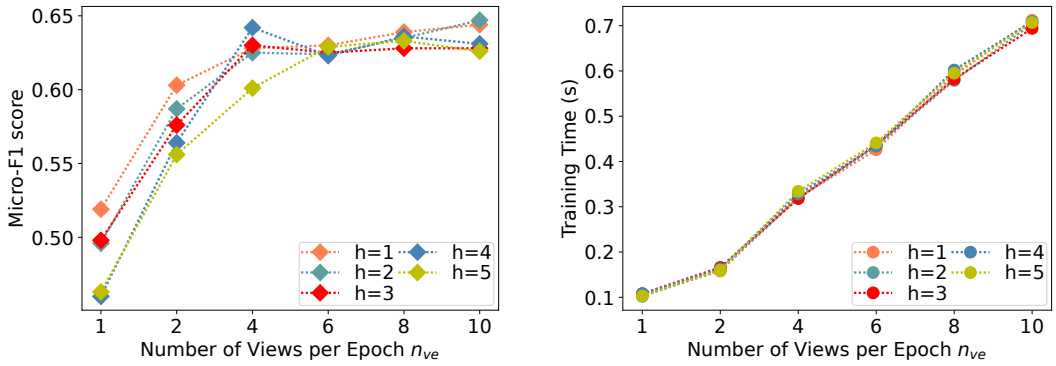


(a) ppi-bp (b) hpo-metab



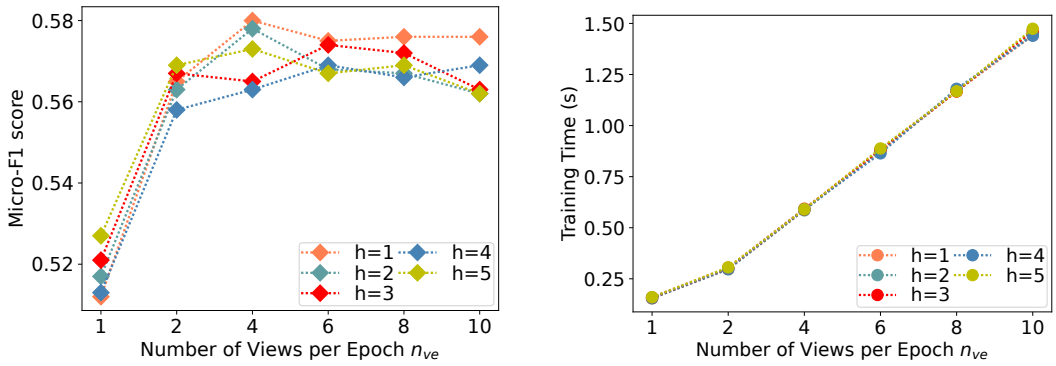
(c) hpo-neuro (d) em-user

Figure 5.2: The effect of number of views in our Preprocessed View variant of *SSNP*-NN.



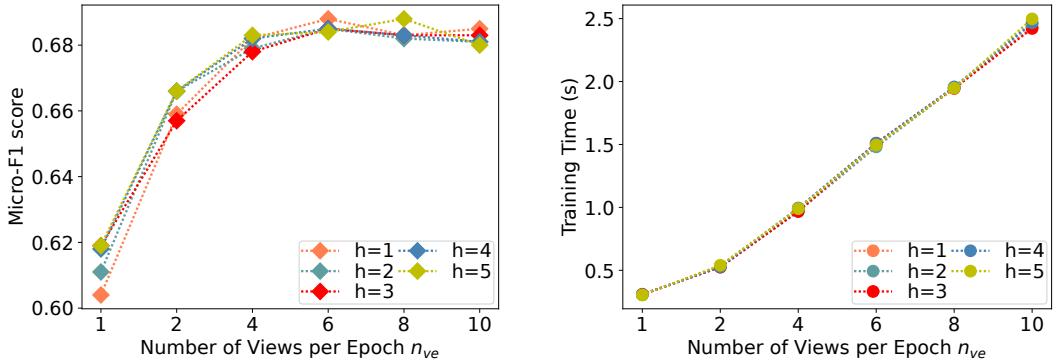
(a) (b)

Figure 5.3: The effect of number of views per epoch on our Preprocessed Online View variant of *SSNP*-NN on ppi-bp.



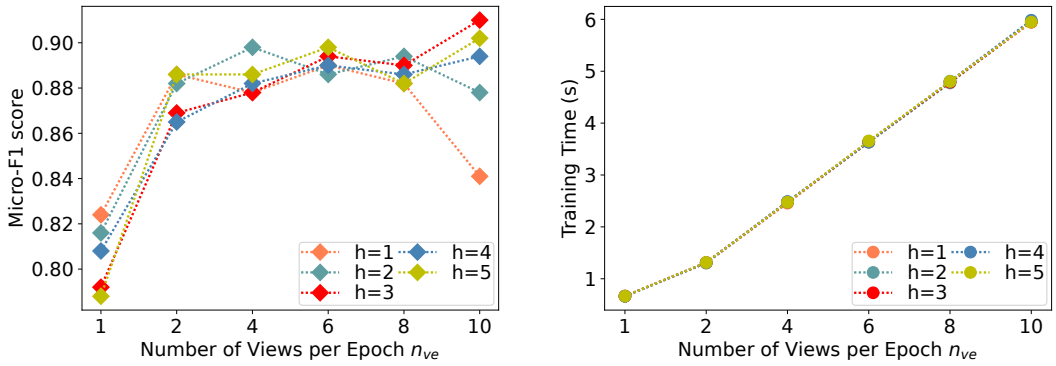
(a) (b)

Figure 5.4: The effect of number of views per epoch for our Preprocessed Online View variant of *SSNP*-NN on hpo-metab.



(a) (b)

Figure 5.5: The effect of number of views per epoch for our Preprocessed Online View variant of *SSNP*-NN on hpo-neuro.



(a) (b)

Figure 5.6: The effect of number of views per epoch for our Preprocessed Online View variant of *SSNP*-NN on em-user.

5.7 Results: Stochastic Pooling Strategies

To answer **Q4**, we intend to study the effect of various stochastic sampling strategies on our *SSNP*-NN model. We fix all hyperparameters as was reported above except those related to our pooling strategies. We set the number of views per epoch n_v to 1 for online views (OV), to 5 or 20 for pre-processed views (PV), and to 20 for pre-processed online views (POV). For POV, we also set the number of views per epoch n_{ve} to 5.

The micro-F1 scores are captured in Table 5.4. The effect of the sampling on the pre-processing and training times are captured in Figure 5.7. For all datasets (except em-user), POV provides the best F1-scores (see Table 5.4). For em-user, OV suppresses POV with a small margin of 0.004. In Figure 5.7, we can see that the average training time for OV in ppi-bp, hpo-metab and hpo-neuro is higher than PV with 5 views and POV. However, pre-processing of OV is faster than all other sampling strategies. For PVs and POV, the pre-processing times are comparable; however, POV offers much faster training time and a higher F1-score (see Table 5.4). In hpo-metab and hpo-neuro, the F1 score of PV with 5 views is higher than that of PV with 20 views, implying that a higher number of views does not necessarily improve performance for PV. However, POV, with 5 views per epoch and a total of 20 views, has the highest F1 score. This means that the stochasticity in the views across epochs allows a better generalization for our model.

Sampling Strategy	**ppi-bp**	**hpo-metab**	**hpo-neuro**	**em-user**
OV	0.527±0.008	0.443±0.055	0.681±0.002	**0.906±0.009**
PV (5 views)	0.628±0.007	0.569±0.015	0.680±0.003	0.878±0.015
PV (20 views)	0.635±0.003	0.553±0.013	0.671±0.003	0.902±0.007
POV	**0.638±0.008**	**0.577±0.017**	**0.686±0.004**	0.902±0.007

Table 5.4: F1-score (avg. over 5 runs) for various sampling strategies, *SSNP*-NN.

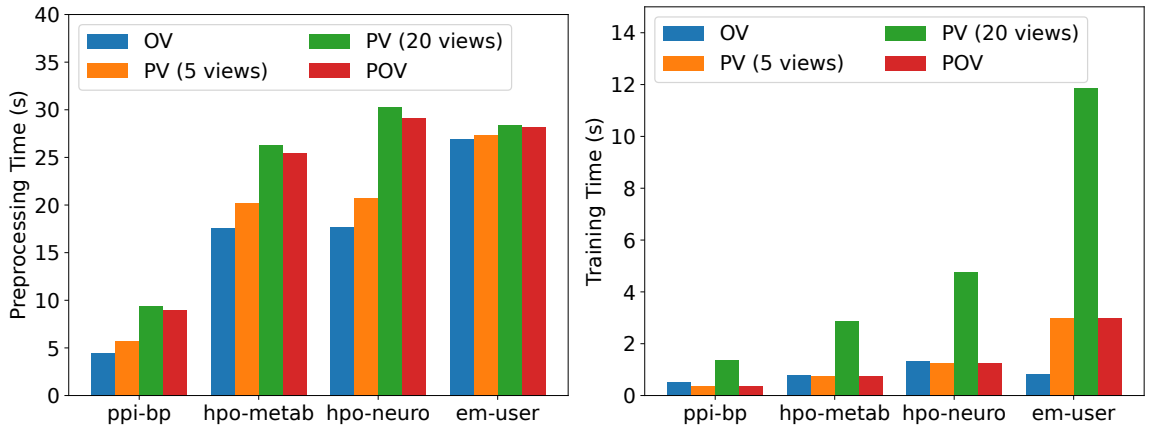


Figure 5.7: The effect of sampling strategies on pre-processing time (left) and training time per epoch (right) in *SSNP*-NN.

5.8 Summary

In this chapter, we showed the empirical results of *SSNP* in comparison to other baselines. Our experiments using *SSNP* with different transformation layers revealed that using simple transformation layer combined with subgraph neighborhood pooling gives good performance while being computationally light. Our *SSNP* with MLP outperforms SubGNN on all datasets, while being multiple magnitudes faster. Additionally, *SSNP*-MLP outperforms simple MLP which indicates that subgraph neighborhood information combined with our data augmentation technique is helpful for learning subgraphs. Furthermore, our multi-view hyperparameter sensitivity analysis on random walk length h, number of views n_v, number of views per epoch n_{ve} reveals that using small random walk length (1 or 2) combined with decent number of views per epoch (i.e, around 4) gives good results. Our ablation studies on various stochastic pooling strategies reveals that POV variant is superior to others. *SSNP* combined with POV beats *SSNP* with OV and PV on 3 out of the datasets. Furthermore, POV has similar pre-processing times to PV, while being much faster in training without

much loss in F1-scores. The simple data augmentation technique of POV allows for generalization of the model without incurring additional computational costs.

Chapter 6

Conclusions

In this chapter, we conclude our thesis with a brief summary of contributions. Then, we discuss a few limitations of our model and possible future research directions that can address these shortcomings.

6.1 Thesis Summary

The state-of-the-art subgraph classification solutions are not scalable due to the use of labeling tricks or artificial message-passing channels for subgraphs. In this thesis, we propose a simple yet powerful model that has our proposed stochastic subgraph neighborhood pooling (*SSNP*) in its core. Leveraging *SSNP*, our model learns the internal connectivity and border neighborhood of subgraphs. We also present simple data augmentation techniques such as OV, PV and POV that help to improve the generalization of our model. Our model combined with our data augmentation techniques outperforms current state-of-the-art subgraph classification models on 3 out of 4 datasets with a speedup of 1.5-3×. Our model with simple transformation layers such as MLP beats majority of the subgraph classification baselines while re-

quiring only node-level operations and being computationally efficient. Experiments on multi-view hyperparameter sensitivity analyses and stochastic pooling strategies reveal that using a small number of views combined with our subgraph neighborhood pooling strategy can improve the performance of the underlying model.

6.2 Future Directions

While *SSNP* can work with any transformation layer and is scalable, there are possible improvements to make the model more robust and scalable. Therefore, we present the current limitations of our model and ways to address them in the next subsections.

6.2.1 Reusing Random Walks

Our model currently performs random walks for each node multiple times even if it occurs in multiple subgraphs. This might introduce additional overhead, especially when the overlaps between subgraphs is large. Therefore, pre-processing random walks for all nodes, similar to SUREL [85], that are part of subgraphs might be essential to bring down the computation cost. The random-walk sequences of all nodes in each subgraph can then be combined to create the induced subgraph neighborhood.

6.2.2 Efficient Subgraph Neighborhood Selection

Random walks are efficient in approximating subgraph neighborhoods. However, the hyperparameters of random walk such as number of walks and length of walks might be dataset or even subgraph dependant. For future work, we plan to explore alternative ways to approximate neighborhood subgraphs using subgraph-aware neighborhood selection operators, as seen in other recent works [72].

6.2.3 Graph Contrastive Learning

The multiple views of subgraph neighborhoods created in our model are used to increase the training data. Another promising direction might be to perform contrastive learning on the different stochastic views of neighborhood subgraphs to allow for maximization of mutual information. However, while contrastive learning techniques can benefit performance, it will create additional overhead in creating positive and negative pairs of data.

Bibliography

[1] Abu-El-Haija, S., Kapoor, A., Perozzi, B., and Lee, J. N-gcn: Multi-scale graph convolution for semi-supervised node classification. In *uncertainty in artificial intelligence* (2020), PMLR, pp. 841–851.

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

[3] Adhikari, B., Zhang, Y., Ramakrishnan, N., and Prakash, B. A. Sub2vec: Feature learning for subgraphs. In *Advances in Knowledge Discovery and Data Mining: 22nd Pacific-Asia Conference* (2018), Springer, pp. 170–182.

[4] Akoglu, L., Tong, H., and Koutra, D. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery 29* (2015), 626–688.

[5] Alsentzer, E., Finlayson, S., Li, M., and Zitnik, M. Subgraph neural networks. *Advances in Neural Information Processing Systems* (2020), 8017–8029.

[6] Alsentzer, E., Li, M. M., Kobren, S. N., Network, U. D., Kohane, I. S., and Zitnik, M. Deep learning for diagnosing patients with rare genetic diseases. *medRxiv* (2022), 2022–12.

[7] Bianchi, F. M., Grattarola, D., and Alippi, C. Spectral clustering with graph neural networks for graph pooling. In *International conference on machine learning* (2020), PMLR, pp. 874–883.

[8] Brin, S., and Page, L. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer Networks* (2012), 3825–3833.

[9] Broder, A. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)* (1997), pp. 21–29.

[10] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).

[11] Cai, L., and Ji, S. A multi-scale approach for graph link prediction. In *Proceedings of the AAAI conference on artificial intelligence* (2020), vol. 34, pp. 3308–3315.

[12] Cai, T., Luo, S., Xu, K., He, D., Liu, T.-y., and Wang, L. Graphnorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning* (2021), PMLR, pp. 1204–1215.

[13] Chamberlain, B. P., Shirobokov, S., Rossi, E., Frasca, F., Markovich, T., Hammerla, N., Bronstein, M. M., and Hansmire, M. Graph neural networks for link prediction with subgraph sketching. In *International Conference on Learning Representations* (2023).

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

[15] Chen, J., Zhu, J., and Song, L. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568* (2017).

[16] Chen, L., Xie, Y., Zheng, Z., Zheng, H., and Xie, J. Friend recommendation based on multi-social graph convolutional network. *IEEE Access 8* (2020), 43618–43629.

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

[18] Chen, Y., Wu, L., and Zaki, M. J. Reinforcement learning based graph-to-sequence model for natural question generation. *arXiv preprint arXiv:1908.04942* (2019).

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

[20] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

[21] Dabhi, S., and Parmar, M. Nodenet: A graph regularised neural network for node classification. *arXiv preprint arXiv:2006.09022* (2020).

[22] Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems* (2016).

[23] Du, J., Wang, S., Miao, H., and Zhang, J. Multi-channel pooling graph neural networks. In *IJCAI* (2021), pp. 1442–1448.

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

[25] Flajolet, P., Fusy, E., Gandouet, O., and Meunier, F. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. *Discrete Mathematics Theoretical Computer Science DMTCS Proceedings vol. AH,...* (03 2012).

[26] Frasca, F., Rossi, E., Eynard, D., Chamberlain, B., Bronstein, M., and Monti, F. Sign: Scalable inception graph neural networks. In *ICML 2020 Workshop on Graph Representation Learning and Beyond* (2020).

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

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

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

[30] Hamilton, W. L. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning 14*, 3, 1–159.

[31] Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017), p. 1025–1035.

[32] Hao, Y., Cao, X., Fang, Y., Xie, X., and Wang, S. Inductive link prediction for nodes having only attribute information. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence* (2021), IJCAI'20.

[33] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[34] Heule, S., Nunkesser, M., and Hall, A. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology* (New York, NY, USA, 2013), EDBT '13, Association for Computing Machinery, p. 683–692.

[35] Hu, R., Aggarwal, C. C., Ma, S., and Huai, J. An embedding approach to anomaly detection. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)* (2016), IEEE, pp. 385–396.

[36] Huang, W., Zhang, T., Rong, Y., and Huang, J. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems 31* (2018).

[37] Huang, Y., Peng, X., Ma, J., and Zhang, M. Boosting the cycle counting power of graph neural networks with I^2-gnns. In *The Eleventh International Conference on Learning Representations* (2023).

[38] Jacob, S. A., Louis, P., and Salehi-Abari, A. Stochastic subgraph neighborhood pooling for subgraph classification, 2023.

[39] Jiang, D., Wu, Z., Hsieh, C.-Y., Chen, G., Liao, B., Wang, Z., Shen, C., Cao, D., Wu, J., and Hou, T. Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models. *Journal of cheminformatics* (2021), 1–23.

[40] Kim, D., and Oh, A. Efficient representation learning of subgraphs by subgraph-to-node translation. In *ICLR 2022 Workshop on Geometrical and Topological Representation Learning* (2022).

[41] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations* (2015).

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

[43] Le, Q., and Mikolov, T. Distributed representations of sentences and documents. In *International conference on machine learning* (2014), pp. 1188–1196.

[44] LeClair, A., Haque, S., Wu, L., and McMillan, C. Improved code summarization via a graph neural network. In *Proceedings of the 28th international conference on program comprehension* (2020), pp. 184–195.

[45] Lee, J., Lee, I., and Kang, J. Self-attention graph pooling. In *International Conference on Machine Learning* (2019), PMLR, pp. 3734–3743.

[46] Leskovec, J., Huttenlocher, D., and Kleinberg, J. Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World wide web* (2010), pp. 641–650.

[47] Li, B., Xia, Y., Xie, S., Wu, L., and Qin, T. Distance-enhanced graph neural network for link prediction. In *ICML 2021 Workshop on Computational Biology* (2021).

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

[49] Liu, C., Yang, Y., Xie, Z., Lu, H., and Ding, Y. Position-aware subgraph neural networks with data-efficient learning. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining* (2023), pp. 643–651.

[50] Louis, P., Jacob, S. A., and Salehi-Abari, A. Sampling enclosing subgraphs for link prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management* (2022), pp. 4269–4273.

[51] Louis, P., Jacob, S. A., and Salehi-Abari, A. Simplifying subgraph representation learning for scalable link prediction. *arXiv preprint arXiv:2301.12562* (2023).

[52] Lü, L., and Zhou, T. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications 390*, 6 (2011), 1150–1170.

[53] Markovitz, A., Sharir, G., Friedman, I., Zelnik-Manor, L., and Avidan, S. Graph embedded pose clustering for anomaly detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 10539–10547.

[54] Mavromatis, C., and Karypis, G. Graph infoclust: Maximizing coarse-grain mutual information in graphs. In *PAKDD* (2021).

[55] Morselli Gysi, D., Do Valle, Í., Zitnik, M., Ameli, A., Gan, X., Varol, O., Ghiassian, S. D., Patten, J., Davey, R. A., Loscalzo, J., et al. Network medicine framework for identifying drug-repurposing opportunities for covid-19. *Proceedings of the National Academy of Sciences* (2021).

[56] Namanloo, A. A., Thorpe, J., and Salehi-Abari, A. Improving peer assessment with graph neural networks. *International Educational Data Mining Society* (2022).

[57] Nishad, S., Agarwal, S., Bhattacharya, A., and Ranu, S. Graphreach: Position-aware graph neural network using reachability estimations. *arXiv preprint arXiv:2008.09657* (2020).

[58] Pan, L., Shi, C., and Dokmanić, I. Neural link prediction with walk pooling. In *International Conference on Learning Representations* (2022).

[59] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* (2019).

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

[61] Ranjan, E., Sanyal, S., and Talukdar, P. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), vol. 34, pp. 5470–5477.

[62] Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903* (2019).

[63] Rybin, D., Sun, R., and Luo, Z.-Q. Invariant layers for graphs with nodes of different types, 2023.

[64] Salehi-Abari, A., and Boutilier, C. Empathetic social choice on social networks. In *AAMAS* (2014), pp. 693–700.

[65] Salehi-Abari, A., and Boutilier, C. Preference-oriented social networks: Group recommendation and inference. In *Proceedings of the 9th ACM Conference on Recommender Systems* (2015), pp. 35–42.

[66] Salehi-Abari, A., Boutilier, C., and Larson, K. Empathetic decision making in social networks. *Artificial intelligence 275* (2019), 174–203.

[67] Shen, Y., Yan, J., Ju, C.-W., Yi, J., Lin, Z., and Guan, H. Improving subgraph representation learning via multi-view augmentation. In *ICML 2022 2nd AI for Science Workshop* (2022).

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

[69] Song, X., Ma, R., Li, J., Zhang, M., and Wipf, D. P. Network in graph neural network. *arXiv preprint arXiv:2111.11638* (2021).

[70] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* (2014), 1929–1958.

[71] Suresh, S., Li, P., Hao, C., and Neville, J. Adversarial graph augmentation to improve graph contrastive learning. *Advances in Neural Information Processing Systems 34* (2021), 15920–15933.

[72] Tan, Q., Zhang, X., Liu, N., Zha, D., Li, L., Chen, R., Choi, S.-H., and Hu, X. Bring your own view: Graph neural networks for link prediction with personalized subgraph selection. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining* (2023), pp. 625–633.

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

[74] Wang, X., Yang, H., and Zhang, M. Neural common neighbor with completion for link prediction. *arXiv preprint arXiv:2302.00890* (2023).

[75] Wang, X., and Zhang, M. Glass: Gnn with labeling tricks for subgraph representation learning. In *International Conference on Learning Representations* (2021).

[76] Wang, Z., Cao, Q., Shen, H., Bingbing, X., Zhang, M., and Cheng, X. Towards efficient and expressive gnns for graph classification via subgraph-aware weisfeiler-lehman. In *Proceedings of the First Learning on Graphs Conference* (2022), pp. 17:1–17:18.

[77] Wang, Z., Liao, J., Cao, Q., Qi, H., and Wang, Z. Friendbook: a semantic-based friend recommendation system for social networks. *IEEE transactions on mobile computing* (2014), 538–551.

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

[79] Wu, L., Cui, P., Pei, J., and Zhao, L. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer Singapore, Singapore, 2022.

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

[81] Xu, D., Cheng, W., Luo, D., Chen, H., and Zhang, X. Infogcl: Information-aware graph contrastive learning. *Advances in Neural Information Processing Systems 34* (2021), 30414–30425.

[82] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations* (2019).

[83] Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning* (2018), PMLR, pp. 5453–5462.

[84] Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning* (2018), pp. 5453–5462.

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

[86] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018), p. 974–983.

[87] Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems 31* (2018).

[88] You, J., Ying, R., and Leskovec, J. Position-aware graph neural networks. In *International Conference on Machine Learning* (2019), PMLR, pp. 7134–7143.

[89] You, Y., Chen, T., Shen, Y., and Wang, Z. Graph contrastive learning automated. In *International Conference on Machine Learning* (2021), PMLR, pp. 12121–12132.

[90] You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., and Shen, Y. Graph contrastive learning with augmentations. *Advances in neural information processing systems 33* (2020), 5812–5823.

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

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

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

[94] Zhang, M., and Chen, Y. Link prediction based on graph neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (2018), p. 5171–5181.

[95] Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. *Proceedings of the AAAI Conference on Artificial Intelligence* (2018).

[96] Zhang, M., and Li, P. Nested graph neural networks. In *Advances in Neural Information Processing Systems* (2021), pp. 15734–15747.

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

[98] Zhao, T., Liu, G., Günnemann, S., and Jiang, M. Graph data augmentation for graph machine learning: A survey. *arXiv preprint arXiv:2202.08871* (2022).

[99] Zhao, T., Liu, Y., Neves, L., Woodford, O., Jiang, M., and Shah, N. Data augmentation for graph neural networks. In *Proceedings of the aaai conference on artificial intelligence* (2021), pp. 11015–11023.

[100] Zhu, S., Yu, K., Chi, Y., and Gong, Y. Combining content and link for classification using matrix factorization. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval* (2007), pp. 487–494.

[101] Zhu, Y., Xu, Y., Yu, F., Liu, Q., Wu, S., and Wang, L. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021* (2021), pp. 2069–2080.

[102] Zitnik, M., Agrawal, M., and Leskovec, J. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics 34*, 13 (2018), i457–i466.

[103] Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., and Gu, Q. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in Neural Information Processing Systems 32* (2019).