Scalable Subgraph Representation Learning through Simplification

by

Paul Louis

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: **Paul Louis**

Master of Science in Computer Science

Thesis title: Scalable Subgraph Representation Learning through Simplification

An oral defense of this thesis took place on June 5, 2023 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Pooria Madani
Research Supervisor	Dr. Amirali Salehi-Abari
Examining Committee Member	Dr. Ken Pu
Thesis Examiner	Dr. Faisal Qureshi

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

Link prediction on graphs is a fundamental problem. Subgraph representation learning approaches (SGRLs), by transforming link prediction to graph classification on the subgraphs around the links, have achieved state-of-the-art performance in link prediction. However, SGRLs are computationally expensive, and not scalable to large-scale graphs due to expensive subgraph-level operations. To unlock the scalability of SGRLs, we propose a new class of SGRLs, that we call *Scalable Simplified SGRL* (*S3GRL*). Aimed at faster training and inference, *S3GRL* simplifies the message passing and aggregation operations in each link's subgraph. *S3GRL*, as a scalability framework, accommodates various subgraph sampling strategies and diffusion operators to emulate computationally-expensive SGRLs. We propose multiple instances of *S3GRL* and empirically study them on small to large-scale graphs. Our extensive experiments demonstrate that the proposed *S3GRL* models scale up SGRLs without significant performance compromise (even with considerable gains in some cases), while offering substantially lower computational footprints (e.g., multi-fold inference and training speedup).

Keywords: Graph Neural Networks, Link Predictions, Subgraph Representation Learning.

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

Paul Louis

Statement of Contributions

I hereby certify that I have been the primary contributor to this thesis by developing the algorithms, implementing them, and designing & executing all experiments. I have also written most content of this thesis. However, some texts of this thesis are borrowed from the papers [57, 58], coauthored by my thesis supervisor, a labmate, and me, in which I was the primary author undertaking the above-listed responsibilities.

Acknowledgments

I would like to express my sincere, eternal gratitude and love for my parents, Appa and Ammi, who have been a constant source of support and encouragement from the beginning. This thesis would not be if it were not for them. I also want to express my love for my pets, Tuttu and Lily, for their love (and barks) are endless and missed. I wish to reunite with everyone soon and hope that they are proud of what my time away has accomplished.

I also wish to express my gratitude to Shweta, who has been a constant source of support, motivation and strength throughout this journey. I also want to thank Dr. Faisal Qureshi for providing pointers on how to improve this thesis and present it better. Finally, I would like to thank my supervisor, Dr. Amirali Salehi-Abari, for all the constructive criticisms, reviews and research directions provided since the inception of this work.

Contents

List of Figures

List of Tables

List of Abbreviations

AA ..... Adamic Adar

roc-auc ..... Area under the ROC Curve

AUC ..... Area under the ROC Curve

BFS ..... Breadth-first Search

CNN ..... Convolutional Neural Network

DFS ..... Depth-first Search

DE-GNN ..... Distance Encoding Graph Neural Network

DRNL ..... Double Radius Node Labeling

GB ..... Gigabytes

GAT ..... Graph Attention Networks

GCN ..... Graph Convolutional Networks

GIN ..... Graph Isomorphism Network

GNNs ..... Graph Neural Networks

GRL ..... Graph Representation Learning

HR ..... Hit Rate

i.i.d ..... Independent and Identically Distributed

MRR ..... Mean Reciprocal Rank

MB ..... Megabytes

MPGNN ..... Message Passing Graph Neural Networks

MLP ..... Multilayer Perceptron

NLP ..... Natural Language Processing

PoS ..... Powers of Subgraphs

PA ..... Preferential Attachment

RNN ..... Recurrent Neural Network

RA ..... Resource Allocation

S3GRL ..... Scalable Simplified Subgraph Representation Learning

SGD ..... Stochastic Gradient Descent

SGRL ..... Subgraph Representation Learning

SGRLs ..... Subgraph Representation Learning approaches

SoP ..... Subgraphs of Powers

WL-Test ..... Weisfeiler-Lehman Isomorphism Test

List of Notations

G ..... used to represent graph structured data

V ..... set containing a graph's vertices/nodes

E ..... set containing a graph's edges

$\mathbf{X}$ ..... matrix containing initial node features

(u, v) ..... target nodes in a link

T ..... target node set

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

L ..... number of convolution layers

l ..... l-th convolution layer

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

$\mathbf{H}$ ..... matrix containing intermediate learnt node embeddings

σ ..... activation function

$\mathbf{I}$ ..... identity matrix

$\tilde{\mathbf{A}}$ ..... normalized adjacency matrix summed with identity matrix

$\hat{\mathbf{A}}$ ..... normalized adjacency matrix

$\mathbf{Z}$ ..... final output embedding matrix of a GNN

$\mathbf{D}$ ..... diagonal degree matrix

$\tilde{\mathbf{D}}$ ..... diagonal degree matrix associated with $\tilde{\mathbf{A}}$

$\mathbf{y}$ ..... label vector associated with nodes

y_u ..... label associated with node u

ξ ..... softmax operation

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

$p(u, v)$ ..... link formation probability for nodes (u, v)

$\mathbf{z_u}$ ..... learnt representation of node u in $\mathbf{Z}$

Ω ..... learnable non-linear function

$\mathbf{q}_{uv}$ ..... joint link representation for nodes (u, v)

$d_{x,y}$ ..... geodesic distance between nodes x and y.

h ..... number of hops of the enclosing subgraph.

G^h_{uv} ..... the h-hop enclosing subgraph around link (u, v).

G_{uv} ..... the enclosing subgraph around link (u, v).

V_{uv}^{h} ..... the set of nodes in the enclosing h-hop subgraph around link (u, v).

$\mathbf{X}_{uv}$ ..... the matrix containing the initial node features for the subgraph G_{uv}.

d ..... dimensionality of initial node feature matrix $\mathbf{X}$.

d' ..... dimensionality of the learnt nodal feature matrix $\mathbf{Z}$.

$\bigoplus$ ..... concatenation operator.

$\mathbf{Y}$ ..... the matrix containing node class probabilities.

$\mathbf{M}_{uv}$ ..... linear diffusion operator matrix for the subgraph G_{uv}.

r ..... number of operators in *S3GRL*.

Ψ ..... subgraph sampling strategy in *S3GRL*.

Φ ..... diffusion operator in *S3GRL*.

S ..... sampling operator set in *S3GRL*.

$\mathbf{Z}_{uv}$ ..... joint nodal-representation matrix for the subgraph G_{uv}.

p ..... in-out hyperparameter of node2vec.

q ..... return hyperparameter of node2vec.

k ..... number of random walks in ScaLed.

Chapter 1

Introduction

This chapter introduces the reader to graph neural networks and its application on learning graph structured data. We detail the multiple downstream tasks which are tackled using graph structured data with an emphasis on the link prediction downstream task. We then highlight the current limitations of graph neural network models on link prediction. Following that, we highlight the thesis contributions to the area, that is aimed at overcoming the computational limitations of current works.

1.1 Motivation

Graphs are ubiquitous in representing relationships between interacting entities in a variety of contexts, ranging from social networks [15] to polypharmacy [113]. Graphs can be used to model entities (as nodes) and can be used to uniquely capture interactions amongst the entities (as edges). Due to their versatility in modeling different kinds of data, they have a variety of applications not limited to: classifying users in a social network, capturing relationships between users in a social network, capturing interaction between groups of proteins, etc. A toy example of how graph structured

data finds application in modeling social interactions is presented using the Karate network graph [99] in Figure 1.1.

Three tasks stand out as the predominant downstream tasks which primarily rely on graph structured. They are node classification, graph classification and link prediction. Among these three, node classification is one of the most popular downstream tasks. The goal in node classification is to tag or classify each node in the graph with one or more pre-defined labels [50]. Some examples of node classification include; classifying proteins based on their interactions in human tissue [114], predicting what community a post was published in [102, 34], classifying documents based on their citations to other documents and bag-of-words representations [93], predicting evaluations using peer assessments [65] etc.

Another focused task on graphs is graph classification. The goal is to classify groups of nodes (i.e., graphs) to one or more pre-defined labels. Graph classification distinguishes itself from node classification as the task is to classify groups of nodes as opposed to classifying individual nodes. An example of graph classification is predicting molecular properties of a group of atoms given their chemical bonds as edges [64, 41, 40].

Among the multiple use cases of modeling data as graphs, one of the main tasks (other than node and graph classification) on graphs is link prediction [56]. Link prediction involves predicting future or missing relationships between pairs of entities, given the graph structure. Link prediction plays a fundamental role in impactful areas, including molecular interaction predictions [42], recommender systems [96, 4], and protein-protein interaction predictions [111]. The early solutions to link prediction relied on hand-crafted, parameter-free heuristics based on the proximity of the source-target nodes. These heuristics can be classified based on their distance (or number of hops) to the source and target nodes that form a link. For example, common

neighbors score the likelihood of formation of a link based on the number of common neighbors. As such, these heuristics and others (e.g., preferential attachment [5] and Jaccard index), which rely on the first hop connecting nodes of source and target nodes can be considered as first-order based heuristics. Second-order heuristics, such as Resource Allocation [112] and Adamic Adar [2] rely on nodes that are up to two hops away from the links. Finally, higher-order heuristics such as PageRank [68], SimRank [45], Katz Index [47] etc. rely on nodes multiple hops away from the source and target node pairs. However, these hop-based heuristic methods require extensive domain knowledge for effective implementation. For example, having high number of common neighbors in a protein-protein interaction network implies that links are less likely to form [52], meaning a trivial application of neighborhood overlap based heuristics would fail. Moreover, these heuristics did not account for any features that were associated with nodes. As a result, using such pre-defined heuristics may fail to adequately summarize how links form in complex networks.

Recently, the success of *graph neural networks (GNNs)* in learning latent representations of nodes [50] has led to their application in link prediction. This is done by aggregating the source-target nodes' representations to construct link representations [49, 70]. However, as vanilla message passing GNN architectures (e.g., GCN [50]) learn a pair of nodes' representations independent of their relative positions to each other, aggregating independent node representations results in poor link representations [107]. To circumvent this, the state-of-the-art subgraph representation learning approaches (SGRLs) [104] learn the enclosing subgraphs of pairs of source-target nodes, while augmenting node features with structural features. SGRLs cast the link prediction problem as a binary classification problem of the subgraph enclosing the target link. Effectively, SGRLs convert link prediction into binary graph classification, where the task is to determine if the subgraph enclosing a target link

(e.g., the h-hop enclosing subgraph [104]) is closed (link is likely to form) or open (link is unlikely to form). The enclosing subgraph based learning coupled with the injection of distance information or labels into nodes relative of the source and target, effectively allow GNNs to learn a bag-of-heuristics. For example, the first-hop subgraph extraction with distance labels calculated for nodes allows an underlying GNN to generalize first-order based heuristics [104]. In theory, SGRLs can approximate any higher-order heuristic by utilizing a h-hop enclosing subgraph [104]. Where, a higher h value allows for better generalization. As such, SGRLs allow one to break-free from pick and choosing network heuristics and have shown to be successful on multiple link prediction datasets [69, 104, 54, 103].

The core issue hampering the deployment of GNNs and SGRLs is their high computational demands on large-scale graphs (e.g., OGB datasets [41, 40]). This high computational demand primarily stems from the enclosing subgraph extractions pertaining to each individual links and the labeling associated with the nodes in the enclosing subgraphs. For example, a popular link prediction dataset, ogbl-ddi [86, 40], has an average degree of 500 nodes. As such the 1-hop enclosing subgraphs on average contain 500 nodes, leading to largely overlapping subgraphs. Moreover, SGRLs utilize GNNs as the underlying neural model utilized to learn on the enclosing subgraphs. The underlying GNNs in SGRLs make use of multiple nested layers of convolutions, with weights associated to each layer. This learning mechanism is to be done per epoch, over each individual training subgraph, leading to increasing training and inference costs. Generally, subgraph extraction over a fixed hop might lead to *neighborhood explosion*, where subgraphs sometimes become as large as the original graphs. This computational overhead is further exaggerated by the need to label each node in each enclosing subgraph as well. As such, many approaches have been proposed for GNNs to relieve this bottleneck, ranging from sampling [13, 102] to

simplification [87] techniques. However, these techniques fail to be applied directly in SGRL models. Although recent work has focused on tackling SGRL scalability by sampling [95, 57] or sketching [12] approaches, little attention is given to simplification techniques [87]. We focus on improving the scalability in SGRLs, by simplifying the underlying training mechanism.

1.2 Contribution

Inspired by the simplification techniques in GNNs such as SGCN [87] and SIGN [26], we propose *Scalable Simplified SGRL* (*S3GRL*) framework, which extends the simplification techniques to SGRLs. Our *S3GRL* learning framework is flexible in emulating many SGRLs (such as SEAL [104], DE-GNN [54]) by offering choices of subgraph sampling and diffusion operators, producing different-sized convolutional filters for each link's subgraph. Our *S3GRL* benefits from precomputing the subgraph diffusion operators, leading to an overall reduction in runtimes while offering a multi-fold scale-up over existing SGRLs in training/inference times. Our *S3GRL* achieves this speed-up by having precomputed convolutional filters calculated once before commencing the training/inference phase. In comparison, existing SGRLs need to run multi-layer convolution filters per epoch, which leads to a slow down in comparison to our framework which utilizes the precomputed operators and has only a single learnable weight matrix.[1] We propose and empirically study multiple instances of our *S3GRL* framework. Our extensive experiments show substantial speedups in *S3GRL* models over state-of-the-art SGRLs while maintaining and sometimes surpassing their efficacies.

We can summarize our contribution as follows,

[1]For added expressivity, the single learnable weight matrix is replaced with an MLP for experiments involving a small subset of graph datasets.

1. We extend simplification methods introduced in graph neural networks to the context of subgraph representation learning with the introduction of our novel *S3GRL*. Our *S3GRL* is the first time simplifications methods have been studied in the context of subgraph representation learning and applied to link prediction tasks.

2. We offer multiple instantiations of our *S3GRL* to showcase the flexibility and robustness of the learning framework.

3. We study and evaluate the performance of the instantiations of *S3GRL* on small to large-scale graph datasets under various evaluation criteria.

4. Our empirical study and analyses of the *S3GRL* framework cover over 14 baselines utilized in link prediction studies on 14 graph datasets, ranging from small to large-scale graphs.

5. Our results showcase the efficacy improvements showcased by our *S3GRL* while offering multi-fold speed-up in training and inference.

6. Our results also showcase a multi-fold reduction in storage requirements: our *S3GRL* models' prepared dataset sizes in comparison to dataset sizes in SGRLs like SEAL are magnitudes lesser and more feasible.

7. We share our findings and framework as an open-source framework,[2] encouraging other researchers to come up with their own instantiations of the *S3GRL* framework, thereby helping advance link prediction research.

[2]https://github.com/venomouscyanide/S3GRL_OGB

1.3 Thesis Organization

We first introduce the reader to the problem being tackled by introducing graphs and graph neural networks in Chapter 1. Following that, in Chapter 2, we provide a detailed background into graph representation learning and subsequently subgraph representation learning. Upon describing the background materials, we then focus on doing a thorough literature review of works most related to our contributions in Chapter 3, and introduce our novel learning framework in Chapter 4. We conclude with experiments and discussions involving our learning framework in Chapter 5, and with conclusions and future works planned in Chapter 6.

1.4 Summary and Impact

We introduce a novel subgraph representation learning framework, *S3GRL*, aimed at faster training and inference for link representation learning on graph-structured data. We achieve this by overcoming the limitations of current slower SGRLs by emulating expensive message passing operations through efficient precomputations. These efficient precomputations only need to be run once and offer a substantial decrease in model training and inference times. This decrease in training and inference times comes from the fact that *S3GRL* does not have multi-layers of convolution filters (and weights), and relies on the once precomputed filters and a single linear weight matrix for training and inference. Our *S3GRL* model is empirically shown to scale to graphs with millions of nodes and edges, all the while retaining the expressivity of more computationally demanding SGRLs. We propose *S3GRL* as a computational lighter SGRL method that can be used in place of more demanding learning methodologies such as SEAL [104] or any other SGRL method [69, 55].

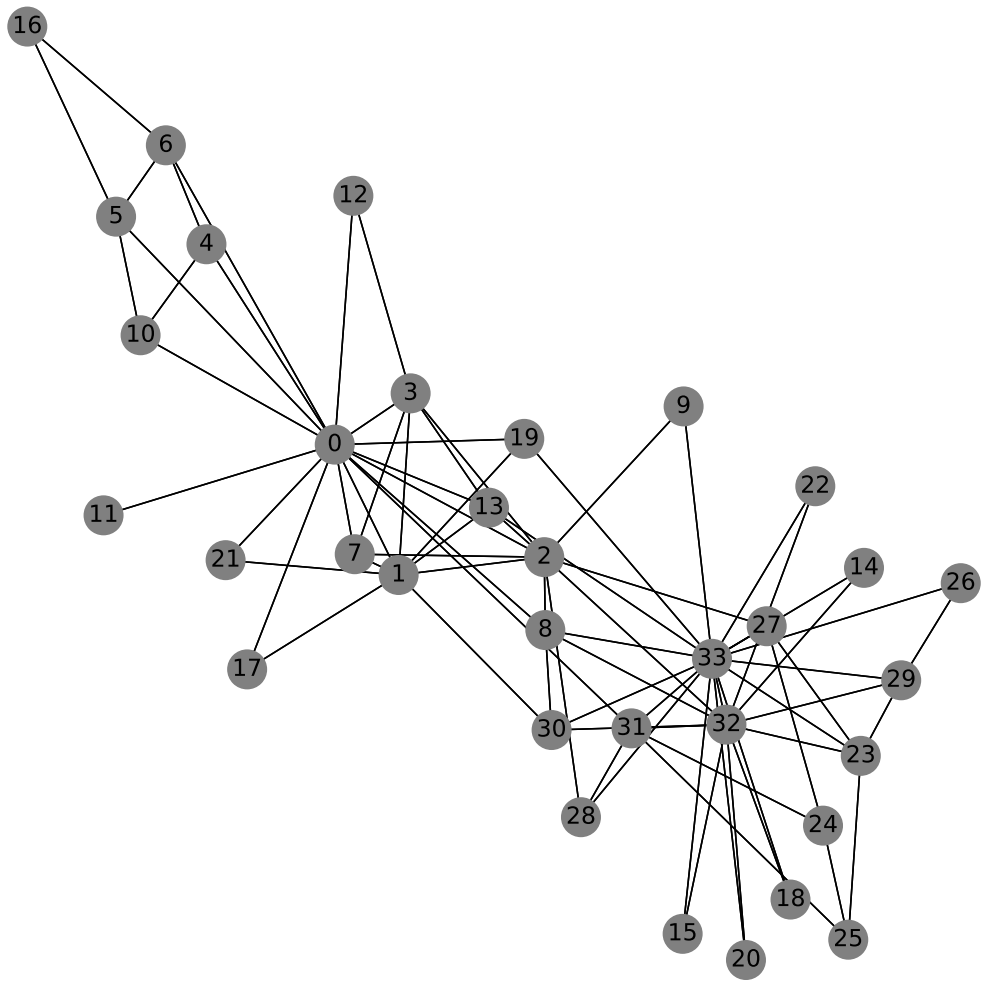


Figure 1.1: An example of modeling data as a graph. Here we have the famous Zachary's Karate Club [99] as an example of modeling social interaction data as a graph. Zachary studied the interaction of 34 members of a Karate club based on their interactions outside of the club. The club later on split into two; lead by nodes 0 and 33. Zachary, using the captured graph and the nodes interactions (edges), was able to successfully predict which nodes will decide to go to which split.

Chapter 2

Background

In this chapter we provide an introduction to graph structured data and a detailed background about graph neural networks and its impact and applications on key downstream tasks. We place our focus on providing the reader with the necessary background knowledge to understand the current limitations of graph neural networks on learning link representations. Following that, we introduce them to subgraph representation learning approaches that better learn link representations and highlight the computational bottlenecks of subgraph representations methods. We close discussions motivating the need for computational lighter subgraph representation learning models in practice.

2.1 Learning on Graphs

Graph structured data has much prevalence in our society. Observed data such as user interaction, collaborations between authors, protein-protein interaction, drug-drug interaction etc. can be modeled as graphs. This has lead to the rise and prevalence of graph neural networks methods, which are currently the de-facto method in learn-

ing graph data. We provide an example of modeling data as a graph in Figure 1.1. Graph Neural Networks have emerged as the widely accepted solution for learning rich dense latent representations of graph data due to its ability to model graph data and interactions between the entities very well. Graph Neural Networks have had immense success in the task of: (1) node classification, (2) link prediction, and (3) graph classification downstream tasks as well. One of the most interesting and rapidly developing domain in this field is link prediction. By modeling the interactions between nodes in a graph as a link prediction problem, graph neural networks can learn the representation of a pair of nodes (i.e., a link) and treat this as a classification problem. Link prediction has found applications in many fields ranging from predicting drug side effects, drug-repurposing [31], understanding molecule interactions [42], building recommender system [51, 96, 4], knowledge graph completion [74] and predicting friendship between a pair of users [15].

2.1.1 What is a Graph?

Before we dive deep into the history and background of graph representation learning, we formally define a graph and briefly talk about the different types of graphs. A graph is represented by $G = (V, E, X)$, where $V = \{1, 2, .., , n\}$ is the set of nodes, $E \subseteq V \times V$ represents the set of edges and $\mathbf{X} \in \mathbb{R}^{n \times d}$ represents the d-dimensional features associated with the node, if any. Furthermore, we use an adjacency matrix $\mathbf{A} \in V \times V$ to represent the connectivity of the graph, where $\mathbf{A}_{u,v} = 1$ implies the existence of an edge $(u, v) \in E$. Typically, we deal with graphs which are undirected, implying if an edge (u, v) exists, i.e, $\mathbf{A}_{u,v} = 1$, it also means that $\mathbf{A}_{v,u} = 1$. In the case of directed graphs, the reverse link (existence of (v, u) given (u, v)) is not always guaranteed. Graphs can also be weighted, where the adjacency matrix contains weights associated with the edges in place of binary values, i.e., $\mathbf{A}_{u,v} \neq 0$ if $(u, v) \in E$. For example,

weights in a citation graph can be the year in which the citation occurred. In case the graph contains nodes and edges of different types, we call this graph a multi-relational graph. Examples of multi-relational graphs include knowledge graphs where edges are represented as tuples of (u, r, v), where r represents the type of relationship that exists between the nodes (u, v). For e.g., in a biomedical knowledge graph, one can represent medicines used to treat diseases as follows: a medicine (node u) used to treat a disease (node v) forms an edge, where r is the relationship between (u, v) representing the 'treatment'. In this work we primarily focus on homogeneous graphs, where nodes and edges are of the same type.

2.1.2 Early Work in Graph Representation Learning

One of the earliest work in graph representation learning focused on learning rich latent representations of nodes based on random walks centered around the nodes in the networks.[1] This line of work include DeepWalk [72], LINE [79] and node2vec [29], and is broadly referred to as *shallow embedding approaches*. DeepWalk was the pioneer model in this category of shallow embedding approaches. In DeepWalk, the authors extend the idea of Skip-Gram [62] and negative-sampling [63] to the context of networks. However, these shallow embeddings methods had several drawbacks that limited them from the context of being applied to real-world tasks. These disadvantages include inability to consume the initial nodal features associated with the node, $\mathbf{X}$, and their inability to be applied in an inductive learning setting. Specifically, we refer to inductive learning setting as one where nodes in testing phase are never seen during training.

Another line of early research in graph representation learning is the matrix factorization methods, where a condensed latent matrix is learnt that approximates

[1] In this thesis we interchangeably refer to graph structured data as both networks and graphs.

some similarity measure between the nodes in the graph [51, 80]. Leading works, categorized under matrix factorization, include GraRep [11], and HOPE [67].

2.1.3 Message Passing Graph Neural Networks

The drawbacks faced by shallow embedding methods lead to the growing interest in Message Passing Graph Neural Networks (MPGNNs)[2] [8, 19] such as Graph Convolutional Networks (GCN) [50]. In MPGNNs, node feature representations are iteratively updated by first aggregating neighborhood features and then updating them through non-linear transformations. MPGNNs differ in formulations of the respective aggregation and update functions [32]. Essentially, MPGGNs extend convolutions applied to structured data (such as images) to non-euclidean data (graphs). We dedicate this subsection to focus on the update and aggregation formulations that make the core of all modern MPGNNs.

Graph Neural Networks (GNNs) typically take as input the adjacency matrix $\mathbf{A}$ and initial nodal features $\mathbf{X}$, apply L layers of convolution-like operations, and output $\mathbf{Z} \in \mathbb{R}^{n \times d'}$, containing d'-dimensional representation for each node as a row. For example, the l^{th}-layer output of Graph Convolution Network (GCN) [50] is given by

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

where, the *normalized adjacency matrix* $\hat{\mathbf{A}}$ is defined as,

$$\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}} \tag{2.2}$$

[2]We use MPGNNs and GNNs interchangeably even though GNNs represent the broader class in GRL.

where $\tilde{\mathbf{A}} = \mathbf{A}+\mathbf{I}$, $\tilde{\mathbf{D}}$ is the diagonal matrix with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and σ is a non-linearity function (e.g., ReLU). Here, $\mathbf{H}^{(0)} = \mathbf{X}$ and $\mathbf{W}^{(l)}$ is the l^{th} layer's learnable weight matrix. After L stacked layers of Eq. 2.1, GCN outputs nodal-representation matrix $\mathbf{Z} = \mathbf{H}^{(L)}$. In the event that initial nodal features, $\mathbf{X}$, are absent, one can initialize it to an identity matrix or even run the shallow embedding methods explained above and use its output as the initial node features to be input to a GNN.

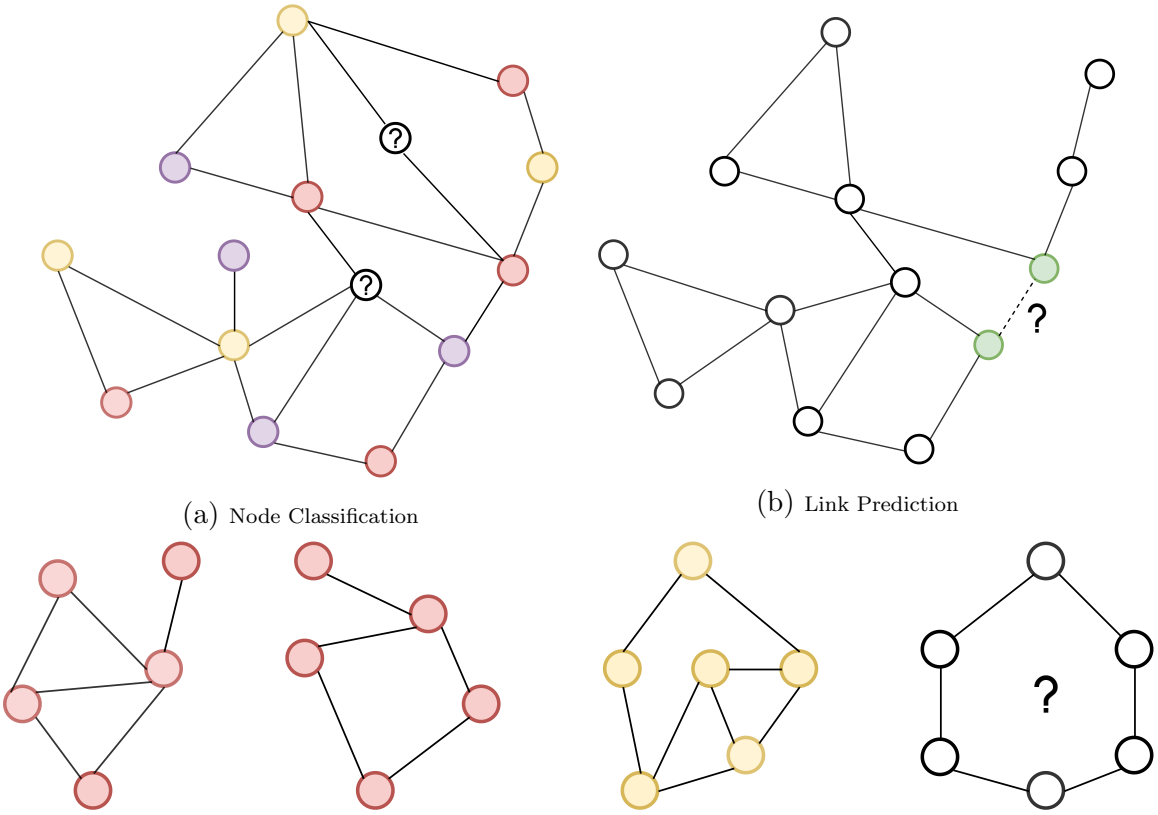
Multiple GNN architectures have been introduced since the inception of GCN. Some popular and pivotal models introduced include GraphSAGE [33], GIN [89] and GAT [82] models. Each of these methods differ in their formulation of the aggregate and update method in Eq. 2.1.

2.2 Downstream Tasks

The primary application of GNNs is in node related tasks. Examples of such tasks include node classification (e.g., classifying user in a social network), link prediction (e.g., predicting if two nodes are likely form a link) and graph classification (e.g., classifying groups of proteins). We summarize and capture all three primary downstream tasks performed using Graph Neural Networks in Figure 2.1.

2.2.1 Node Classification

Among the downstream tasks associated with graph structured data, node classification stands out as the most popular task. The objective in node classification is to label each node from a set of predefined labels. For example, for a node u, we wish to identify the label y_u associated with it. An example of this would be identifying if a user (node) in a social network is a bot or not. We can apply both shallow-embedding based and GNN based models for this task by computing the softmax over the out-



(a) Node Classification

(b) Link Prediction

(c) Graph Classification

Figure 2.1: This figure is best viewed in color. The three primary downstream tasks solved using Graph Neural Networks. Subfigure 2.1a gives an example of node classification, where the task is to predict the class to which a node belongs to. Subfigure 2.2 capture the link prediction task, where the goal is to predict the likelihood that two nodes form a link. Finally, Subfigure 2.1c capture the graph classification problem, where the goal is to classify the class a graph belongs to.

put embeddings $\mathbf{Z}$ output by any of these models. I.e., one can calculate the labels associated with all the nodes in the network with

$$\mathbf{y} = \xi(\mathbf{Z}), \tag{2.3}$$

where ξ is the softmax operation over the learnt nodal embeddings $\mathbf{Z}$ to output class probabilities associated with each node. The difficulty in learning nodal embeddings stems from the fact that nodal data in a graph is not independently and identically distributed (i.i.d). This is primarily due to each node's links to other nodes in the graph. This relationship/linkage makes each node's embedding reliant on those of its neighboring nodes, its neighbors' neighboring nodes, and so on.

2.2.2 Graph Classification

Graph classification is similar to node classification. However, the distinction comes from the fact that in graph classification, we intend to classify groups of nodes rather than individual nodes. An example of graph classification is classifying posts into communities, given the interaction graph of the users engaged in the post [91]. Application of shallow-embeddings methods and GNNs to graph classification can be achieved by *pooling* the nodes for which we wish to predict the class and then applying softmax similar to node classification. For instance,

$$\mathbf{y} = \xi(pool(\mathbf{Z}, G)), \tag{2.4}$$

where *pool* is any *permutation invariant* pooling operation such as, max, min or mean pooling of the target graph G's learnt nodal embeddings $\mathbf{Z}$ and ξ is the softmax operation. Eq. 2.4 is similar to Eq. 2.3 except that we need to pool the target graph's nodes before calculating the label. We also note that in graph classification, since

the objective is to make independent predictions related to each graph, the input graphs follow an i.i.d. As a result, one can consider graph classification to be easier in comparison to the node classification problem.

It is also worth pointing out that both node and graph classification can belong to either of the two categories; multi-label and multi-class classification. In multi-class classification, we are tasked with tagging an entity (node or graph) with one label. Whereas, in multi-label classification we are tasked with tagging an entity with one or more labels. In both cases, during training, only a subset of the nodes or graphs will be exposed. An independent set of unseen nodes or graphs are then used to make the prediction using the learnt (frozen) weights of GNNs.

2.2.3 Link Prediction

Another important application of graph neural networks is in link prediction. Link prediction forms the primary focus of this thesis. Many solutions to link prediction problem [56, 59, 60, 83, 53] has been proposed ranging from simple heuristics (e.g., common neighbors, Adamic-Adar [2], Katz [47]) to graph neural networks (GNNs) [49, 104, 69, 34, 9, 10]. The link prediction problem with an example is highlighted in Figure 2.2.

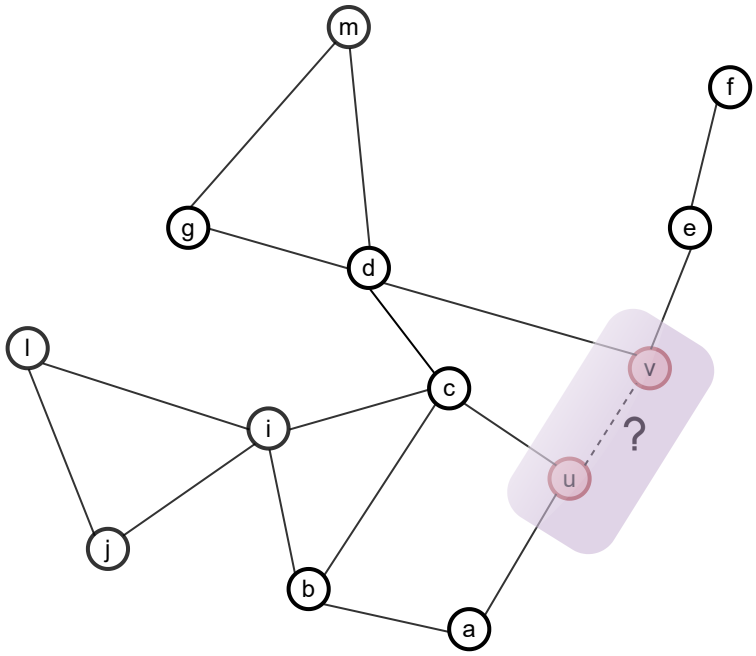


Figure 2.2: This figure is best viewed in color. An example of the link prediction problem. The task in link prediction is to predict the likelihood that two nodes form a link. In this example, the task is to predict the likelihood that u and v form a link given the graph structure enclosing both nodes.

As mentioned earlier, link prediction heuristics can be classified into different categories based on their distances (or number of hops) to the target links. For example, first-order heuristics include common neighbor and preferential attachment. Second-order heuristics include resource allocation and Adamic Adar. We dedicate the next few sections to formulate some of the most popular first-order and second-order link prediction heuristics.

The first-order heuristic, common neighbors score (CN), can be formulated as,

$$CN(u, v) = |N(u) \cap N(v)| \tag{2.5}$$

where, $N(u)$ denotes the neighborhood set of node u. Similarly, one can formulate preferential attachment score (PA) as,

$$PA(u,v) = |N(u)| * |N(v)|, \tag{2.6}$$

where $*$ denotes multiplication.

As seen in the above formulations, both CN and PA estimate the first-order common neighborhood overlap to determine the likelihood that two nodes u and v form a link. Second-order heuristics simply extend this neighborhood overlap calculation to include second-order nodes as well. For example, the Adamic Adar score (AA) can be formulated as

$$AA(u,v) = \Sigma_{k \in N(u) \cap N(v)} \frac{1}{log(|N(k)|)} \tag{2.7}$$

Similarly, another popular second-order heuristic, resource allocation (RA) can be formulated as,

$$RA(u,v) = \Sigma_{k \in N(u) \cap N(v)} \frac{1}{|N(k)|} \tag{2.8}$$

As seen in the formulation of RA and AA, the scoring is dependent on the neighborhood overlap calculation involving nodes that are up to 2 hops away from the target pairs. Higher-order heuristics such as PageRank [68] and SimRank [45] perform their formulations involving nodes that are multiple hops away from the target link.

Even though link prediction heuristics preform well on some datasets, they do not perform well across all datasets. For example, on datasets where heterophily is exhibited, link prediction heuristics such as Adamic Adar and common neighbor, which score links based on neighborhood overlap are seen to fail. Moreover, such heuristics also are parameter-free, meaning they cannot be trained to generalize over any given dataset. Finally, it is also worth nothing that most heuristics do not

consume nodal features in their calculations.

To overcome the above listed limitations of heuristic based methods, shallow embedding methods and GNNs were suggested for use in solving link prediction tasks. To apply GNN or latent feature based methods for solving the link prediction objective, one can consume the learnt latent representation matrix $\mathbf{Z}$, and, for any target nodes $T = \{u, v\}$, compute its joint link representation:

$$\mathbf{q}_{uv} = \text{pool}\left(\mathbf{z}_u, \mathbf{z}_v\right), \tag{2.9}$$

where, $\mathbf{z}_u$, $\mathbf{z}_v$ are u's and v's learned representations in $\mathbf{Z}$, and pool is a pooling/readout operation (e.g., Hadamard product or concatenation) to aggregate the pairs' representations. Then, the link probability $p(u, v)$ for the target is given by

$$p(u, v) = \Omega(\mathbf{q}_{uv}), \tag{2.10}$$

where Ω is a learnable non-linear function (e.g., MLP) transforming the target embedding $\mathbf{q}_{uv}$ into a link probability.

Why GNNs fail on capturing rich link representations

Even though GNNs show highly competitive performance for node and graph classification tasks, it falls short in link prediction tasks. This shortcoming of GNNs on link prediction task is due to the inability of GNNs in capturing node representations that can distinguish the different roles played by nodes in forming individual links [107]. We illustrate this shortcoming further with the example plotted in Figure 2.3. In the figure, a GNN, in the absence of node features, will give the same representation to the nodes $V1$, $V3$ and $V4$. This arises from the fact that every node in the input graph has the same neighboring node set (each node has two neighbors each). As the

neighboring node set of each node in the graph is the same, each node ends up with the same vector embedding when calculated by a L-layer MPGNN. This problem has been referred to as the *automorphic node* problem [12]. As a result, the link representations , calculated by any permutation invariant operation, for both $(V1, V3)$ and $(V1, V4)$ will be the same. However, one can argue that it is more likely that the link $(V1, V3)$ will have a greater chance of forming a link due to the presence of a common neighbor when compared to the link $(V1, V4)$.

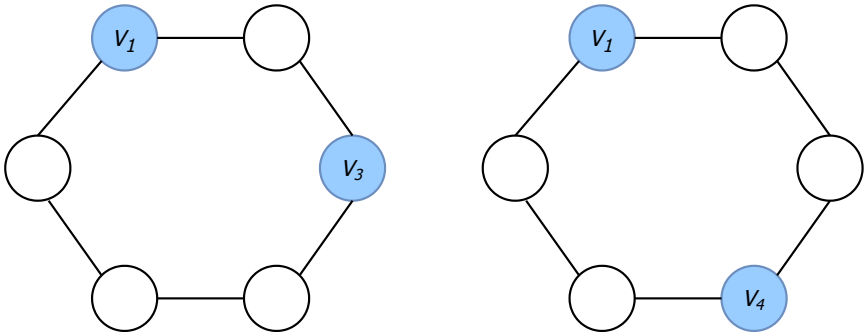


Figure 2.3: An example of when GNNs for link prediction can fail. In this example GNNs, in the absence of nodal features, will give score the chance of linkage of $(V1, V3)$ the same as $(V1, V4)$. However, it is more likely that $(V1, V3)$ forms a link compared to $(V1, V4)$, as $(V1, V3)$ share a common neighbor.

As shown above, the class of MPGNN solutions for link prediction falls short in capturing graph automorphism and different structural roles of nodes in forming links [107]. Subgraph representation learning approaches (SGRLs) [104] successfully overcome this limitation by casting the link prediction problem as a binary graph classification on the enclosing subgraph about a link, and adding structural labels to node features. We capture how SGRLs overcome the automorphic node problem in GNNs in Figure 2.4. As seen in the figure, SGRLs first extract the h-hop enclosing subgraph (with $h = 1$ in the example), about the target links $(V1, V3)$ and $(V1, V4)$. Following this, each extracted subgraph is fed into a GNN to learn if the target link will be formed or not. By doing so, the learnt vector representations for the target

link $(V1, V3)$ is different from $(V1, V4)$. This is primarily owing to the the fact the the extracted subgraphs around both target links are unique (see Figure 2.4).

The above discovery led to the emergence of subgraph representation learning approaches (SGRLs) [104, 55], which offer state-of-the-art results on the link prediction task. However, a common theme impeding the practicality and deployment of SGRLs is the lack of its scalability to large-scale graphs. We dedicate the following sections to elaborate more on SGRLs with a focus on showcasing its strengths (capturing outstanding link representations) and weaknesses (scalability issues).

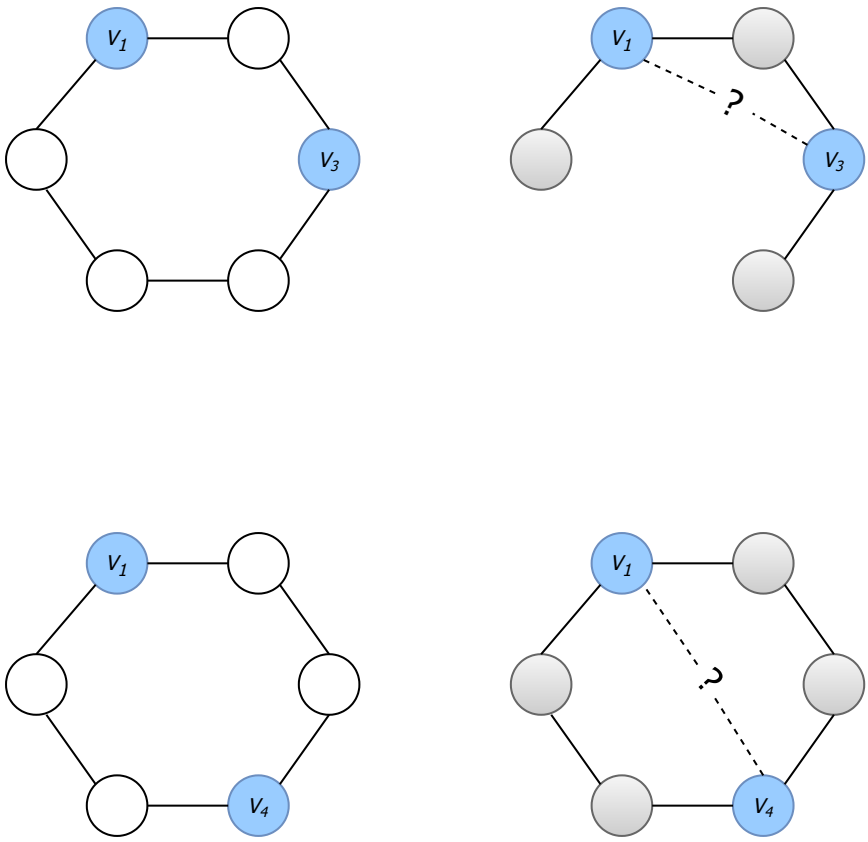


Figure 2.4: This figure is best viewed in color. How SGRLs overcome the problem posed in Figure 2.3. On the left we capture the original graphs with pairs of nodes $(V1, V3)$ and $(V1, V4)$ highlighted, for which we wish to see if links would be formed. SGRLs effectively extract the subgraph involving the target nodes (shown on the right column), and run GNNs on this extracted subgraph. In this example, we extract the 1-hop subgraphs around $(V1, V3)$ and $(V1, V4)$. Through this process, the learnt representations for $(V1, V3)$ is different from $(V1, V4)$, due to the fact that the enclosing subgraphs capture $V1$'s role in forming links with $V3$ and $V4$ uniquely.

2.2.4 Subgraph Graph Neural Networks

Subgraph representation learning approaches (SGRLs) [103, 104, 107, 95, 57, 78] treat link prediction as a binary graph classification problem on enclosing subgraph $G_{uv} = (V_{uv} \subseteq V, E_{uv} \subseteq E)$ around a target $\{u, v\}$. We can define the subgraph enclosing target pair of nodes $\{u, v\}$ as follows,

Definition 1 (Enclosing Subgraph [104]) *Given a graph G, the h-hop enclosing*

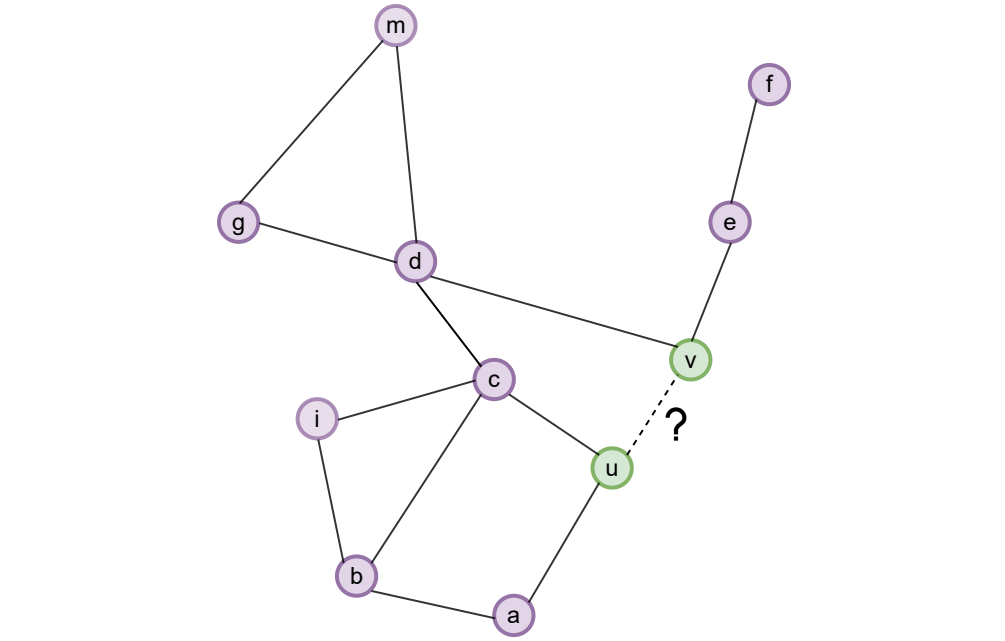
subgraph around target nodes (u,v) is the subgraph G^h_{uv} induced from G with the set of nodes $\{j \| d(j,x) \leq h$ or $d(j,y) \leq h\}$, where $d(i,j)$ is the geodesic distance between node i and j.

SGRLs aim to classify if the enclosing subgraph G^h_{uv} is *closed* or *open* (i.e., the link exists or not).[3] We capture how SGRLs use the enclosing subgraphs around target nodes for link prediction in Figure 2.5. For each G^h_{uv}, SGRLs produce its nodal-representation matrix $\mathbf{Z}_{uv}$ using the stack of convolution-like operators (see Eq. 2.1). Then, similar to Eq. 2.9, the nodal representations would be converted to link probabilities, with the distinction that the pooling function is graph pooling (e.g., SortPooling [105]), operating over all node's representations (not just those of targets) to produce the fixed-size representation of the subgraph. To improve the expressiveness of GNNs on subgraphs, SGRLs augment initial node features with some structural features determined by the targets' positions in the subgraph. These augmented features are known as *node labels* [107]. Node labeling techniques fall into *zero-one* or *geodesic distance-based* schemes [43]. One of the most popular labeling scheme is the double-radius node labeling scheme,

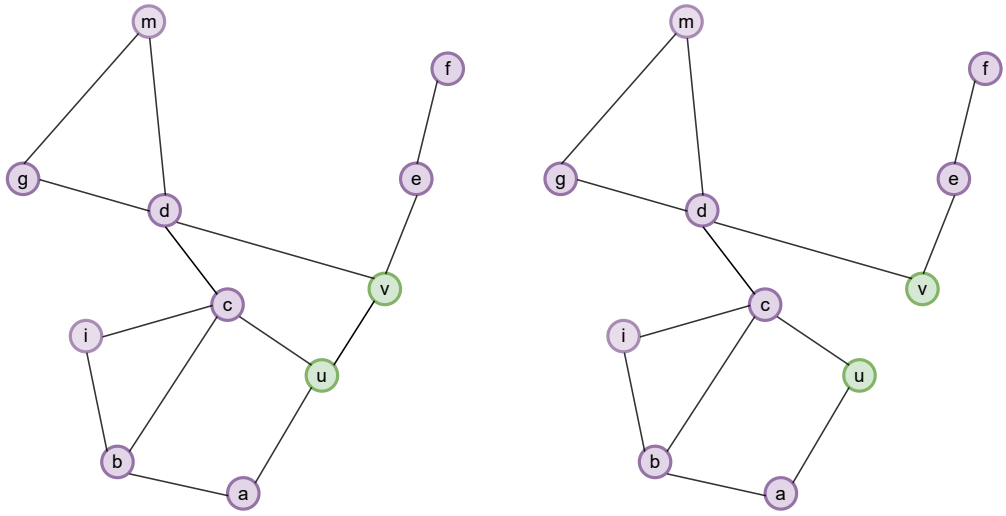
$$DRNL(x, G^h_{uv}) = 1 + min\left(d_{xu}, d_{xv}\right) + \lfloor d'/2 \rfloor \lceil d'/2 - 1 \rceil, \tag{2.11}$$

where, x represents the nodes in the subgraph G^h_{uv}, d_{xu} is the geodesic distance of x to u in G^h_{uv} when node v is removed, and $d' = d_{xu} + d_{xv}$. Note that the distance of x to each target node u is calculated in isolation by removing the other target node v from the subgraph. The target nodes are labeled as 1 and a node with ∞ distance to at least one of the target nodes is given the label 0. Each node label is then represented

[3]We interchangeably use notations G^h_{uv} and G_{uv} in an effort to refer to the enclosing subgraph surrounding the target nodes (u, v). The superscript, h, which denotes the size of the enclosing subgraphs is sometimes dropped for brevity.



(a) Enclosing Subgraph



(b) Closed (c) Open

Figure 2.5: This figure is best viewed in color. As a solution to the link prediction problem, SGRLs extract the enclosing subgraph surrounding the target pairs of nodes and pass it to a GNN, which learns to decide if the enclosing subgraph is closed (i.e., link forms) or open (i.e., link does not form). In this example, Subfigure 2.5a shows the original graph with the target nodes (u, v) for which we want to predict if a link will be formed or not. SGRLs extract the enclosing subgraph surrounding the nodes (u, v) and then treats it as a binary classification problem, i.e., whether the enclosing subgraph of (u, v) will be closed (like Subfigure 2.5b) or open (like Subfigure 2.5c).

by its one-hot encoding, and expands the initial node features, if any.

We refer to $\mathbf{X}_{uv}$ as the matrix containing both the initial node features and labels generated by a valid labeling scheme [107]. SGRLs then use $\mathbf{X}_{uv}$ for the learning process pertaining to each link. The expressiveness power of SGRLs comes with high computational costs due to subgraph extractions, node labeling, and independent operations on overlapping large subgraphs. Note that the subgraph size grows exponentially with the hop size h, and large-degree nodes (e.g., celebrities in a social network) possess very large enclosing subgraphs even for a small h. This computational overhead is exaggerated in denser graphs and deeper subgraphs due to the exponential growth of subgraph sizes. We visualize this bottleneck in Figure 2.6. We focus on addressing this scalability issue of SGRLs in this thesis.

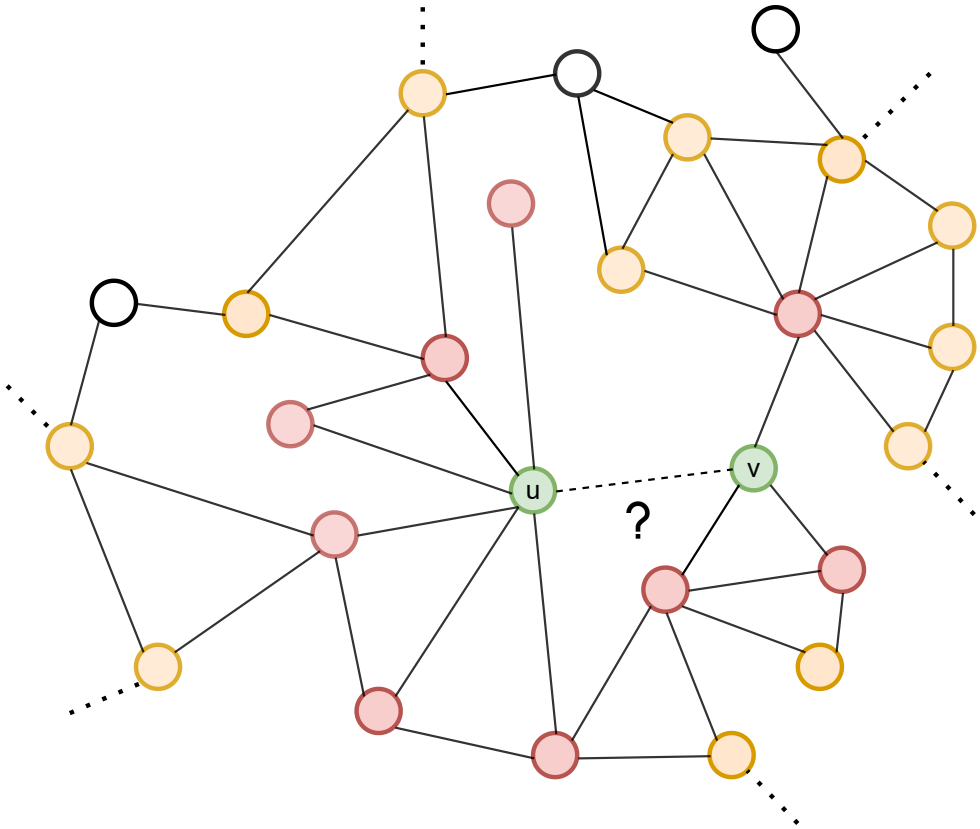


Figure 2.6: This figure is best viewed in color. An example of where extracting the enclosing subgraph surrounding links can be computationally demanding. In this example, the 2-hop enclosing subgraph of (u, v) contains a large number of nodes. One can also visualize how much the subgraph grows when 2-hop subgraphs (highlighted by the yellow nodes) are extracted vs. just one hop subgraphs (highlighted by the red nodes). This is sometimes referred to as the neighborhood explosion problem.

Chapter 3

Related Work

In this chapter we familiarize the reader with the related works in graph representation learning and its applications on the link prediction objective through a thorough literature review. We then focus our discussions on subgraph representation learning methods, the current state-of-the-art class of models for link prediction. We close discussions discussing scalability bottlenecks faced by GNNs and in particular, SGRLs.

3.1 Shallow Embedding Methods

Graph representation learning (GRL) [32] has numerous applications in drug discovery [88], knowledge graph completion [109], and recommender systems [96]. The key downstream tasks in GRL are node classification [33], link prediction [104] and graph classification [105]. The early work in GRL, *shallow encoders* [72, 29], learn dense latent node representations by taking multiple random walks rooted at each node. The earliest work in this category of shallow encoders or shallow embedding methods is DeepWalk [72]. DeepWalk extended concepts and models established in NLP (natural language processing) at the time to the context of networks. DeepWalk proposed

talking multiple random walks from each node in the graph and then utilizing a Skip-Gram [62] model to encode the nodes encountered in the random walk. DeepWalk treats the nodes encountered in each random walk as a "language" for which a latent representation is calculated using the Skip-Gram model. The leaning process is seen as increasing the likelihood of nodes in a random walk to occur together. DeepWalk can be summarized as an unsupervised deep learning approach for capturing rich low dimensional representations of nodes which can be applied to a plethora of independent downstream tasks. The optimization process in DeepWalk utilizes Stochastic Gradient Decent (SGD) [7] with negative sampling [63].

node2vec [29] is an extension to the DeepWalk model aimed at improving the random walk generation at each node. DeepWalk [72] assigns equal importance to both breadth first (BFS) and depth first traversals (DFS) from a node through the use of unbiased uniform random walks. However, there are graph datasets which follow homophily principles [24, 92], where a DFS style of traversals could be preferred over BFS (i.e., a "zoomed-out" macro level view of the neighborhood of the nodes is essential for calculating community structure). The authors of node2vec formulate this balance between preferring DFS and BFS biased random walks with the introduction of the in-out hyperparameter (q) to control the shift of importance between homophily principles (DFS) and structural equivalence (BFS) [37]. node2vec also introduces another notable hyperparameter p in addition to the return parameter q, which captures the importance of revisiting a node already visited. The node2vec model is equivalent to DeepWalk when $p = q = 1$.

Other flavors of random walk style shallow embedding models also exist. Few of the prominent models other than Deepwalk and node2vec are metapath2vec [20] and struc2vec [75]. metapath2vec is a meta bath biased random walk method that is aimed at modeling heterogeneous graphs. This was an improvement over node2vec

and DeepWalk, both of which were designed to only work with homogeneous graphs. One obvious disadvantage of metapath2vec is in the addition of the metapath hyperparameter to influence the random walks by, which needs to be fine tuned for each graph dataset. struc2vec on the other hand is yet another random walk method that is designed to be applicable in niche areas where the focus of the embeddings should be on capturing the structural similarity over other features such as homophily. struc2vec is not proposed as an improvement over other models, but is made to target graphs that require structural information to be captured with more importance over other properties.

Another approach that does not fall under the domain of random walk based models, but one that is used to capture node embeddings is the LINE (Large-scale Information Network Embedding) model [79]. The LINE model captures node embeddings through the use of two embedding objective functions, $O1$ and $O2$. $O1$ has the goal of capturing the node's direct connection information and $O2$ has the goal of capturing the node's structural information on a global level using the 2-hop adjacency matrix $\mathbf{A}^2$. Here, $\mathbf{A}^2$ refers to the resulting matrix from the multiplication of the input graph's adjacency matrix with itself. It is also to be note that $\mathbf{A}^2$ contains the number of 2-length paths between the nodes in the original graph. LINE, despite being conceptually similar to both DeepWalk and node2vec, does not explicitly use random walks for the learning process.

The aforementioned models all fall in the category of shallow node embedding approaches. This is due to the fact that the embeddings produced by these models are equivalent to a lookup in an embedding table. The embeddings for a node either exist or they don't. However, for real world networks, nodes keep getting added to the graph on an ad-hoc basis and more powerful models to capture node embeddings in an inductive manner is essential.

We can summarize the shortcomings of the shallow embeddings approaches as follows,

1. Shallow embedding approaches do not consume the initial nodal features, $\mathbf{X}$, that are part of the graph. This is a significant drawback on datasets such as the Cora, CiteSeer and PubMed datasets [93], where the graphs are presented with rich feature information that can aid in the downstream prediction task.

2. Shallow embedding approaches are inherently *transductive* in nature [33]. Under the transductive learning setting, all the nodes for which we seek to learn the latent representation for needs to be exposed during the training phase. As such, shallow embedding models will not be feasible in an 'online' setting, where nodes are introduced on the fly. This is because for each new node introduced, the whole input graph has to undergo training and the model has to perform optimizations.

These shortcomings led to growing interest in Message Passing Graph Neural Networks (MPGNNs)[1] [8, 19] such as Graph Convolutional Networks (GCN) [50]. In MPGNNs, node feature representations are iteratively updated by first aggregating neighborhood features and then updating them through non-linear transformations. MPGNNs differ in formulations of aggregation and update functions [32]. We dedicate the next section to discuss in detail about MPGNNs.

How Shallow Embedding Models are Related to This Thesis

Shallow embedding approaches like DeepWalk and node2vec are well established unsupervised learning methods. As such, in our work, we use it to create the initial

[1] We use MPGNNs and GNNs interchangeably even though GNNs represent the broader class in GRL.

nodal features for datasets which come with no nodal features. Our *S3GRL* requires initial nodal features in its learning process, and as such, for non-attributed datasets, we first run shallow embedding methods to generate the initial nodal features as a preprocessing step.

3.2 Message Passing Graph Neural Networks

One of the first works that introduced deep neural networks for graphs is the PATCHY-SAN model introduced by Neipert et al. in [66]. PATCHY-SAN was the first of its kind model that extended convolutions to non-Euclidean graph structured data. However, it used a breadth-first style traversal for capturing the receptive field of each node and as the degree of nodes increases, the traversal becomes more computationally demanding. Moreover, PATCHY-SAN processes nodes according to a fixed ranking of the nodes. To overcome the disadvantages posed by PATCHY-SAN, the Graph Convolutional Neural network (GCN) [50] architecture was proposed.

GCN is the primary deep learning architecture that laid the foundation for the class for message passing graph neural network architectures (MPGNNs). Most message passing graph neural network architectures can be formalized with the use of two functions; the aggregate function and the update function [32]. The aggregate function is used to formulate the method of aggregating messages from a node's neighbors and the update function is used to define how the update of a node's vector embedding using the aggregated message information takes place. Multiple flavors of message passing neural networks exist that is based on these two functions. All such flavors of MPGNNs including GCN can be derived by modifying these two fundamental functions. Some examples of such architectures include GraphSAGE, Graph Attention Networks (GAT) [82], Graph Isomorphism Network (GIN) [89] etc.

Even though GCN is the pioneer message passing graph neural network model, it lacked in its ability to be applicable in an inductive learning setting. GraphSAGE [33] was the first MPGNN architecture formulated to work in the inductive learning setting. Moreover, the GraphSAGE model also places emphasis on scaling up of graph neural networks by sampling a subset of the nodes' neighbors at each hop, or, pruning the computational graph originating from each node. One additional contribution from the authors of GraphSAGE is in the flexibility of the aggregation operator's formulation. The flexible definition of GraphSAGE aggregation allows for both permutation invariant aggregation operators like average or sum to be used as well as permutation variant operators like Long Short-Term Memory (LSTM) [36] models to be used.

Both GCN and GraphSAGE are isotropic models in the sense that while calculating node embeddings, equal importance is given to the every node in the node's neighborhood. Graph Attention Networks [82] was the first model to introduce and extend the concept of attention [81] to graph structure data. GAT falls into the category of anisotropic graph neural networks. GAT utilizes either single-headed attention or multi-headed attention in the aggregation step to weight the messages of a node's neighbors differently. As shown in the experimental results of GAT, weighing a nodes' neighbors messages based on attention is seen to improve GAT's results on downstream tasks compared to GraphSAGE and GCN.

GCN, GraphSAGE and GAT models additionally can be categorized as graph neural networks are at most as powerful as the 1-Weisfeiler-Lehman (WL) graph isomorphism test [85]. The 1-WL test is a powerful, iterative graph coloring algorithm used to help determine if two graphs are isomorphic or not. This limitation led to the creation of the graph isomorphism network(GIN) [89], which is a GNN that is as powerful as the 1-WL test. GIN utilizes an MLP in the MPGNN update and

aggregate functions to emulate an injective function that maps unique aggregated representations of a node's neighborhood to non-overlapping (unique) representations. GIN is able to push the expressivity of MPGNNs to be as powerful as the 1-WL test owing to the universal approximation theory of MLPs [39, 38].

Another interesting development in MPGNNs is focusing on creating more expressive GNN architectures. One of the first models in this line of work include the introduction of Identity Aware graph neural networks [97] that extends graph neural networks' ability to be expressive beyond the 1-WL test. ID-GNN uses a heterogeneous message aggregation function; one update function for the focus node and another update function for the other nodes. Additionally, the focus node being processed is colored differently in order to make the distinction in the update functions. IDGNN paved way to a plethora of other works that can be classified under Subgraph GNNs [43, 106, 6, 110]. Authors in [25] were the first to show that Subgraph GNNs are more powerful than the 1-WL test but bounded by the 3-WL test. This line of work in deriving more expressive GNNs are orthogonal to this thesis.

Even with the advancement of node representation learning through more powerful GNNs being devised, MPGNNs performance on link prediction tasks remained subpar. We dedicate the next section to talk about SGRLs, which advanced the representation capability of GNNs on link prediction tasks.

How MPGNNs are Related to This Thesis

MPGNNs have been extensively used in link prediction tasks. However, as mentioned earlier, they have their drawbacks when it comes to learning link representations. We use MPGNNs as a baseline to empirically showcase the weaknesses of different MPGNN architectures compared to SGRLs, including our *S3GRL* model instances.

3.3 Subgraph Graph Neural Networks

Link prediction is a fundamental problem on graphs, where the objective is to compute the likelihood of the presence of a link between a pair of nodes (e.g., users in a social network) [56]. Network heuristics such as common neighbors or Katz Index [47] were the first solutions for link prediction. However, these predefined heuristics fail to generalize well on different graphs. To address this, MPGNNs have been used to learn representations of node pairs independently, then aggregate the representations for link probability prediction [49, 61]. However, as stated earlier, this class of MPGNN solutions falls short in capturing graph automorphism and capturing the different structural roles played by the nodes in forming links [107]. SEAL [104] successfully overcomes this drawback by treating the link prediction problem as a binary classification problem on the subgraph enclosing a link, and through the use of labeling tricks [107] for augmenting the node features. SEAL led to the emergence of subgraph representation learning approaches (SGRLs) [104, 55], which offer state-of-the-art results on the link prediction task. SGRLs [103, 104, 107, 95, 57] treat link prediction as a binary graph classification problem on enclosing subgraph $G_{uv} = (V_{uv} \subseteq V, E_{uv} \subseteq E)$ around a target $\{u, v\}$. SGRLs aim to classify if the enclosing subgraph G_{uv} is *closed* or *open* (i.e., the link exists or not). However, as stated in earlier sections, the expressiveness of SGRLs come at the cost of expensive preprocessing involving subgraph extractions and labeling.

SEAL (Learning from Subgraphs, Embeddings, Attributes for Link prediction) [104] is a foundational paper in link prediction for graphs. The authors are motivated by two flaws in existing link prediction models. The first one being that specific heuristic driven approaches are not a universal nor feasible solution for varying graph datasets. Secondly, higher order heuristics, such as PageRank, can be very compute

heavy while running on medium to large sized graphs. To this end, the authors propose a $\gamma - decaying\ heuristic$. The authors show that any higher order heuristics (such as PageRank) can be approximated with just a localized subgraph view around the nodes. The larger the induced subgraphs around nodes, exponentially less the errors will be in calculating the heuristics. This lead the authors to the conclusion that localized induced subgraphs around a pair of nodes with a small hop distance is sufficient in learning about their chances of forming links. Moreover, a localized view of the nodes reduces the infiltration of influence from far away nodes as well. In practice, SEAL implements this $\gamma - decaying\ heuristic$ framework with the help of GNNs (e.g., DGCNN [105]). SEAL does this by first extracting the subgraph around links and then feeding into a GNN that is aimed at learning the heuristics associated with the subgraphs.

Since SEAL, multiple works have since been introduced that either improve SEAL or complement it. One of the earliest works in this category is Distance Encoding GNN (DE-GNN) [55]. The idea proposed by authors in (DE-GNN) is to inject distance encoding information along with the node attributes to help GNNs in learning to distinguish otherwise indistinguishable nodes according to the 1-WL test. To this end, the authors propose two versions of DE-GNN, the vanilla version and a more advanced DEA-GNN which uses the distance encoding information inside the aggregation step as a controller. The vanilla version, DE-GNN, injects distance information as one-hot vectors alongside (concatenates) the existing node attribute information of the nodes. The distance information injected can be the shortest path to the other nodes or even the PageRank scores calculated. DEA-GNN on the other hand, utilizes the distance encoding information during the aggregation step. For example, while aggregating node representations for each layer, node representations up to shortest path distance of n can be considered (as opposed to just aggregating for one hop per

layer in GCN) [21]. The DEA-GNN version is at least as powerful as the DEGNN, but, is not guaranteed to always be better as seen in the experiments. Moreover, the two versions can be used together (have distance encoding as feature and also use distance encoding during aggregation step).

Following that, both distance encoding and the DRNL labeling schemes were unified under the labeling scheme framework in [107]. Zhang et.al, studied the different labeling schemes utilized in subgraph representation learning works and identified and established the labeling scheme framework. Labeling scheme framework introduced rules, which if satisfied, could help classify any labeling used to augment initial nodal features as a labeling trick. Other than distance encoding and DRNL, some other labeling tricks that has found success include, the zero-one labeling trick. The zero-one labeling trick is a simple, but powerful labeling trick where 1 is added to nodes that form a link (i.e., the nodes in T), and 0 otherwise. Zero-one labeling trick has been proven to be very powerful in works such as GLASS [84] which focus on subgraph classification [3].

Multiple other subgraph representation learning models also exists. For example, DEAL was proposed as an extension of SEAL to be applicable in an inductive learning setting [34]. On the other hand, Line Graph Neural Networks (LGLP) convert the subgraphs into line graphs before feeding into GNNs to get rid of the graph pooling step in SEAL [10]. Whereas, mLink creates a multi-scale view of the subgraphs using labeling tricks [9]. More recently, WalkPool was introduced which is proposed as a pooling layer that is added to subgraph representation learning frameworks aimed at improving the lack of higher order topological information that is missing during learning in models like SEAL [104]. The authors of WalkPool argue that DRNL labeling scheme is a substitute for adding first order topological information but cannot fully capture higher order information such as motifs. To this end, the authors

create a new pooling layer called WalkPool. WalkPool pooling consumes the latent (sub)graphs that is output from GNNs such as SEAL, VGAE [49] in the SGRL framework. It can easily be seen as a substitute for max, min, mean or SortPooling [105] layers currently utilized in SGRL frameworks.

All of these works discussed above can be grouped together under the general framework of subgraph representation learning approaches (SGRLs), aimed at increasing the expressive power of GNNs for link prediction. In the next sections, we discuss about scalability of GNNs followed by our main focus, scalability of SGRLs.

How SGRLs are Related to This Thesis

SGRLs are the current state-of-the-art class of neural models for the link prediction task. However, SGRLs come with high computational costs. In this thesis, we propose *S3GRL*, which is aimed at relieving the computational overheads that are plaguing SGRLs. As such, we use multiple SGRLs, like SEAL, DE-GNN and WalkPool, as a baseline for comparison in experiments against *S3GRL*. This comparison is done to empirically showcase *S3GRL*'s scalability vs. other state-of-the-art SGRL models.

3.4 Scalability in Graph Neural Networks

Recently, a new research direction has emerged in GNNs on scalable training and inference techniques, which mainly focuses on node classification tasks. A common practice is to use different sampling techniques. Sampling based GNN scalability works can be categorized into node based sampling and graph sampling methods. One of the earliest works that can be classified under the node sampling category is GraphSAGE [33]. The authors proposed pruning the computational graph related to each node by setting an upper limit in the number of neighbors that will be sampled

per hop for a node. Even though GraphSAGE is able to reduce the computational cost associated with training graph neural networks, there still exists the *neighborhood-explosion* problem in GraphSAGE as the number of nodes increases exponentially as the layers increase. To overcome the lack of theoretical guarantees of convergence in GraphSAGE and to overcome the neighborhood explosion problem, Chen et al. proposed a novel *importance sampling* based on the degree of nodes in the graph (as opposed to the neighbors of a node) to sample nodes per convolution layer [13] independent of the connections between them.[2] To overcome the limitations of FastGCN, Zou et al. [115] proposed a layer-dependent sampling of nodes called LADIES.[3] Following node sampling based techniques, Chiang et al. [16] proposed a graph clustering based mechanism, called ClusterGCN, for mini-batch training of nodes in graph neural networks. Cluster-GCN falls into the category of graph sampling methods. The authors in Cluster-GCN proposed to apply a graph clustering algorithm to construct mini-batches of nodes to speed-up the training process. To improve ClusterGCN, Zeng et al. proposed GraphSAINT [102] which introduced two regularization parameters in the training phase aimed at reducing the variance and bias across clusters. Apart from that, GraphSAINT also introduced an edge-sampling algorithm, random walk based algorithm and a uniform node sampling algorithm to construct induced subgraphs for forming clusters/mini-batches. However, all the sampling based models focus on the task of node classification and not link prediction.

Apart from node and graph sampling based approaches, there also exists works like VR-GCN [14] and GNNAutoScale [23] which focus on utilizing historical embeddings of nodes to improve the training speeds. Another interesting line of work is in using

[2]It can be argued that FastGCN is a *layer sampling* technique as opposed to a node sampling technique. This is because n nodes are sampled per layer agnostic of their connections to the previous and following layers.

[3]LADIES, similar to FastGCN can be argued to be a *layer sampling* technique as well.

techniques from knowledge distillation [35] to train a student MLP to accelerate the inference speedup [108]. This line of research is orthogonal to the models and methods researched so far with respect to scaling up GNNs and falls under the category of *inference acceleration* methods. Regardless of the scalability models discussed so far, and others [46, 98, 100, 101] that exist, the primary focus of these works are on node classification and not on the link prediction objective.

3.4.1 Scalability by Simplification

Other scalability efforts in GNNs utilize simplification techniques, such as removing intermediate non-linearities [87, 26], to speed up the training and inference. S-GCN [87] and SIGN [26] fall under this category of models. S-GCN was the pioneering model under the simplification of GNN category. The authors in S-GCN were motivated by questioning why GNNs started out with complicated approaches, involving multiple nested convolutions, in place of simpler linear learning components. To this end, the authors study the impact of the non-linearity function in the GCN formulation (σ in Eq. 2.1), and propose completely dropping it. What results in dropping the non-linearity in Eq. 2.1 is a collapse of the multiple nested convolutions, and layer-wise weight multiplications. However, as there is no longer the non-linearity function in between each layer's weight update, the authors proposed replacing the individual weights with a single weight matrix instead. This is the S-GCN model. The S-GCN model demonstrated to be very scalable and easier for training and inference owing to its singular linear weight. More interestingly, the updated formulation also allowed for easy precomputation, which involved simple matrix multiplications which could be performed even before training, leading to even more overall speed-up.

An extension of S-GCN is SIGN [26]. In SIGN, the authors proposed using multiple operators in place of the singular diffusion operator utilized in S-GCN. The

authors are motivated by the inception module in CNNs [77], and proposed using multiple operators which, similar to S-GCN, can be easily pre-computed and lead to better generalization on graph data, all the while boosting scalability. Essentially, SIGN becomes S-GCN when a single adjacency operator operator is utilized in its framework. We mathematically detail both S-GCN and SIGN models in Chapter 4.

Scalability by simplification approaches have been shown to speed up the training of MPGNNs (at the global level) for node and graph classification tasks, and even for link prediction [73]; however, they are not directly applicable to SGRLs which exhibit superior performance for link prediction. In this work, we take the scalability-by-simplification approach for SGRLs (rather than for MPGNNs) by introducing *Scalable Simplified SGRL* (*S3GRL*). Allowing diverse definitions of subgraphs and subgraph-level diffusion operators for SGRLs, *S3GRL* emulates SGRLs in generalization while speeding them up.

3.4.2 Scalability of SGRLs

There is a growing interest in the scalability of SGRLs [95, 57, 12], with a main emphasis on the sampling approaches. SUREL [95, 94], by deploying random walks with restart, approximates subgraph structures; however, it does not place its focus on using GNNs for link prediction. In SUREL, the subgraph extraction is replaced by M-many m-length random walk sequences from the nodes given. Replacing subgraph extraction with random walks helps reduce the preprocessing time required for training. To account for the loss of information, the authors also propose Relational Positional Encoding (RPE) for the nodes in the walks. The walk sequence along with the RPE vectors of the query nodes form the sequence that is provided to a neural network for learning. The neural network can be an RNN, CNN, transformer or a GNN. To further improve the SUREL model, the authors also propose methods for

the fast storage and access of the random walk sequences and the RPE vectors. The authors also come up with a fast approach for combining the walk data of multiple nodes in the query nodes. Finally, the authors also propose a way to train mini-batches of queries using a breadth first search originating from a subset of the query nodes.

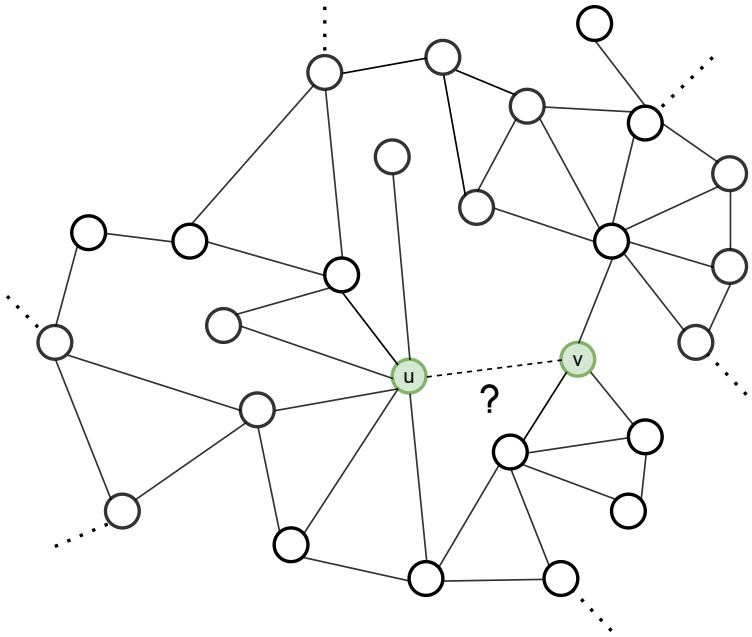
ScaLed [57] uses similar sampling techniques of SUREL, but to sparsify subgraphs in SGRLs for better scalability. ScaLed accomplishes this by taking multiple random walks with restarts from the target links. We capture how ScaLed's random walk based enclosing subgraph helps create sparsified enclosing subgraphs in comparison to SEAL in Figure 3.1. Moreover, the ScaLed model can use any labeling trick (e.g., DRNL, zero-one labeling, etc.) [107] to encode the distances between target nodes and other nodes in the sampled subgraphs. Similar to SEAL, the distance labels along with the nodal features (if any) of the nodes in the sampled subgraph are fed into a graph neural network with graph pooling operation (e.g., DGCNN with SortPooling operation [105]) for the classification task. The ScaLed model offers easy plug-and-play modularity into most graph neural networks (e.g., GCN [50], GIN [89], GraphSAGE [33], DGCNN [105], etc.), and can also be used alongside any regularization technique or loss function.

Another notable work in this category is ELPH/BUDDY [12]. ELPH/BUDDY uses subgraph sketches as messages in MPGNNs but does not learn link representations explicitly at a subgraph level. Our work complements this body of research. Our framework can be combined with (or host) these methods for better scalability (see our experiments for an example). Finally, another notable mention is the SSNP [44] model. The SSNP model uses subgraph neighborhood in the pooling layer to enhance the expressiveness of the representations used for subgraph classification. SSNP operates on the base graph and thus does not require any subgraph extractions.

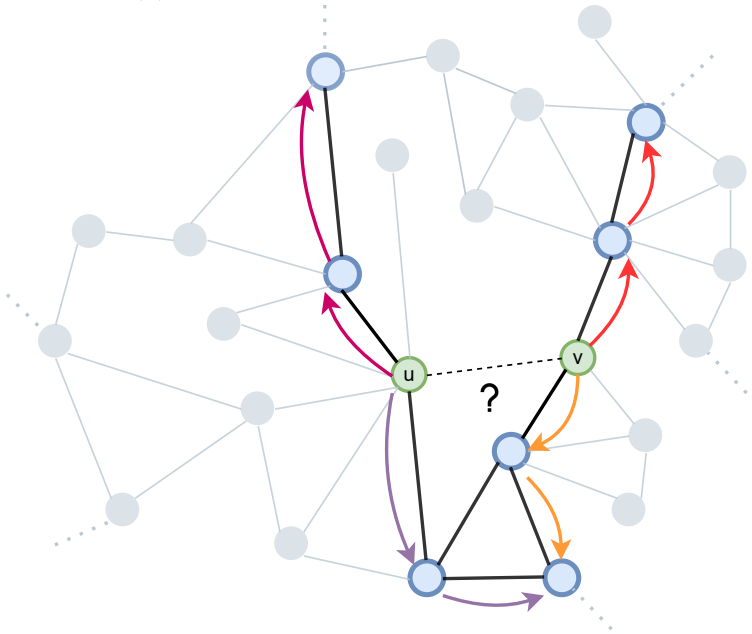
However, SSNP is aimed at the subgraph classification task and not link prediction.

How Scalability of SGRLs are Related to This Thesis

Multiple scalability efforts have been made for GNNs. However, the bulk of these efforts have been targeted towards node and graph classification tasks. SGRLs are the premier class of models in link prediction. Although SGRLs offer state-of-the-art results in link prediction, little work has been done in scalability of SGRLs. Our *S3GRL* framework is aimed at overcoming the computational bottlenecks faced by SGRLs while offering limited to no loss of link representation learning expressivity.



(a) The original dense subgraph enclosing (u, v).



(b) The sparsified subgraph enclosing (u, v) found upon applying the ScaLed model.

Figure 3.1: This figure is best viewed in color. An example of how the ScaLed model sparsifies dense enclosing subgraphs. In this example, Subfigure 3.1a captures the dense enclosing subgraph around the target nodes (u, v). ScaLed, by taking the subgraph induced by the random walks originating from (u, v) (2 random walks of length 2 in this example), is able to sparsify the subgraph resulting in faster training and inference times for SGRLs.

Chapter 4

Approach

4.1 Preliminaries

Consider a graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ is the set of nodes (e.g., individuals, proteins), $E \subseteq V \times V$ represents their relationships (e.g., friendship or protein-to-protein interactions). We also let $\mathbf{A} \in \mathbb{R}^{n \times n}$ represent the *adjacency matrix* of G, where $A_{uv} \neq 0$ if and only if $(u, v) \in E$. Additionally, the graph G can be directed or undirected. If G is directed, $A_{uv} \neq 0$ does not imply $A_{vu} \neq 0$. On the other hand, if G is undirected $A_{uv} = A_{vu}$. We further assume each node possesses a d-dimensional feature vector (e.g., user's profile, features of proteins, etc.) stored as a row in *feature matrix* $\mathbf{X} \in \mathbb{R}^{n \times d}$.

4.2 Problem Statement

Link Prediction. The goal in link prediction is to infer the presence or absence of an edge between *target nodes* $T = \{u, v\}$ given the (partially) observed matrices $\mathbf{A}$ and $\mathbf{X}$. The learning problem is to find a *likelihood function* f such that it assigns a

higher likelihood to those target nodes with a missing link. The likelihood function can be formulated by various neural network architectures (e.g., MLP [30] or GNNs [18]).

4.3 Scalability by Simplification

To improve the scalability of GNNs for node classification tasks, several attempts are made to simplify GNNs by removing their intermediate nonlinearities, thus making them shallower, easier to train, and scalable. Simplified GCN (SGCN) [87] has removed all but the last non-linearities in L-layer GCNs, to predict class label probabilities $\mathbf{Y}$ for all nodes by

$$\mathbf{Y} = \xi(\hat{\mathbf{A}}^L\mathbf{X}\mathbf{W}), \tag{4.1}$$

where, ξ is softmax or logistic function, and $\mathbf{W}$ is the only learnable weight matrix. The term $\hat{\mathbf{A}}^L\mathbf{X}$ can be precomputed once before training. Benefiting from this precomputation and a shallower architecture, SGCN improves scalability. SIGN has extended SGCN to be more expressive yet scalable for node classification tasks. Its crux is to deploy a set of linear diffusion matrices $\mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(r)}$ that can be applied to node-feature matrix $\mathbf{X}$. The class probabilities $\mathbf{Y}$ are computed by,

$$\mathbf{Y} = \xi(\mathbf{Z}\mathbf{W}') \tag{4.2}$$

where,

$$\mathbf{Z} = \sigma\left(\bigoplus_{i=0}^{r}\mathbf{M}^{(i)}\mathbf{X}\mathbf{W}_i\right). \tag{4.3}$$

Here, ξ and σ are non-linearities, $\mathbf{W}_i$ is the learnable weight matrix for diffusion oper-

ator $\mathbf{M}^{(i)}$, $\bigoplus$ is a concatenation operation, and $\mathbf{W}'$ is the learnable weight matrix for transforming node representations to class probabilities. Letting $\mathbf{I}$ be the identity matrix, $\mathbf{M}^{(0)} = \mathbf{I}$ in Eq. 4.3 allows the node features to contribute directly (independent of graph structure) into the node representation and consequently to the class probabilities. The $\mathbf{M}^{(i)}$ operator matrices help capture different heuristics in the graph. For example, the powers of the adjacency matrix as operators capture the number of walks between nodes. As with SGCN, the terms $\mathbf{M}^{(i)}\mathbf{X}$ in SIGN can be once precomputed before training. These precomputations and the shallow architecture of SIGN lead to substantial speedup during training and inference with limited compromise on node classification efficacy. Motivated by these efficiencies, our *S3GRL* framework extends SIGN for link prediction on subgraphs while addressing the computational bottleneck of SGRLs.

4.4 Proposed *S3GRL* Framework

We propose *Scalable Simplified SGRL* (*S3GRL*), which benefits from the expressiveness power of SGRLs while offering the simplicity and scalability of SIGN and SGCN for link prediction. Our framework leads to a multi-fold speedup in both training and inference of SGRL methods while maintaining or boosting their state-of-the-art performance (see experiments below).

Our *S3GRL* framework consists of two key components: (i) *Subgraph sampling strategy* $\Psi(G, T)$ takes as an input the graph G and target pairs $T = \{u, v\}$ and outputs the adjacency matrix $\mathbf{A}_{uv}$ of the enclosing subgraphs G_{uv} around the targets. The subgraph sampling strategy Ψ can capture various subgraph definitions such as h-hop enclosing subgraphs [104]), random-walk-sampled subgraphs [95, 57], and heuristic-based subgraphs [100]; (ii) *Diffusion operator* $\Phi(\mathbf{A}_{uv})$ takes the subgraph

adjacency matrix $\mathbf{A}_{uv}$ and outputs its diffusion matrix $\mathbf{M}_{uv}$. A different class of diffusion operators are available: adjacency/Laplacian operators to capture connectivity, triangle/motif-based operators [28] to capture inherent community structure, personalized pagerank-based (PPR) operators [27] to identify important connections. Each of these operators and their powers can constitute the different diffusion operators in *S3GRL*.

In our *S3GRL* framework, each model is characterized by the *sampling-operator* set $\mathcal{S} = \{(\Psi_i, \Phi_i)\}_{i=0}^{r}$, where Ψ_i and Φ_i are the i^{th} subgraph sampling strategy and diffusion operator, respectively. For a graph G, target pair $T = \{u, v\}$, and sampling-operator pair (Ψ_i, Φ_i), one can find T's sampled subgraph $\mathbf{A}_{uv}^{(i)} = \Psi_i(G, T)$ and its corresponding diffusion matrix $\mathbf{M}_{uv}^{(i)} = \Phi_i(\mathbf{A}_{uv}^{(i)})$. For instance, one can define Ψ_i to sample the random-walk-induced subgraph $\mathbf{A}_{uv}^{(i)}$ rooted at $\{u, v\}$ in G. Then, Φ_i can compute the l-th power of $\mathbf{A}_{uv}^{(i)}$, to count the number of l-length walks on the subgraph. *S3GRL* computes the operator-level node representations of the selected subgraph $\mathbf{A}_{uv}^{(i)}$ by

$$\mathbf{Z}_{uv}^{(i)} = \mathbf{M}_{uv}^{(i)} \mathbf{X}_{uv}^{(i)}. \tag{4.4}$$

Here, $\mathbf{X}_{uv}^{(i)}$ is the node feature matrix for nodes in the selected subgraph $\mathbf{A}_{uv}^{(i)}$ for the sampling-operator pair (Ψ_i, Φ_i). Eq. 4.4 can be viewed as feature smoothing where the diffusion matrix $\mathbf{M}_{uv}^{(i)}$ is applied over node features $\mathbf{X}_{uv}^{(i)}$. *S3GRL* then concatenates the operator-level nodal-representation matrix $\mathbf{Z}_{uv}^{(i)}$ of all sampling-operator pairs to form the *joint nodal-representation matrix*:

$$\mathbf{Z}_{uv} = \bigoplus_{i=0}^{r} \mathbf{Z}_{uv}^{(i)} \tag{4.5}$$

The concatenation between nodal-representation matrices with dimensionality mismatch should be done with care: the corresponding rows (belonging to the same

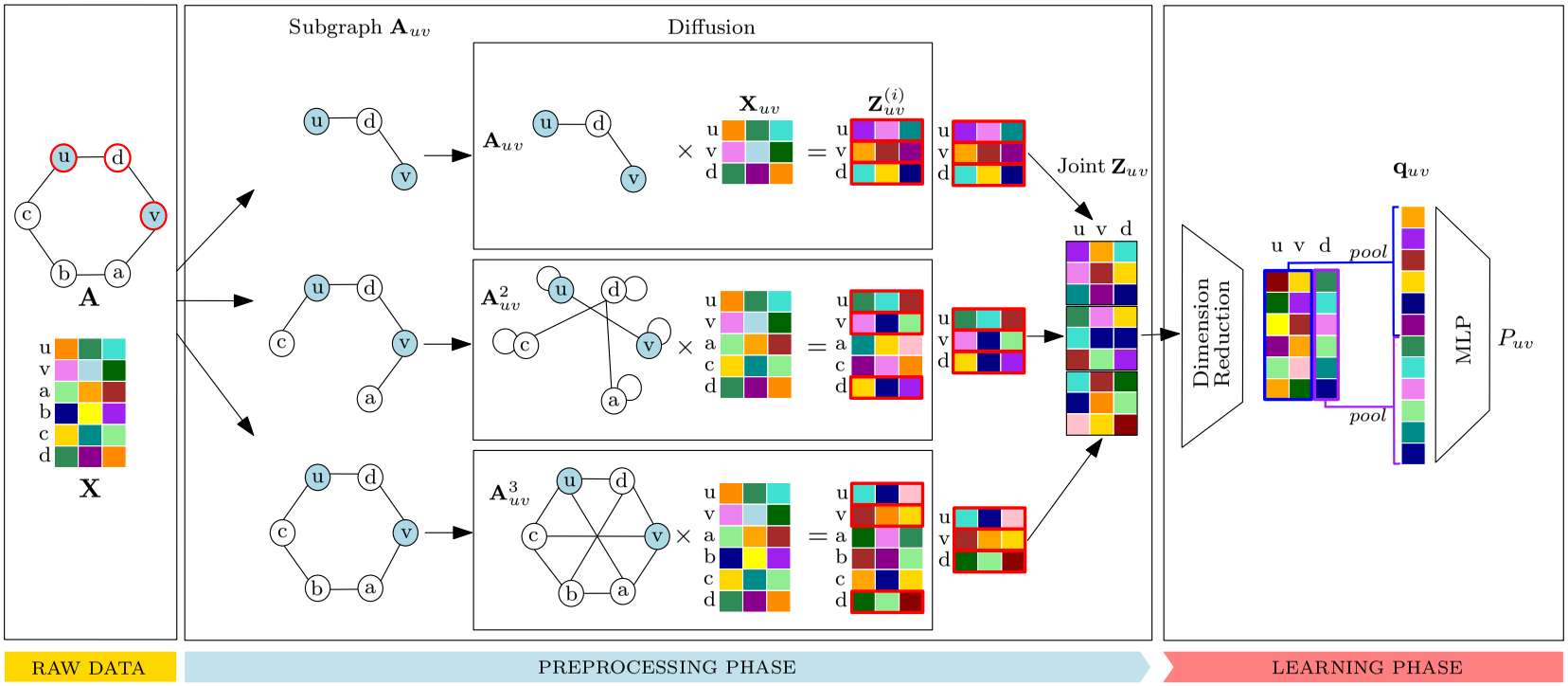


Figure 4.1: This figure is best viewed in color. Our *S3GRL* framework: In the preprocessing phase (shown by the shaded blue arrow), first multiple subgraphs are extracted around the target nodes u and v (shaded in blue) by various sampling strategies. Diffusion matrices are then created from extracted subgraph adjacency matrices by predefined diffusion operators (e.g., powers of subgraphs in this figure). Each diffusion process involves the application of the subgraph diffusion matrix on its nodal features to create the matrix $\mathbf{Z}^{(i)}_{uv}$. The operator-level node representations of selected nodes (with a red border in raw data) are then aggregated for all subgraphs to form the joint $\mathbf{Z}_{uv}$ matrix. The selected nodes in this example are the target nodes $\{u, v\}$, and their common neighbor d. In the learning phase (as shown by the shaded red arrow), the joint matrix $\mathbf{Z}_{uv}$ undergoes dimensionality reduction followed by pooling using center pooling (highlighted by blue-border box) and common neighbor pooling (highlighted by purple-border box). Finally, the target representation $\mathbf{q}_{uv}$ is transformed by an MLP to a link probability P_{uv}.

node) should be inline, where missing rows are filled with zeros (analogous to zero-padding for graph pooling). The joint nodal-representation matrix $\mathbf{Z}_{uv}$ goes through a non-linear feature transformation (for dimensionality reduction) by learnable weight matrix $\mathbf{W}$ and non-linearity σ. This transformed matrix is then further downsampled by the graph pooling pool to form the target's representation:

$$\mathbf{q}_{uv} = \text{pool}\left(\sigma\left(\mathbf{Z}_{uv}\mathbf{W}\right)\right), \tag{4.6}$$

from which the link probability is computed by

$$P_{uv} = \Omega(\mathbf{q}_{uv}), \tag{4.7}$$

with Ω being a learnable non-linear function (e.g., MLP) to convert the target representation $\mathbf{q}_{uv}$ into a link probability.

Our *S3GRL* exhibits substantial speedup in inference and training through precomputing $\mathbf{Z}_{uv}$ (using the target links provided in the input dataset) in comparison to more computationally-intensive SGRLs [55, 69], designed based on multi-layered GCN or DGCNN [105] (see our experiments for details). Existing SGRLs utilize a GCN or GCN-like architecture (e.g., DGCNN, GAT [82] etc.) during the training and inference phase. This results in multiple layers of convolutions and weights associated with each convolution layer, for each extracted subgraph around the target link, per epoch of training (see. Eq. 2.1). In contrast, our *S3GRL* utilizes the precomputed $\mathbf{Z}_{uv}$ and a single linear weight matrix W during training and inference. Note that only Equations 4.6 and 4.7 are utilized in training and inference in *S3GRL* (see also Figure 4.1). Apart from this computational speedup, *S3GRL* offers other advantages analogous to other prominent GNNs. We dedicate the following subsections to details these advantages.

4.4.1 Disentanglement of Data and Model.

The composition of subgraph sampling strategy Ψ and diffusion operator Φ facilitates the disentanglement (or decoupling) of data (i.e., subgraph) and model (i.e., diffusion operator). This disentanglement resembles shaDow-GNN [100], in which the depth of GNNs (i.e., its number of layers) is decoupled from the depth of enclosing subgraphs. Similarly, in *S3GRL*, one can explore high-order diffusion operators (e.g., adjacency matrix to a high power) in constrained enclosing subgraphs (e.g., the ego network consisting of the target link and its h-hop neighborhood). This combination simulates multiple message-passing updates between the nodes in the local subgraph, thus achieving local oversmoothing [100] and ensuring the final subgraph representation possesses information from all the nodes in the local subgraph.

4.4.2 Multi-View Representation.

Our *S3GRL* framework via the sampling-operator pairs provides multiple views of the enclosing neighborhood of each target pair. This capability allows hosting models analogous to multi-view [1, 9] models.

4.5 Our Proposed Instances

In this section, we introduce two instances of our *S3GRL* framework, differentiating in the choice of sampling-operator pairs.

4.5.1 Powers of Subgraphs (PoS)

This instance intends to mimic a group of SGRLs with various model depths on a fixed-sized enclosing subgraph while boosting scalability and generalization (e.g.,

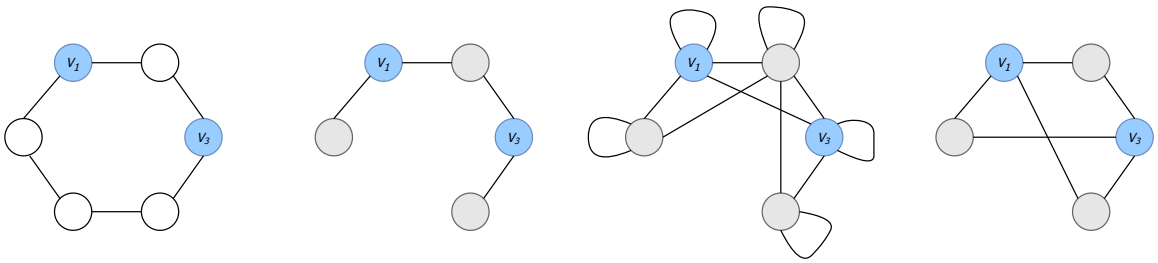


Figure 4.2: Example of PoS operators created with $h = 2$, $r = 3$. The left-most subgraph shows the original enclosing subgraph for the target nodes $(V1, V3)$. Towards its right, we capture the $r = 3$ PoS operators, which help capture the different connectivity around $(V1, V3)$, without involving explicit subgraph extractions. For the creation of the PoS operators, first, the 1-hop enclosing subgraph around the target link $(V1, V2)$ is extracted. Following this, the powers of the extracted subgraph are calculated, which then forms the PoS operators.

SEAL [104]). The sampling-operator set $\mathcal{S} = \{(\Psi_i, \Phi_i)\}_{i=0}^{r}$ is defined as follows: (i) the sampling strategy $\Psi_i(G, T)$ for any i is constant and returns the adjacency matrix $\mathbf{A}_{uv}$ of the h-hop enclosing subgraph G_{uv}^h about target $T = \{u, v\}$. The subgraph G_{uv}^h is a node-induced subgraph of G with the node set $V_{uv}^h = \{j | d(j, u) \leq h \text{ or } d(j, v) \leq h\}$, including the nodes with the geodesic distance of at most h from either of the target pairs [104]; (ii) the i-th diffusion operator $\Phi_i(\mathbf{A}) = \mathbf{A}^i$ is the i-th power of adjacency matrix $\mathbf{A}$. This operator facilitates information diffusion among nodes i-length paths apart. Putting (i) and (ii) together, one can derive $\mathbf{M}_{uv}^{(i)} = \mathbf{A}_{uv}^i$ for Eq. 4.4. Note that $\mathbf{M}_{uv}^{(0)} = \mathbf{I}$ allows the node features to contribute directly (independent of graph structure) to subgraph representation. PoS possesses two hyperparameters r and h, controlling the number of diffusion operators and the number of hops in enclosing subgraphs, respectively. We capture an example of the operators created by PoS with $h = 2, r = 3$ in Figure 4.2.

One can intuitively view PoS as equivalent to SEAL that uses a GNN model of depth r with "skip-connections" [90] on the h-hop enclosing subgraphs. The skip-connections property allows consuming all the intermediate learned representations

of each layer to construct the final link representation. Similarly, PoS uses varying i-th power diffusion operators combined with the concatenation operator in Eq. 4.6 to generate the representation $\mathbf{q}_{uv}$. In this light, the two hyperparameters of r and h (resp.) in PoS control the (virtual) depth of the model and the number of hops in enclosing subgraphs (resp.).

4.5.2 Subgraphs of Powers (SoP)

This instance of *S3GRL*, by transforming the global input graph G, brings long-range interactions into the local enclosing subgraphs. Let $\Psi_H(G, T, h)$ be *h-hop sampling strategy*, returning the enclosing h-hop subgraph about the target pair $T = \{u, v\}$. The SoP model defines $\mathcal{S} = \{(\Psi_i, \Phi_i)\}_{i=0}^{r}$ with $\Psi_i(G, T) = \Psi_H(G^i, T, h)$ and $\Phi_i(\mathbf{A}_{uv}) = \mathbf{A}_{uv}$. Here, G^i is the i-th power of the input graph (computed by $\mathbf{A}^i$), in which two nodes are adjacent when their geodesic distance in G is at most i. In this sense, the power of graphs brings long-range interactions to local neighborhoods at the cost of neighborhood densification. However, as the diffusion operator is an identity function, SoP prevents overarching to the further-away information of indirect neighbors. SoP consists of two hyperparameters r and h that control the number of diffusion operators and hops in local subgraphs, respectively.

4.5.3 Comparing our *S3GRL* instances: PoS vs. SoP.

PoS and SoP are similar in capturing information diffusion among nodes (at most) r-hop apart. But, their key distinction is whether long-range diffusion occurs in the global level of the input graph (for SoP) or the local level of the subgraph (for PoS). SoP is not a "typical" SGRL, however, it still uses subgraphs around the target pair on the power of graphs. We also refer the reader to the experiments section

to help compare and contrast the efficacy and computational requirements of our PoS vs. SoP. The experiments are designed to help guide any user in selecting the appropriate *S3GRL* instance, keeping in mind the efficacy-compute trade-offs for a target dataset.

4.5.4 Subgraph Pooling

Our proposed instances of *S3GRL* are completed by defining the graph pooling function in Eq. 4.6. We specifically consider computationally-light pooling functions, which focus on pooling the targets' or their direct neighbors' learned representations.[1] We consider simple *center* pooling,

$$\text{pool}_C(\mathbf{Z}) = \mathbf{z}_u \odot \mathbf{z}_v, \tag{4.8}$$

where $\odot$ is the Hadamard product, and $\mathbf{z}_u$, $\mathbf{z}_v$ are u's and v's learned representations in nodal-representation matrix $\mathbf{Z}$ (i.e., the two rows of the target pairs in $\mathbf{Z}$). We also introduce *common-neighbor* pooling,

$$\text{pool}_N(\mathbf{Z}) = AGG(\{\mathbf{z}_i | i \in N_u \cap N_v\}), \tag{4.9}$$

where AGG is any invariant graph readout function (e.g., mean, max, or sum), and N_u and N_v are the direct neighborhood of targets u, v in the original input graph. We also define *center-common-neighbor* pooling,

$$\text{pool}_{CCN} = \text{pool}_C \oplus \text{pool}_N \tag{4.10}$$

[1] Our focus is backed up by recent empirical findings [12] showing the effectiveness of the target node's embeddings and the decline in the informativeness of node embeddings as the nodes get farther away from targets.

In addition to their efficacy, these pooling functions allow one to further optimize the data storage and computations. As the locations of *pooled nodes* (e.g., target pairs and/or their direct neighbors) for these pooling functions are specified in the original graph, one could just conduct necessary (pre)computations and store those data affecting the learned representation of pooled nodes, while avoiding unnecessary computations and data storage. Figure 4.1 shows this optimization through red-bordered boxes of rows for each $\mathbf{Z}_{uv}^{(i)}$. This information is the only required node representations in $\mathbf{Z}_{uv}^{(i)}$ utilized in the downstream pool operation. This pruning of $\mathbf{Z}_{uv}^{(i)}$ to just include the target link nodes and the common neighbors results in magnitudes less storage requirements when compared to a traditional SGRL which stores all nodes for each extracted subgraph. We capture this reduction in dataset sizes and the memory storage requirements of our *S3GRL* achieved through our optimizations empirically in our experiments below. We consider center pooling as the default pooling for PoS and SoP, and we refer to them as PoS$^+$ and SoP$^+$ when center-common-neighbor pooling is deployed. We study both center and center-common-neighbor pooling for PoS. However, due to the explosion of h-hop subgraph sizes in SoP$^+$, owing to the global reach of nodes, we limit our experiments on SoP to only center pooling.

4.5.5 Inference Time Complexity.

Let p be the number of pooled nodes (e.g., $p = 2$ for center pooling), d be the dimension of initial input features, r be the number of operators, and d' be the reduced dimensionality in Eq. 4.6. The inference time complexity for any of PoS, SoP, and their variants is $O(rpdd' + d'^2)$. Consider the dimensionality reduction of the joint matrix $\mathbf{Z}_{uv}$ by the weight matrix $\mathbf{W}$ in Eq. 4.6. As $\mathbf{Z}_{uv}$ is rp by d and $\mathbf{W}$ is d by d', their multiplication is in $O(rpdd')$ time. The pooling for any proposed

variants is $O(pd')$. Assuming Ω in Eq. 4.7 being an MLP with one d'-dimensional hidden layer, its computation is in $O(d'^2)$ time.

Chapter 5

Experiments

We carefully design an extensive set of experiments to assess the extent to which our model scales up SGRLs while maintaining their state-of-the-art performance. Our experiments intend to address these questions: **(Q1)** How effective is the *S3GRL* framework compared with the state-of-the-art SGRLs for link prediction methods? **(Q2)** What is the computational gain achieved through *S3GRL* in comparison to SGRLs? **(Q3)** How well does our best-performing model, PoS$^+$, perform on the Open Graph Benchmark datasets for link prediction, graphs with millions of nodes and edges [41, 40]? **(Q4)** How complementary can *S3GRL* be in combination with other scalable SGRLs (e.g., ScaLed [57]) to further boost scalability and generalization?

5.1 Datasets

For our experiments, we use directed and undirected, weighted and unweighted, attributed and non-attributed datasets that are publicly available and commonly used in other link prediction studies [104, 105, 55, 69, 57, 12]. Table 5.1 shows the statistics of our datasets. We divide our datasets into three categories; non-attributed

	Dataset	# Nodes	# Edges	Avg. Deg.	# Feat.
Non-attr.	NS	1,461	2,742	3.75	NA
	Power	4,941	6,594	2.67	NA
	Yeast	2,375	11,693	9.85	NA
	PB	1,222	16,714	27.36	NA
Attributed	Cora	2,708	4,488	3.31	1,433
	CiteSeer	3,327	3,870	2.33	3,703
	PubMed	19,717	37,676	3.82	500
	Texas	183	143	1.56	1,703
	Wisconsin	251	197	1.57	1,703
OGB	Collab	235,868	1,285,465	8.2	128
	DDI	4,267	1,334,889	500.5	NA
	Vessel	3,538,495	5,345,897	2.4	3
	PPA	576,289	30,326,273	73.7	58
	Citation2	2,927,963	30,561,187	20.7	128

Table 5.1: The statistics of the non-attributed, attributed, and OGB datasets.

datasets, attributed datasets and the OGB datasets. The edges in the attributed and non-attributed dataset categories are randomly split into 85% training, 5% validation, and 10% testing sets, except for Cora, CiteSeer and PubMed datasets chosen for comparison in Table 5.8, where we follow the experimental setup in [12] with a random split of 70% training, 10% validation, and 20% testing sets. We note that there are two datasets splits considered for Cora, PubMed and CiteSeer. This is done to ensure a fair comparison against the baselines considered in Table 5.3 and the baselines considered in Table 5.8, and showcase how well our *S3GRL* models fare against varying dataset splits. For the OGB datasets, we follow the dataset split provided by the OGB team [40].

5.2 Experimental Setup

In this section we discuss in detail about the experimental setup for our models and the baselines considered for comparison. We also present key experimental results

and analysis of the data.

5.2.1 Evaluation Metrics

For all models, on the attributed and non-attributed datasets, we report the average of the area under the curve (AUC) of the testing data over 10 runs with different fixed random seeds.[1] Similarly, for the experiments against BUDDY on OGB and Planetoid datasets, we report the average of 10 runs (with different fixed random seeds) on the efficacy measures consistent with [12].

In each run, we test a model on the testing data with those parameters with the highest efficacy measure on the validation data. Our code is implemented in PyTorch Geometric [22] and PyTorch [71].[2] To compare the computational efficiency, we report the average training and inference time (over 50 epochs) and the total preprocessing and runtime for a fixed run on the attributed and non-attributed datasets. Finally, to compare the storage requirements and dataset sizes of our models vs. SGRL baselines, we capture the sizes of the prepared datasets of SEAL vs. our *S3GRL* models on the attributed and non-attributed datasets.[3]

5.2.2 Baselines

For attributed and non-attributed datasets, we compare our *S3GRL* models (PoS, PoS$^+$, and SoP) with 15 baselines belonging to five different categories of link prediction models. Our *heuristic* benchmarks include common neighbors (CN), Adamic Adar (AA) [2], and personalized pagerank (PPR). For *message-passing GNNs (MPGNNs)*,

[1] We exclude Average Precision (AP) results due to their known strong correlations with AUC results [69].

[2] Our codes are available at https://github.com/venomouscyanide/S3GRL_OGB. All experiments are run on servers with 50 CPU cores, 377 GB RAM, and 11 GB GTX 1080 Ti GPUs.

[3] We consider SEAL to be a representative model to showcase the dataset size requirements of a typical SGRL. Other SGRLs like DE-GNN, WalkPool etc. can be expected to have similar dataset storage requirements to SEAL.

we select GCN [50], GraphSAGE [33] and GAT [81]. Our *latent factor (LF)* benchmarks are node2vec [29] and Matrix Factorization [51]. The *autoencoder (AE)* methods include GAE & VGAE [49], adversarially regularized variational graph autoencoder (ARVGA) [70] and Graph InfoClust (GIC) [61]. Finally, our examined SGRLs include SEAL [104], GCN+DE (distance encoding [55]), and WalkPool [69]. For OGB datasets, we compare against BUDDY [12] and its baselines which include a subset of our baselines outlined above and GraphSAGE [33].

5.2.3 Hyperparameter Settings

All hyperparameters of baselines are optimally selected based on their original papers or the shared public implementations. When possible, we also select the *S3GRL* hyperparameters to match the benchmarks' hyperparameters for a fair comparison. We dedicate the following paragraphs to discuss important hyperparameters related to our work and the baselines used in detail.

For SGRL and *S3GRL* methods, we set the number of hops $h = 2$ for the non-attributed datasets (except WalkPool on the Power dataset with $h = 3$), $h = 3$ on attributed datasets (except for WalkPool with $h = 2$ based on [69]), and $h = 1$ for OGB datasets. In *S3GRL* models we set the number of operators $r = 3$ for all datasets except ogbl-collab, ogbl-citation2 and ogbl-ppa with $r = 1$. Across all models, we use a zero-one labeling scheme, except for ogbl-citation2 and ogbl-ppa, where DRNL is used instead. The *AGG* graph readout function in center-common-neighbor pooling is set to a simple mean aggregation, except for ogbl-vessel, ogbl-citation2 and ogbl-ppa where sum is used instead. In *S3GRL* models, across the attributed and non-attributed datasets, we set the hidden dimension in Eq. 4.6 to 256, and also implement Ω in Eq. 4.7 as an MLP with one 256-dimensional hidden layer. For all models, we set the dropout to 0.5 and train them for 50 epochs with Adam [48] and a batch size

of 32 (except for MPGNNs with full-batch training on the input graph).

For the comparison against BUDDY on OGB and Planetoid datasets [93], the results are taken from [12], except for ogbl-vessel dataset, where the baseline figures are taken from the publicly-shared leaderboards.[4]

For PPR, we set $\alpha = 0.85$. All methods (except heuristics, SGRL, and *S3GRL* methods) use a Hadamard product to create the link representation from the target pair's representations. All autoencoder-based models use a GCN encoder to produce node embeddings and an inner product of the learned node embedding matrix to reconstruct the original adjacency matrix. The hidden dimensionality (i.e., dimensions of embeddings) for all baselines taken for comparison on the attributed and non-attributed datasets is set to 32 except for the autoencoder models, SEAL, and GCN+DE. Autoencoder models use varying-sized hidden dimensionalities, ranging from 32 to 64, for their encoders whereas SEAL and GCN+DE have 256 hidden dimensions for the attributed datasets.

We normalize the adjacency matrix of the selected subgraphs in PoS and PoS$^+$ and the input graph adjacency matrix in SoP. Our graph normalization is symmetric degree normalization without additional self-loops (except for the datasets in Table 5.8, where re-normalization trick is used [50]). For *S3GRL* models trained on the non-attributed datasets, we use 16-dimensional node2vec pretrained embeddings [29] as the initial node features. In *S3GRL* models, we augment each subgraph's initial feature matrix with the zero-one labeling scheme, except for ogbl-citation2 and ogbl-ppa datasets where DRNL labeling scheme is used instead. Additionally, *S3GRL* models are implemented with a weight decay of 0.0001 and the ELU [17] activation function as the non-linearity function, except for experiments involving the Planetoid

[4]https://ogb.stanford.edu/docs/leader_linkprop hosts the ogbl-vessel leaderboard from which we report the baseline results as of April 2, 2023, except for BUDDY, run by adding support from https://github.com/melifluos/subgraph-sketching.

datasets and ogbl-vessel, ogbl-ddi and ogbl-collab datasets in Table 5.8, where ReLU is used instead.

To fit the datasets into memory, we train SEAL and GCN+DE on Pubmed in a dynamic train mode (i.e., subgraphs are extracted on the fly during training), and we set the maximum number of nodes per hop to 100 for WalkPool on Ecoli, PB, and PubMed datasets. For SEAL and WalkPool, we augment the initial node features with DRNL labeling whereas GCN+DE deploys a distance-encoding labeling scheme. For our baselines, the initial node features for non-attributed datasets are set to identity, except in WalkPool with a 16-dimensional vector of ones, and SEAL and GCN+DE with no node features. Finally, Walkpool, SEAL, and GCN+DE use a 32-dimensional embedding table for learning the node labels.

5.2.4 Reproducibility

All our experiments are performed over fixed random seeds ensuring reproducibility if repeated multiple times on the same hardware. However, it is to be noted that random numbers generated on different hardware could slightly vary our results. But, rerunning experiments on the same hardware ensures reproducibility on the same machine. Moreover, we share publicly all steps and scripts required to reproduce the experimental results showcased in this thesis.

5.3 Results and Discussions

In this section, we discuss in detail about the results for all experiments conducted. We compare and contrast the results of our *S3GRL* instances vs. the chosen baselines with a focus on discussing the efficacies and time taken.

5.3.1 Results on Attributed and Non-Attributed Datasets

On the attributed datasets, the *S3GRL* models, particularly PoS$^+$ and PoS, consistently outperform others (see Table 5.3). Their gain/improvement, compared to the best baseline, can reach 2.93 (for Wisconsin) and 2.77 (for CiteSeer). This state-of-the-art performance of PoS$^+$ and PoS suggests that our simplification techniques do not weaken SGRLs' efficacy and even improve their generalizability. Most importantly, this AUC gain is achieved by multiple times less computation: The *S3GRL* models benefit 2.3–13.9x speedup in training time for citation network datasets of Cora, CiteSeer, and PubMed (see Table 5.5). Similarly, inference time witnesses 3.1–51.2x speedup, where the maximum speedup is achieved on the largest attributed dataset PubMed. Our *S3GRL* models exhibit higher dataset preprocessing times compared to SGRLs (see Table 5.5). However, this is easily negated by our models' faster accumulative training and inference times that lead to 1.4–11.9x overall runtime speedup across all attributed datasets (min. for Texas and max. for PubMed). Moreover, we notice a significant reduction of dataset sizes, leading up to a 99% reduction (see Table 5.6 rows corresponding to PB and Yeast datasets) over typical SGRL datasets which consume all nodes related to each enclosing subgraph. This further showcases the feasibility of storage requirements offered by our *S3GRL* models in comparison to SGRLs like SEAL, DE-GNN, WalkPool etc. This reduction in sizes is due to the optimizations realized through our novel pooling operation coupled with the efficient precomputations.

We observe comparatively lower AUC values for SoP, which can be attributed to longer-range information being of less value on these datasets. Regardless, SoP still shows comparable AUC to the other SGRLs while offering substantially higher speedups. We also note that due to the preprocessed $\mathbf{Z}_{uv}$ matrices corresponding to PoS and SoP having similar matrix sizes, both utilize the same storage space on

disk. However, as stated earlier, SoP continues to be a faster dataset preparation mechanism due to the lack of explicit subgraph extractions, unlike its counterpart PoS.

Despite only consuming the source-target information, PoS achieves first or second place in the citation networks indicating the power of the center pooling. Moreover, the higher efficacy of PoS$^+$ compared to PoS across a few datasets indicate added expressiveness provided by center-common-neighbor pooling. Finally, the autoencoder methods outperform SGRLs on citation networks. However, *S3GRL* instances outperform them and have enhanced the learning capability of SGRLs.

For the non-attributed datasets, although WalkPool outperforms others, we see strong performance from our *S3GRL* instances. Our instances appear second or third (e.g., Yeast or Power) or have a small margin to the best model (e.g., NS or PB). The maximum loss in our models is bounded to 2.43% (see Power's gain in Table 5.2). Regardless of the small loss of efficacy, our *S3GRL* models demonstrate multi-fold speedup in training and inference times: training with 1.4–18.9x speedup and inference with 1.4–40.2x (the maximum training and inference speedup on Yeast). We see a similar pattern of higher preprocessing times for our models; however, it gets negated by faster training and inference times leading to an overall speedup in runtimes with the maximum of 15.8x for Yeast. SoP shows a relatively lower AUC in the non-attributed dataset except for PB and Yeast, possibly indicating that long-range information is more crucial for them. We again see substantially less storage requirements for our *S3GRL* models in comparison to SGRLs. For example, we get up to a 99% reduction in PubMed (See Table 5.7), a dataset in which SEAL requires 280 Gigabytes (GB) of memory to simply store the training dataset, in comparison to only a 2 GB requirement for all our instances.

For all datasets, we usually observe higher AUC for PoS$^+$ than its PoS variant

suggesting the expressiveness power of center-common-neighbor pooling over simple center pooling. Of course, these slight AUC improvements come with slightly higher computational costs (see Tables 5.5 and 5.4).

	Model	Non-attributed			
		NS	**Power**	**PB**	**Yeast**
Heuristic	AA	92.14±0.77	58.09±0.55	91.76±0.56	88.80±0.55
	CN	92.12±0.79	58.09±0.55	91.44±0.59	88.73±0.56
	PPR	92.50±1.06	62.88±2.18	86.85±0.48	91.71±0.74
MPGNN	GCN	91.75±1.68	69.41±0.90	90.80±0.43	91.29±1.11
	GraphSAGE	91.39±1.73	64.94±2.10	88.47±2.56	87.41±1.64
	GIN	83.26±3.81	58.28±2.61	88.42±2.09	84.00±1.94
LF	node2vec	91.44±0.81	73.02±1.32	85.08±0.74	90.60±0.57
	MF	82.56±5.90	53.83±1.76	91.56±0.56	87.57±1.64
AE	GAE	92.50±1.71	68.17±1.64	91.52±0.35	93.13±0.79
	VGAE	91.83±1.49	66.23±0.94	91.19±0.85	90.19±1.38
	ARVGA	92.16±1.05	66.26±1.59	90.98±0.92	90.25±1.06
	GIC	90.88±1.85	62.01±1.25	73.65±1.36	88.78±0.63
SGRL	SEAL	**98.63**±0.67	85.28±0.91	**95.07**±0.35	**97.56**±0.32
	GCN+DE	**98.66**±0.66	80.65±1.40	**95.14**±0.35	96.75±0.41
	WalkPool	**98.92**±0.52	**90.25**±0.64	**95.50**±0.26	**98.16**±0.20
S3GRL	PoS (ours)	97.23±1.38	**86.67**±0.98	94.83±0.41	95.47±0.54
	PoS$^+$ (ours)	98.37±1.26	**87.82**±0.96	95.04±0.27	**96.77**±0.39
	SoP (ours)	90.61±1.94	75.64±1.33	94.34±0.30	92.98±0.58
Gain		-0.55	-2.43	-0.46	-1.39

Table 5.2: Average AUC for non-attributed datasets (over 10 runs). The top 3 models are indicated by **First**, **Second**, and **Third**. Green is best model among our *S3GRL* variants. Yellow is the best baseline. Gain is AUC difference of Green and Yellow.

5.3.2 Results on Large Scale Datasets

As shown in Table 5.8, our PoS$^+$ model can easily scale to graph datasets with millions of nodes and edges. Under the experimental setup in BUDDY, our PoS$^+$ still outper-

	Model	Attributed				
		Cora	**CiteSeer**	**PubMed**	**Texas**	**Wisconsin**
Heuristic	AA	71.48±0.69	65.86±0.80	64.26±0.40	54.69±3.68	55.60±3.14
	CN	71.40±0.69	65.84±0.81	64.26±0.40	54.36±3.65	55.08±3.08
	PPR	82.87±1.01	74.35±1.51	75.80±0.35	53.81±7.53	62.86±8.13
MPGNN	GCN	89.14±1.20	87.89±1.48	92.72±0.64	67.42±9.39	72.77±6.96
	GraphSAGE	85.96±2.04	84.05±1.72	81.60±1.22	53.59±9.37	61.81±9.66
	GIN	68.74±2.74	69.63±2.77	82.49±2.89	63.46±8.87	70.82±8.25
LF	node2vec	78.32±0.74	75.36±1.22	79.98±0.35	52.81±5.31	59.57±5.69
	MF	62.25±2.21	61.65±3.80	68.56±12.13	60.35±5.62	53.75±9.00
AE	GAE	90.21±0.98	88.42±1.13	94.53±0.69	68.67±6.95	75.10±8.69
	VGAE	92.17±0.72	90.24±1.10	92.14±0.19	**74.61±8.61**	74.39±8.39
	ARVGA	**92.26±0.74**	90.29±1.01	92.10±0.38	73.55±9.01	72.65±7.02
	GIC	91.42±1.24	**92.99±1.14**	91.04±0.61	65.16±7.87	75.24±8.45
SGRL	SEAL	90.29±1.89	88.12±0.85	97.82±0.28	71.68±6.85	77.96±10.37
	GCN+DE	91.51±1.10	88.88±1.53	98.15±0.11	**76.60±6.40**	**74.65±9.56**
	WalkPool	92.24±0.65	89.97±1.01	**98.36±0.11**	**78.44±9.83**	79.57±11.02
S3GRL	PoS (ours)	**94.65±0.67**	**95.76±0.59**	**98.97±0.08**	73.75±8.20	**82.50±5.83**
	PoS$^+$ (ours)	**94.77±0.68**	**95.72±0.56**	**99.00±0.08**	**78.44±9.83**	**79.17±10.87**
	SoP (ours)	91.24±0.80	88.23±0.73	95.91±0.29	69.49±7.12	72.29±14.42
Gain		+2.51	+2.77	+0.64	0	+2.93

Table 5.3: Average AUC for attributed datasets (over 10 runs). The top 3 models are indicated by **First**, **Second**, and **Third**. Green is best model among our *S3GRL* variants. Yellow is the best baseline. Gain is AUC difference of Green and Yellow.

forms all the baselines (including BUDDY) for all Planetoid datasets, confirming the versatility of our model to different dataset splits and evaluation criteria. Our PoS$^+$ outperforms others on ogbl-collab, ogbl-vessel and ogbl-citation2 datasets as well. For ogbl-ppa, we come in third place. For ogbl-ddi, PoS$^+$ performs significantly lower than SEAL and BUDDY; but we still perform better than the heuristic baselines. This performance on ogbl-ddi might be a limitation of our PoS$^+$, possibly because the common-neighbor information is noisy in denser ogbl-ddi, with the performance degradation exaggerated by the lack of node feature information. However, this set

Model	Training	Inference	Preproc.	Runtime
	NS (non-attributed)			
SEAL	4.91 ± 0.23	0.14 ± 0.01	17.86	275.28
GCN+DE	3.58 ± 0.12	0.10 ± 0.01	11.73	198.98
WalkPool	7.66 ± 0.09	0.41 ± 0.02	12.18	427.03
PoS	2.24 ± 0.15	0.06 ± 0.01	34.86	152.23
PoS$^+$	2.54 ± 0.06	0.07 ± 0.00	41.43	173.78
SoP	2.26 ± 0.11	0.06 ± 0.00	24.67	142.45
Speedup	3.42(1.41)	6.83(1.43)	0.72(0.28)	3(1.14)
	Power (non-attributed)			
SEAL	11.73 ± 0.02	0.33 ± 0.01	44.48	658.14
GCN+DE	8.62 ± 0.27	0.25 ± 0.01	28.59	479.4
WalkPool	18.46 ± 0.76	0.87 ± 0.06	33.51	1024.55
PoS	5.58 ± 0.48	0.14 ± 0.01	97.71	388.97
PoS$^+$	6.12 ± 0.24	0.16 ± 0.01	107.77	426.53
SoP	5.41 ± 0.23	0.14 ± 0.01	65.65	347.62
Speedup	3.41(1.41)	6.21(1.56)	0.68(0.27)	2.95(1.12)
	Yeast (non-attributed)			
SEAL	24.03 ± 0.40	0.54 ± 0.05	115.02	1362.85
GCN+DE	18.41 ± 0.71	0.46 ± 0.06	82.19	1040.72
WalkPool	174.80 ± 1.06	8.05 ± 0.11	90.75	9443.17
PoS	9.95 ± 1.45	0.20 ± 0.06	259.96	775.58
PoS$^+$	10.49 ± 0.61	0.20 ± 0.04	206.52	749.87
SoP	9.24 ± 0.74	0.22 ± 0.04	117.23	597.29
Speedup	18.92(1.76)	40.25(2.09)	0.98(0.32)	15.81(1.34)
	PB (non-attributed)			
SEAL	64.62 ± 5.59	2.32 ± 0.10	531.79	3947.45
GCN+DE	55.82 ± 1.59	2.01 ± 0.09	398.81	3346.80
WalkPool	133.30 ± 0.52	6.48 ± 0.15	136.29	7291.50
PoS	13.42 ± 0.77	0.33 ± 0.04	1754.88	2452.39
PoS$^+$	15.56 ± 1.28	0.29 ± 0.05	2527.23	3331.56
SoP	13.32 ± 0.72	0.25 ± 0.06	333.58	1022.90
Speedup	10.01(3.59)	25.92(6.09)	1.59(0.05)	7.13(1.00)

Table 5.4: Computation time of SGRLs vs. our *S3GRL* models on non-attributed datasets: average training time (over 50 epochs), average inference time, preprocessing time, and total runtime (preprocessing, training, and inference time) for 50 epochs. Green is the fastest and Red is slowest for each group of SGRLs and *S3GRL*. Max(min) speedup corresponds to the ratio of time taken by the slowest (fastest) SGRLs to our fastest (slowest) model.

Model	**Training**	**Inference**	**Preproc.**	**Runtime**
	Cora (attributed)			
SEAL	18.37 ± 1.49	0.73 ± 0.12	113.32	1090.94
GCN+DE	14.85 ± 0.53	0.62 ± 0.08	80.48	872.68
WalkPool	18.53 ± 0.91	1.00 ± 0.15	27.43	1034.33
PoS	5.44 ± 0.52	0.15 ± 0.02	106.45	394.12
PoS$^+$	5.87 ± 0.17	0.17 ± 0.01	93.69	401.05
SoP	5.04 ± 0.17	0.15 ± 0.03	35.65	300.10
Speedup	3.68(2.53)	6.67(3.65)	3.18(0.26)	3.64(2.18)
	CiteSeer (attributed)			
SEAL	12.54 ± 0.69	0.58 ± 0.10	93.52	768.72
GCN+DE	11.43 ± 0.71	0.52 ± 0.07	71.97	685.98
WalkPool	15.32 ± 0.54	0.87 ± 0.05	22.82	859.27
PoS	4.82 ± 0.22	0.15 ± 0.01	78.62	335.37
PoS$^+$	4.91 ± 0.39	0.17 ± 0.02	72.02	331.55
SoP	4.96 ± 0.14	0.17 ± 0.01	31.57	293.96
Speedup	3.18(2.3)	5.8(3.06)	2.96(0.29)	2.92(2.05)
	PubMed (attributed)			
SEAL	533.18 ± 4.64	38.46 ± 1.08	141.76	30150.31
GCN+DE	423.73 ± 2.67	34.44 ± 1.21	106.00	24311.00
WalkPool	150.27 ± 6.22	8.10 ± 1.06	341.12	8474.72
PoS	38.90 ± 2.89	0.79 ± 0.10	2986.74	5017.78
PoS$^+$	45.32 ± 2.21	0.92 ± 0.11	2976.80	5335.86
SoP	38.38 ± 2.90	0.75 ± 0.15	526.62	2526.22
Speedup	13.89(3.32)	51.28(8.80)	0.65(0.04)	11.93(1.59)
	Texas (attributed)			
SEAL	0.32 ± 0.01	0.01 ± 0.00	2.55	20.46
GCN+DE	0.31 ± 0.01	0.01 ± 0.00	1.87	18.55
WalkPool	0.55 ± 0.08	0.03 ± 0.01	0.92	32.54
PoS	0.16 ± 0.01	0.01 ± 0.00	1.87	12.71
PoS$^+$	0.18 ± 0.01	0.01 ± 0.00	2.26	11.96
SoP	0.15 ± 0.01	0.01 ± 0.00	1.34	9.61
Speedup	2.62(1.72)	3(1)	1.9(0.41)	3.39(1.46)
	Wisconsin (attributed)			
SEAL	0.47 ± 0.01	0.02 ± 0.00	3.29	29.27
GCN+DE	0.43 ± 0.01	0.02 ± 0.00	2.63	26.19
WalkPool	0.85 ± 0.04	0.06 ± 0.00	1.08	49.38
PoS	0.21 ± 0.02	0.01 ± 0.00	2.65	15.86
PoS$^+$	0.24 ± 0.02	0.01 ± 0.00	2.91	16.07
SoP	0.22 ± 0.01	0.01 ± 0.00	1.87	13.75
Speedup	4.05(1.79)	6(2)	1.76(0.37)	3.59(1.63)

Table 5.5: Computation time of SGRLs vs. our *S3GRL* models on attributed datasets: average training time (over 50 epochs), average inference time, preprocessing time, and total runtime (preprocessing, training, and inference time) for 50 epochs. Green is the fastest and Red is slowest for each group of SGRLs and *S3GRL*. Max(min) speedup corresponds to the ratio of time taken by the slowest (fastest) SGRLs to our fastest (slowest) model.

Dataset	Model	Train Size	Validation Size	Test Size
NS	PoS	5.27	0.16	0.31
	PoS$^+$	8.83	0.27	0.53
	SoP	5.27	0.16	0.31
	SEAL	21.26	0.64	1.22
Reduction %		75.21	75	74.59
Power	PoS	12.66	0.37	0.75
	PoS$^+$	13.27	0.39	0.79
	SoP	12.66	0.37	0.75
	SEAL	24.88	0.72	1.49
Reduction %		49.11	48.61	49.66
PB	PoS	32.09	0.95	1.89
	PoS$^+$	137.19	4.04	7.99
	SoP	32.09	0.95	1.89
	SEAL	27814.36	816.16	1631.41
Reduction %		99.88	99.88	99.88
Yeast	PoS	22.45	0.66	1.32
	PoS$^+$	80.65	2.37	4.75
	SoP	22.45	0.66	1.32
	SEAL	2321.44	65.63	135.84
Reduction %		99.03	98.99	99.02

Table 5.6: Precomputed dataset sizes in Megabytes (MB) of SEAL vs. our *S3GRL* models on the non-attributed datasets. Green is the most storage efficient (smallest sized dataset) and Red is the least storage efficient (largest sized dataset). Reduction % corresponds to the maximum reduction in storage requirements for preparing the dataset, i.e., (Red - Green) / Red $\times$ 100.

Dataset	Model	Train Size	Validation Size	Test Size
Cora	PoS	786.44	23.05	46.18
	PoS$^+$	920.93	26.81	53.83
	SoP	786.44	23.05	46.18
	SEAL	19521.52	620.09	1172.68
Reduction %		95.97	96.28	96.06
CiteSeer	PoS	1750.53	51.34	102.91
	PoS$^+$	1996.66	57.90	117.38
	SoP	1750.53	51.34	102.91
	SEAL	18716.19	500.03	1083.71
Reduction %		90.64	89.73	90.50
PubMed	PoS	2311.06	67.97	135.93
	PoS$^+$	2665.93	78.58	156.21
	SoP	2311.06	67.97	135.93
	SEAL	287324.77	8500.97	16670.80
Reduction %		99.19	99.20	99.18
Texas	PoS	29.77	0.84	1.67
	PoS$^+$	32.17	0.86	1.90
	SoP	29.77	0.84	1.67
	SEAL	295.89	7.04	15.53
Reduction %		89.93	88.06	89.24
Wisconsin	PoS	41.02	1.15	2.40
	PoS$^+$	44.27	1.23	2.50
	SoP	41.02	1.15	2.40
	SEAL	349.59	9.99	20.04
Reduction %		88.26	88.48	88.02

Table 5.7: Precomputed dataset sizes in Megabytes (MB) of SEAL vs. our *S3GRL* models on the attributed datasets. Green is the most storage efficient (smallest sized dataset) and Red is the least storage efficient (largest sized dataset). Reduction % corresponds to the maximum reduction in storage requirements for preparing the dataset, i.e., (Red - Green) / Red $\times$ 100.

Model	Cora HR@100	CiteSeer HR@100	PubMed HR@100	Collab HR@50	DDI HR@20	Vessel roc-auc	Citation2 MRR	PPA HR@100
CN	33.92±0.46	29.79±0.90	23.13±0.15	56.44±0.00	17.73±0.00	48.49±0.00	51.47±0.00	27.65±0.00
AA	39.85±1.34	35.19±1.33	27.38±0.11	64.35±0.00	18.61±0.00	48.49±0.00	51.89±0.00	32.45±0.00
GCN	66.79±1.65	67.08±2.94	53.02±1.39	44.75±1.07	**37.07±5.07**	43.53±9.61	84.74±0.21	18.67±1.32
SAGE	55.02±4.03	57.01±3.74	39.66±0.72	48.10±0.81	**53.90±4.74**	49.89±6.78	82.60±0.36	16.55±2.40
SEAL	**81.71±1.30**	**83.89±2.15**	**75.54±1.32**	**64.74±0.43**	30.56±3.86	**80.50±0.21**	**87.67±0.32**	**48.80±3.16**
BUDDY	**88.00±0.44**	**92.93±0.27**	**74.10±0.78**	**65.94±0.58**	**78.51±1.36**	**55.14±0.11**	**87.56±0.11**	**49.85±0.20**
PoS$^+$ (ours)	**91.55±1.16**	**94.79±0.58**	**79.40±1.53**	**66.83±0.30**	22.24±3.36	**80.56±0.06**	**88.14±0.08**	**42.42±1.80**

Table 5.8: Results for PoS$^+$ in comparison to the datasets and baselines chosen in BUDDY [12]. The top three performing models are **First**, **Second**, and **Third**.

of experiments confirms the competitive performance of PoS$^+$ on the OGB datasets with large-scale graphs under different types of efficacy measurements.

5.3.3 *S3GRL* as a scalability framework.

To demonstrate the flexibility of *S3GRL* as a scalability framework, we explore how easily it can host other scalable SGRLs. Specifically, we exchange the subgraph sampling strategy of PoS (and PoS$^+$) with the random-walk induced subgraph sampling technique in ScaLed. Through this process, we reduce our operator sizes and benefit from added regularization offered through stochasticity in random walks. We fixed its hyperparameters, the random walk length $h = 3$, and the number of random walks $k = 20$. Table 5.9 shows the average computational time and AUC (over 10 runs). The subgraph sampling of ScaLed offers further speedup in PoS and PoS$^+$ in training, dataset preprocessing, and overall runtimes. For PoS$^+$ variants, we even witness AUC gains on Cora and no AUC losses on CiteSeer. The AUC gains of PoS$^+$ could be attributed to the regularization offered through sparser enclosing subgraphs. This demonstration suggests that *S3GRL* can provide a foundation for hosting various SGRLs to further boost their scalability.

	Model	Training	Preproc.	Runtime	AUC
Cora	PoS	4.82±0.25	83.28±3.16	336.70±2.47	94.65±0.67
	PoS + ScaLed	4.73±0.22	60.32±3.50	308.73±4.27	94.35±0.52
	PoS$^+$	5.59±0.31	94.33±7.36	387.95±7.81	94.77±0.65
	PoS$^+$ + ScaLed	5.49±0.28	69.95±5.12	358.40±7.08	94.80±0.58
CiteSeer	PoS	4.66±0.20	61.86±0.64	308.20±2.86	95.76±0.59
	PoS + ScaLed	4.65±0.21	56.65±2.59	302.45±3.95	95.52±0.65
	PoS$^+$	4.72±0.35	79.14±5.34	330.29±5.61	95.60±0.52
	PoS$^+$ + ScaLed	4.66±0.27	65.56±5.13	313.38±6.25	95.60±0.53

Table 5.9: Results for *S3GRL* as a scalability framework: ScaLed's subgraph sampling combined with PoS and PoS$^+$.

5.4 Summary

We showcase the generalizability of our *S3GRL* instances, PoS and SoP on a variety of small to large-scale data with varying types of efficacy measurements. *S3GRL* proves to be a faster way of reaching the performance of SGRLs with efficient pre-computations offered through operator diffusion.

Chapter 6

Conclusions

We dedicate this chapter to conclude our work and discuss in detail a plethora of future directions aimed at further improving *S3GRL*. We wish to showcase the strength of our *S3GRL* as a foundational SGRL scalability framework and how it can be used in constructing future SGRLs.

6.1 Thesis Summary

Subgraph representation learning methods (SGRLs), albeit comprising state-of-the-art models for link prediction, suffer from large computational overheads. This thesis proposes a novel SGRL framework *S3GRL*, aimed at faster inference and training times while offering flexibility to emulate many other SGRLs. We achieve this speedup through easily precomputable subgraph-level diffusion operators in place of expensive message-passing schemes. *S3GRL* supports multiple subgraph selection choices for the creation of the operators, allowing for a multi-scale view around the target links. Our experiments on multiple instances of our *S3GRL* framework show significant computational speedup over existing SGRLs, while offering matching (or higher) link

prediction efficacies.

6.2 Future Directions

We dedicate this section to detail the planned future works involving our *S3GRL* frameworks and its instances. Each future work is explained in more detail in the following subsections.

6.2.1 *Hybrid* PoS

One of current limitations of our *S3GRL* framework is the existence of the h, r hyperparameter choices. h controls the depth of the subgraphs, whereas, r controls the number of operators created. However, as an end-user, it might be difficult to fine-tune the h value to the dataset at hand. To this end, one might consider a *hybrid* version of PoS. *Hybrid* PoS is an extension to PoS's style of operator creation with a few modifications: Ψ_r is modified to create the 1-hop, 2-hop, ..., until the h hop subgraph for each link, while Φ_r calculates up to the r-th power of each enclosing subgraph. *Hybrid* is aimed at making our W learn to assign different "importance" to each i-th hop subgraph and its corresponding views. We present *hybrid* as a cheaper way to adaptively learn h values from the underlying data.

6.2.2 Operator Creation Parallelization

Another interesting research direction is to parallelize the initial subgraph extraction and operator creation methods. Currently, the operator creation phase of *S3GRL* is performed sequentially for each link in the dataset. However, there is scope for faster processing by having multiple processes or threads performing a subset of the operator creation in parallel. This parallelization of the operator creation in *S3GRL*

could substantially speed up the preprocessing and lead to a overall faster runtimes, especially in servers with multiple CPU cores available.

6.2.3 Recommendation with Implicit Data

One of the potential applications of our *S3GRL* framework is in recommender systems with implicit feedback [4, 76]. The link prediction objective in *S3GRL* can easily be extended to recommendation settings. One can view user-item or group-item interactions as a bipartite graph where the goal is to predict an interaction (or a link) between users (or groups) and items. With the abundance of implicit data available (e.g., amount of time spent on a post, amount of times a media was replayed, etc.), *S3GRL* can be used for recommender systems driven by implicit feedback. Moreover, this can also help bring in more scalability to recommender systems owing to the scalability components and techniques introduced in *S3GRL*.

6.2.4 Training *S3GRL* on Dynamic Graphs

In this work we present *S3GRL*, a novel end-to-end SGRL framework for learning on *static graphs*. Static graphs are graphs who's edges and nodes do not change over time. However, there also exists scenarios where graphs are likely to lose or gain nodes and links as time passes. These graphs are referred to as *dynamic graphs*. Examples of systems that employ dynamic graphs include recommender systems and financial transaction systems. One future work is to study the effect of training *S3GRL* instances on dynamic graphs. One can easily extend the precomputations and operator creation to involve the new nodes and links that are added on an ad-hoc basis. However, the effect of the addition of new nodes and links on the already trained data might result in stale learnt weights. In the future, we wish to come up

with efficient ways to correctly identify such stale data points and re-train the model on the new nodes/links that get added.

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).

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

[3] Alsentzer, E., Finlayson, S., Li, M., and Zitnik, M. Subgraph neural networks. *Advances in Neural Information Processing Systems 33* (2020), 8017–8029.

[4] Askari, B., Szlichta, J., and Salehi-Abari, A. Variational autoencoders for top-k recommendation with implicit feedback. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval* (2021), pp. 2061–2065.

[5] Barabási, A.-L., and Albert, R. Emergence of scaling in random networks. *Science 286*, 5439 (1999), 509–512.

[6] Bevilacqua, B., Frasca, F., Lim, D., Srinivasan, B., Cai, C., Balamurugan, G., Bronstein, M. M., and Maron, H. Equivariant subgraph

aggregation networks. In *International Conference on Learning Representations* (2022).

[7] Bottou, L., et al. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nimes 91*, 8 (1991), 12.

[8] Bruna, J., Zaremba, W., Szlam, A., and Lecun, Y. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations* (2014).

[9] 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.

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

[11] Cao, S., Lu, W., and Xu, Q. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* (New York, NY, USA, 2015), CIKM '15, Association for Computing Machinery, p. 891–900.

[12] 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).

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

[14] Chen, J., Zhu, J., and Song, L. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning* (2018), pp. 942–950.

[15] 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.

[16] 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.

[17] 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).

[18] Davidson, T. R., Falorsi, L., De Cao, N., Kipf, T., and Tomczak, J. M. Hyperspherical variational auto-encoders. In *Uncertainty in Artificial Intelligence* (2018).

[19] 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).

[20] Dong, Y., Chawla, N. V., and Swami, A. Metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2017), KDD '17, Association for Computing Machinery, p. 135–144.

[21] Du, J., Zhang, S., Wu, G., Moura, J. M., and Kar, S. Topology adaptive graph convolutional networks. *arXiv preprint arXiv:1710.10370* (2017).

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

[23] Fey, M., Lenssen, J. E., Weichert, F., and Leskovec, J. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *International Conference on Machine Learning* (2021), PMLR, pp. 3294–3304.

[24] Fortunato, S. Community detection in graphs. *Physics Reports 486*, 3 (2010), 75–174.

[25] Frasca, F., Bevilacqua, B., Bronstein, M., and Maron, H. Understanding and extending subgraph gnns by rethinking their symmetries. *Advances in Neural Information Processing Systems 35* (2022), 31376–31390.

[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] Gasteiger, J., Weissenberger, S., and Günnemann, S. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems* (2019).

[28] Granovetter, M. The strength of weak ties: A network theory revisited. *Sociological theory* (1983), 201–233.

[29] Grover, A., and Leskovec, J. node2vec: Scalable feature learning for networks. In *International Conference on Knowledge Discovery and Data mining* (2016).

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

[31] Gysi, D. M., 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 118*, 19 (2021).

[32] Hamilton, W. L. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning 14*, 3 (2020), 1–159.

[33] 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* (Red Hook, NY, USA, 2017), NIPS'17, Curran Associates Inc., p. 1025–1035.

[34] 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, IJCAI-20* (7 2020), pp. 1209–1215. Main track.

[35] Hinton, G., Vinyals, O., Dean, J., et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531 2*, 7 (2015).

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

[37] Hoff, P. D., Raftery, A. E., and Handcock, M. S. Latent space approaches to social network analysis. *Journal of the American Statistical Association 97*, 460 (2002), 1090–1098.

[38] Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks 4*, 2 (1991), 251–257.

[39] Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks 2*, 5 (1989), 359–366.

[40] Hu, W., Fey, M., Ren, H., Nakata, M., Dong, Y., and Leskovec, J. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430* (2021).

[41] Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687* (2020).

[42] Huang, K., Xiao, C., Glass, L. M., Zitnik, M., and Sun, J. Skipgnn: predicting molecular interactions with skip-graph networks. *Scientific reports 10*, 1 (2020), 1–16.

[43] Huang, Y., Peng, X., Ma, J., and Zhang, M. Boosting the cycle counting power of graph neural networks with I^2-gnns. *arXiv preprint arXiv:2210.13978* (2022).

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

[45] Jeh, G., and Widom, J. Simrank: A measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on*

Knowledge Discovery and Data Mining (New York, NY, USA, 2002), KDD '02, Association for Computing Machinery, p. 538–543.

[46] Jia, Z., Lin, S., Ying, R., You, J., Leskovec, J., and Aiken, A. Redundancy-free computation for graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2020), pp. 997–1005.

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

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

[49] Kipf, T. N., and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

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

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

[52] Kovács, I. A., Luck, K., Spirohn, K., Wang, Y., Pollis, C., Schlabach, S., Bian, W., Kim, D.-K., Kishore, N., Hao, T., et al. Network-based prediction of protein interactions. *Nature communications 10*, 1 (2019), 1240.

[53] Kumar, A., Singh, S. S., Singh, K., and Biswas, B. Link prediction techniques, applications, and performance: A survey. *Physica A: Statistical Mechanics and its Applications 553* (2020), 124289.

[54] Li, B., Xia, Y., Xie, S., Wu, L., and Qin, T. Distance-enhanced graph neural network for link prediction.

[55] 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 33* (2020), 4465–4478.

[56] Liben-Nowell, D., and Kleinberg, J. The link prediction problem for social networks. In *International Conference on Information and Knowledge Management* (2003).

[57] Louis, P., Jacob, S. A., and Salehi-Abari, A. Sampling enclosing subgraphs for link prediction. In *International Conference on Information & Knowledge Management* (2022).

[58] Louis, P., Jacob, S. A., and Salehi-Abari, A. Simplifying subgraph representation learning for scalable link prediction, 2023.

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

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

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

[62] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

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

[64] Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663* (2020).

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

[66] Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *International conference on machine learning* (2016), PMLR, pp. 2014–2023.

[67] Ou, M., Cui, P., Pei, J., Zhang, Z., and Zhu, W. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (2016), pp. 1105–1114.

[68] Page, L., Brin, S., Motwani, R., and Winograd, T. The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab, 1999.

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

[70] Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., and Zhang, C. Adversarially regularized graph autoencoder for graph embedding. In *International Joint Conference on Artificial Intelligence* (2018).

[71] 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).

[72] Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *International Conference on Knowledge Discovery and Data mining* (2014).

[73] Pho, P., and Mantzaris, A. V. Link prediction with simple graph convolution and regularized simple graph convolution. In *International Conference on Information System and Data Mining* (2022).

[74] Ren, H., Dai, H., Dai, B., Chen, X., Zhou, D., Leskovec, J., and Schuurmans, D. Smore: Knowledge graph completion and multi-hop reasoning in massive knowledge graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2022), pp. 1472–1482.

[75] Ribeiro, L. F., Saverese, P. H., and Figueiredo, D. R. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining* (2017), pp. 385–394.

[76] Sajjadi Ghaemmaghami, S., and Salehi-Abari, A. Deepgroup: Group recommendation with implicit feedback. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (New York, NY, USA, 2021), CIKM '21, Association for Computing Machinery, p. 3408–3412.

[77] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 1–9.

[78] 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* (New York, NY, USA, 2023), WSDM '23, Association for Computing Machinery, p. 625–633.

[79] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web* (2015), pp. 1067–1077.

[80] Tang, L., and Liu, H. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery 23* (2011), 447–478.

[81] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems* (2017).

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

[83] Wang, P., Xu, B., Wu, Y., and Zhou, X. Link prediction in social networks: the state-of-the-art. *Science China Information Sciences 58*, 1 (2015), 1–38.

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

[85] 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.

[86] Wishart, D. S., Feunang, Y. D., Guo, A. C., Lo, E. J., Marcu, A., Grant, J. R., Sajed, T., Johnson, D., Li, C., Sayeeda, Z., Assempour, N., Iynkkaran, I., Liu, Y., Maciejewski, A., Gale, N., Wilson, A., Chin, L., Cummings, R., Le, D., Pon, A., Knox, C., and Wilson, M. DrugBank 5.0: a major update to the DrugBank database for 2018. *Nucleic Acids Research 46*, D1 (11 2017), D1074–D1082.

[87] Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International Conference on Machine Learning* (2019).

[88] Xiong, Z., Wang, D., Liu, X., Zhong, F., Wan, X., Li, X., Li, Z., Luo, X., Chen, K., Jiang, H., et al. Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism. *Journal of medicinal chemistry 63*, 16 (2019), 8749–8760.

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

[90] 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).

[91] Yanardag, P., and Vishwanathan, S. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2015), KDD '15, Association for Computing Machinery, p. 1365–1374.

[92] Yang, J., and Leskovec, J. Overlapping communities explain core–periphery organization of networks. *Proceedings of the IEEE 102* (12 2014), 1892–1902.

[93] Yang, Z., Cohen, W., and Salakhudinov, R. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning* (2016), PMLR, pp. 40–48.

[94] Yin, H., Zhang, M., Wang, J., and Li, P. Surel+: Moving from walks to sets for scalable subgraph-based graph representation learning. *arXiv preprint arXiv:2303.03379* (2023).

[95] Yin, H., Zhang, M., Wang, Y., Wang, J., and Li, P. Algorithm and system co-design for efficient subgraph-based graph representation learning. *Proceedings of the VLDB Endowment 15*, 11 (2022), 2788–2796.

[96] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender

systems. In *International Conference on Knowledge Discovery and Data mining* (2018).

[97] You, J., Gomes-Selman, J., Ying, R., and Leskovec, J. Identity-aware graph neural networks. *arXiv preprint arXiv:2101.10320* (2021).

[98] You, J., Xiao, G., Ying, R., and Leskovec, J. Hybridgnn: Scaling deep gnns on large graphs.

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

[100] 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. In *Advances in Neural Information Processing Systems* (2021).

[101] Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. Accurate, efficient and scalable graph embedding. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2019), IEEE, pp. 462–471.

[102] 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).

[103] Zhang, M., and Chen, Y. Weisfeiler-lehman neural machine for link prediction. In *International Conference on Knowledge Discovery and Data Mining* (2017).

[104] Zhang, M., and Chen, Y. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems* (2018).

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

[106] Zhang, M., and Li, P. Nested graph neural networks. In *Advances in Neural Information Processing Systems* (2021), M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., pp. 15734–15747.

[107] 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), vol. 34.

[108] Zhang, S., Liu, Y., Sun, Y., and Shah, N. Graph-less neural networks: Teaching old mlps new tricks via distillation. *arXiv preprint arXiv:2110.08727* (2021).

[109] Zhang, Z., Zhuang, F., Zhu, H., Shi, Z., Xiong, H., and He, Q. Relational graph neural network with hierarchical attention for knowledge graph completion. In *AAAI Conference on Artificial Intelligence* (2020).

[110] Zhao, L., Jin, W., Akoglu, L., and Shah, N. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *International Conference on Learning Representations* (2022).

[111] Zhong, W., He, C., Xiao, C., Liu, Y., Qin, X., and Yu, Z. Long-distance dependency combined multi-hop graph neural networks for protein–protein interactions prediction. *BMC bioinformatics 23*, 1 (2022), 1–21.

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

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

[114] Zitnik, M., and Leskovec, J. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics 33*, 14 (07 2017), i190–i198.

[115] 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. In *Advances in Neural Information Processing Systems* (2019).