

ScholarWorks@GSU

Mining Local and Global Structure on Static and Dynamic Graphs

| | |
|---------------|---|
| Item Type | Dissertation |
| Authors | Islam, Muhammad Ifte Khairul |
| Citation | Islam, Muhammad Ifte Khairul |
| DOI | https://doi.org/10.57709/y8hq-cr79 |
| Download date | 2026-04-13 02:46:26 |
| Link to Item | https://hdl.handle.net/20.500.14694/15658 |

Mining Local and Global Structure on Static and Dynamic Graphs

by

Muhammad Ifta Khairul Islam

Under the Direction of Esra Akbas, Ph.D.

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2025

ABSTRACT

Graphs provide a powerful framework for representing complex relationships in diverse domains, including social networks, biological systems, transportation, and communication networks. Graph representation learning has emerged as a key technique for extracting meaningful embeddings, enabling tasks such as node classification, graph classification, and community detection. While Graph Neural Networks (GNNs) have significantly advanced this field, many existing methods fail to jointly capture local structures and higher-order dependencies, limiting their effectiveness. To address these challenges, this dissertation proposes novel graph representation learning models that integrate both local and global structural information for improved performance across static and dynamic graphs. **In our first work**, we introduce a proximity-based graph compression method for network embedding. Our approach leverages neighborhood similarity to compress the input graph into a smaller, more structured representation, preserving local proximity within super-nodes. Learning embeddings on the compressed graph reduces computational costs while maintaining global structural information. We then refine these embeddings back to the original graph, enhancing representation quality. **In our Second work**, we propose a multi-channel Motif-based Graph Pooling method named (MPool) that captures the higher-order graph structure with motif and also considers the local and global graph structure through a combination of selection and clustering-based pooling operations. **In our third work**, we propose a graph contrastive learning model with graph compression for graph classification. Using K-core and K-truss-based compression, we generate two views that capture both low-order and higher-order structures. These representations are optimized via contrastive loss and integrated for the final graph classification. **In our fourth work**, we propose a Dynamic Graph Contrastive Learning (DyGCL) method for event prediction. Our model DyGCL employs a local view encoder to effectively capture the local dynamic structure and a global view encoder

to capture the higher-order structure of the dynamic graphs. Our extensive experiments demonstrate that our proposed methods outperforms the state-of-the-art methods for different graph mining tasks like node classification, graph classification, and event prediction on various real-world datasets. These contributions advance graph representation learning by effectively integrating local and global structural information, opening new possibilities for more interpretable and scalable models in future research.

INDEX WORDS: Graph Neural Network, Graph Pooling, Graph Contrastive Learning, Graph Classification, Event Prediction

Copyright by
Muhammad Ifta Khairul Islam
2025

Mining Local and Global Structure on Static and Dynamic Graphs

by

Muhammad Ifta Khairul Islam

Committee Chair:

Esra Akbas

Committee:

Zhipeng Cai

Hemanth D Venkateswara

Ugur Kursuncu

Electronic Version Approved:

Office of Graduate Services

College of Arts and Sciences

Georgia State University

July 2025

DEDICATION

To my Family.

ACKNOWLEDGMENTS

I am grateful to Dr. Esra Akbas, my dissertation advisor and committee chair, for her thoughtful guidance, insightful suggestions, and steady support throughout my doctoral studies. Her mentorship has had a meaningful impact on the way I approach research and problem-solving. I would also like to thank Dr. Zhipeng Cai, Dr. Hemanth D. Venkateswara, and Dr. Ugur Kursuncu for serving on my committee and for their valuable time, perspectives, and encouragement.

I appreciate the collaborative spirit and helpful discussions shared by members of DELab, which contributed significantly to my progress. I also acknowledge the Department of Computer Science at Georgia State University for maintaining a strong academic environment, and the National Science Foundation (NSF) for supporting this research through funding and resources.

I am thankful to Allah (SWT) for granting me the strength and perseverance needed to complete this work, and to the Prophet Muhammad (peace be upon him) for guiding principles that continue to inspire my daily life. Finally, I owe heartfelt thanks to my family and friends for their continued support and understanding—your presence has been a steady source of motivation

TABLE OF CONTENTS

| | |
|--|-----------|
| ACKNOWLEDGMENTS | v |
| LIST OF TABLES | ix |
| LIST OF FIGURES | xi |
| 1 INTRODUCTION | 1 |
| 1.1 Background and Motivation | 1 |
| 1.2 Overview of the Studies | 2 |
| 1.2.1 Proximity-Based Compression for Network Embedding | 2 |
| 1.2.2 MPool: Motif-Based Graph Pooling | 4 |
| 1.2.3 Graph Contrastive Learning via Structure-Aware Graph Compression | 5 |
| 1.2.4 DyGCL: Dynamic Graph Contrastive Learning For Event Prediction | 6 |
| 2 RELATED WORKS | 9 |
| 2.1 Network Embedding | 9 |
| 2.2 Graph Neural Networks | 11 |
| 2.3 Graph Pooling | 12 |
| 2.4 Event prediction | 13 |
| 2.4.1 Graph-based event prediction | 14 |
| 3 Proximity-based Compression for Network Embedding | 16 |
| 3.1 Introduction | 16 |
| 3.2 Methodology | 17 |
| 3.2.1 Neighborhood similarity-based graph compression | 18 |
| 3.2.2 Network Embedding | 24 |
| 3.3 Experiments | 27 |
| 3.3.1 Analysis of NECL | 30 |

| | | |
|----------|---|-----------|
| 3.3.2 | Comparisons of All Methods | 36 |
| 4 | MPool: Motif-Based Graph Pooling | 40 |
| 4.1 | Introduction | 40 |
| 4.2 | Methodology | 40 |
| 4.2.1 | Framework | 41 |
| 4.2.2 | Preliminaries and Problem Formulation | 41 |
| 4.2.3 | Motif Based Graph Pooling Models | 43 |
| 4.2.4 | Readout Function and Output Layer | 48 |
| 4.2.5 | Graph Reconstruction | 48 |
| 4.2.6 | Complexity Analysis | 49 |
| 4.3 | Experiment | 49 |
| 4.3.1 | Overall Evaluation | 51 |
| 5 | Graph Contrastive Learning via Structure-Aware Graph Compression | 58 |
| 5.1 | Introduction | 58 |
| 5.2 | Methodology | 59 |
| 5.2.1 | Preliminaries and Problem Formulation | 59 |
| 5.2.2 | Graph Neural Network | 59 |
| 5.2.3 | Contrastive Learning | 60 |
| 5.2.4 | Graph Contrastive Learning with Graph Compression | 62 |
| 5.3 | Experiments | 67 |
| 5.3.1 | Datasets: | 68 |
| 5.3.2 | Baseline: | 69 |
| 5.3.3 | Experimental Setup: | 69 |
| 5.3.4 | Result: | 70 |
| 6 | DyGCL: Dynamic Graph Contrastive Learning For Event Prediction | 73 |
| 6.1 | Introduction | 73 |
| 6.2 | Methodology | 74 |

| | | |
|----------|---|------------|
| 6.2.1 | Preliminaries and Problem Description | 74 |
| 6.2.2 | Dynamic Graph Contrastive Learning | 76 |
| 6.3 | Experiment | 86 |
| 6.3.1 | Datasets | 86 |
| 6.3.2 | Baseline | 87 |
| 6.3.3 | Experimental Settings | 90 |
| 6.3.4 | Result | 90 |
| 7 | FUTURE WORK | 97 |
| 8 | CONCLUSION | 99 |
| | REFERENCES | 101 |

LIST OF TABLES

| | | |
|-----|--|----|
| 3.1 | Performance comparisons of NECL with baseline methods (BL) | 30 |
| 3.2 | Compression ratio with the similarity threshold $\lambda = 0.5$ | 31 |
| 3.3 | Performance comparisons of all methods | 36 |
| 4.1 | Graph statistics. $ G $, V_{avg} , E_{avg} , and $ C $ denote the number of graphs, the average number of nodes and edges, the number of classes in each dataset, respectively. | 50 |
| 4.2 | Comparison of our models with baseline pooling methods for biochemical datasets. | 52 |
| 4.3 | Comparison of our models with baseline pooling methods for social network datasets. | 53 |
| 4.4 | MPool _S MPool _C and MPool _{cmb} performance with different GNN models. | 55 |
| 4.5 | MPool _S MPool _C and MPool _{cmb} performance with different motifs. | 55 |
| 4.6 | Autoencoder position loss. | 57 |
| 5.1 | Dataset statistics. $ G $ is the number of graphs, V_{avg} and E_{avg} denote the average number of nodes and edges per graph, and $ C $ is the number of classes. | 68 |
| 5.2 | Comparison of our models with baseline methods on Graph Classification. | 69 |
| 5.3 | Performance comparisons with the main model (DGCL), the model with supervised loss ($DGCL_{sup}$), the model with K-core view encoder ($DGCL_{core}$), and the model with K-truss view encoder ($DGCL_{truss}$). | 71 |
| 5.4 | Performance comparisons of different GNN models on GCLC for DD and IMDB datasets. | 72 |
| 5.5 | Performance comparisons of different values on α on GCLC for DD and IMDB datasets. | 72 |
| 6.1 | Dataset statistics. $ S $ is the number of samples, \bar{N} and \bar{E} are the average number of nodes and edges, respectively, and $ Ev $ is the number of events. | 86 |

| | | |
|-----|--|----|
| 6.2 | Performance comparisons of our model with baseline models on event prediction | 86 |
| 6.3 | Performance comparisons of different GNN models on DyGCL for Thailand and Egypt datasets. | 93 |
| 6.4 | Performance comparisons of different graph pooling models on DyGCL for Thailand and Egypt datasets. | 94 |
| 6.5 | Performance comparisons of different RNN models on DyGCL for Thailand and Egypt datasets. | 95 |
| 6.6 | Performance comparisons with the main model (DyGCL), the model with supervised loss (DyGCL _{sup}), the model with local encoder (DyGCL _L), and the model with global encoder (DyGCL _G | 95 |

LIST OF FIGURES

| | | |
|------|---|----|
| 3.1 | Example of graph compressing: a and b are merged into super-node ab connected to the neighbours of both a and b. | 20 |
| 3.2 | Detailed classification results on Citeseer. | 32 |
| 3.3 | Detailed classification results on Wiki. | 33 |
| 3.4 | Detailed classification results on DBLP. | 33 |
| 3.5 | Detailed classification results on BlogCatalog. | 34 |
| 3.6 | Run time analyses for different similarity threshold values λ | 35 |
| 3.7 | The ratio of vertices/edges of the compressed graphs to that of the original graphs. | 35 |
| 3.8 | Detailed comparisons of classification results on Citeseer | 38 |
| 3.9 | Detailed comparisons of classification results on Wiki | 38 |
| 3.10 | Detailed comparisons of classification results on DBLP | 39 |
| 3.11 | Detailed comparisons of classification results on BlogCatalog | 39 |
| 4.1 | Motif Networks with size 2-4. | 43 |
| 4.2 | An illustration of our motif-based pooling methods. | 44 |
| 4.3 | Graph reconstruction Results after Pooling. | 56 |
| 5.1 | An illustration of our Graph Contrastive Learning with Graph Compression. | 62 |
| 6.1 | An overview of Dynamic Graph Contrastive Learning, (DyGCL) architecture. We feed input graphs into the Local View Encoder and Global View Encoder. Local View encoder learns the dynamic graph representation through the dynamic node representation. Global-View Encoder learns dynamic graph representation through graph pooling and the LSTM model. After optimizing representations by contrastive learning, they are combined by an MLP layer and fed to the predictor for event prediction. | 75 |

| | | |
|-----|--|----|
| 6.2 | Detailed Event prediction results of DyGCL model for the different number of historic and lead days. | 90 |
| 6.3 | Temporal pooled graphs in Global View Encoder for a sample event in NYC cab dataset. | 94 |

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

A graph is a mathematical structure used to model relationships between entities. It consists of nodes (vertices) representing entities and edges representing their connections. Graphs are prevalent in real-world applications, such as computer networks (routers/computers as nodes, links as edges), social networks (individuals and their interconnections, including co-authorship and citation networks), and biological networks (protein interactions essential for biological functions).

Recently, graph representation learning from graphs has become popular for many problems in network analysis [1, 2, 3, 4, 5]. Graph representation learning is defined as encoding structural information of graphs into a low-dimensional vector space on their connections [6]. By preserving the structural information of the network, nodes with links will be close to each other in the vector space. Recently, deep learning models have been adopted for graph representation learning, and many different Graph Neural Networks (GNNs) [7] have been proposed to capture the complex relation information within graph data. GNNs have shown competitive performance for many graph-related tasks, such as semi-supervised node classification, link prediction, and graph classification. By inputting a graph with node attributes and message propagation along the edges, while some GNN models learn the node-level representation for node classification [7, 8, 9, 10, 11, 12, 13], some others learn graph-level representation for graph classification [14, 15, 16].

However, many real-world graphs exhibit higher-order structures, including long-range dependencies, hierarchical relationships, and temporal dynamics. Existing approaches often struggle to integrate both local and global structures, limiting their effectiveness. These methods are limited to capturing local structures, such as immediate node neighborhoods, and struggle to represent higher-order structures. This limitation hinders the performance of graph representation in tasks requiring a deeper understanding of global graph topology.

This dissertation addresses these challenges by proposing four novel methods for graph representation learning that effectively capture higher-order and lower-order structures in both static and dynamic graphs. Our methods enhance graph representations by incorporating global structural patterns and temporal dependencies, enabling more expressive and adaptive learning frameworks. We investigate two important problems of graph representation learning: node classification and graph classification. Through extensive experiments, we demonstrate their advantages in improving predictive performance across various graph-based tasks.

1.2 Overview of the Studies

1.2.1 Proximity-Based Compression for Network Embedding

In this study, we aim to develop a more efficient and effective approach for network embedding. We consider an undirected, connected, and simple graph, simple graph $G = (V; E)$ where V is the set of vertices, and $E \subseteq \{V \times V\}$ is the set of edges. Formally, we define network embedding as follows:

Definition 1 (Network embedding). *Network embedding is a mapping $\phi : V \rightarrow \mathbb{R}^d, d \ll |V|$ which represents each vertex $v \in V$ as a point in a low-dimensional space \mathbb{R}^d .*

Here d is a parameter specifying the number of dimensions of our node representation.

Although recent methods show promising performance for various applications, graph embedding still has some challenges. First, many of these methods are not scalable to large graphs (scalability problem). Secondly, due to non-convex optimization, these approaches can easily get stuck at a bad local minima as the result of poor initialization (initialization problem). This may generate dissimilar representations for vertices within the same or similar neighborhood set. Also, many of these methods ignore the global structure in the graph.

These challenges have motivated us to use graph compression (summarization) algorithm that reduces the complexity and size of large graphs. We propose a proximity-based graph compression method that compresses the input graph to a smaller graph by incorporating the neighborhood similarity of its vertices into super-nodes. NECL compresses the graph by merging vertices with similar neighbors into super-nodes instead of random edge merging. NECL employs the embedding of the compressed graph to obtain the embedding of the original graph. This brings down the embedding cost and captures the global structure of the original graph without losing the locality kept in the super-nodes. In addition to reducing the graph's size for embedding, we get fewer pairwise relationships from random walks on a smaller set of super-nodes, which generates less diverse training data for the embedding part. All these facts improve efficiency while maintaining *similar* or *better effectiveness* compared to the

baseline methods. We then project the embedding of super-nodes to the original nodes.

1.2.2 MPool: Motif-Based Graph Pooling

In this work, we propose a model for graph classification, a fundamental task in graph learning that aims to predict labels for entire graphs by leveraging both node features and graph structures. The formal definition of the graph classification problem is as follows:

Graph Classification: We denote a graph as $G(V, A, X)$ where V is the node-set, $A \in \mathbb{R}^{N \times N}$ is the adjacency matrix, and $X \in \mathbb{R}^{N \times d}$ is the feature matrix with d dimensional node feature and N is the number of nodes in the graph. We denote a graph collection as (\mathcal{G}, Y) where $\mathcal{G} = \{G_0, G_1, \dots, G_n\}$ with G_i 's are graphs and Y is the set of the graph labels. In this paper, we work on the graph classification problem, whose goal is to learn a function $f : \mathcal{G} \rightarrow Y$ to predict the graph labels with a graph neural network in an end-to-end way.

GNN models apply different graph pooling methods to learn graph-level representation for graph classification. However, existing pooling methods often fail to incorporate higher-order graph structures, which play a crucial role in graph mining tasks such as classification [17], and community detection [18]. There are different technical tools that have been proposed to model higher-order graph structures [19] such as hypergraphs [20], simplicial complexes [21], and motifs [22]. Between these tools, motifs (graphlets) are small, frequent, and connected subgraphs widely used to measure the connectivity patterns of nodes [23, 24]. Despite their effectiveness in various graph mining tasks, including classification [17], and community detection [25], motifs have yet to be explored in graph pooling operations..

In this work, to address these problems, we propose a multi-channel Motif-based Graph

Pooling method named (MPool) that captures the higher-order graph structure with motif and also local and global graph structure with a combination of selection and clustering-based pooling operation. We utilize motifs to model the relation between nodes and use this model for message passing and pooling in GNN. We develop three motif-based graph pooling models (MPool_S, MPool_C, and MPool_{comb}): selection and clustering based and combined model (MPool_{comb}). For the selection-based graph pooling model, we design a node ranking model considering motif-based relations of nodes. For clustering-based graph pooling, we design a motif-based clustering model that learns a differentiable soft assignment based on learned embedding from the convolution layer. After learning the assignment matrix, we grouped the nodes in the same cluster to create a coarsened graph. In our Combined model, we combine both selection and clustering-based methods into one model that learns both local and global graph structures. All models incorporate higher-order graph structure in graph representation with taking motifs into consideration while pooling.

1.2.3 Graph Contrastive Learning via Structure-Aware Graph Compression

In this work, we propose a graph contrastive learning model for graph classification, following the same problem formulation outlined in the second project. Contrastive learning, a self-supervised technique, has been integrated into GNNs to enhance graph representation learning [26, 27, 28]. It aims to maximize feature consistency by contrasting multiple semantically coherent augmented views of the same graph [29]. However, existing methods generate these views through random node and edge dropping, leading to structural information loss and focusing primarily on low-level graph features.

To overcome these limitations, we introduce a structure-aware graph contrastive learning model via graph compression. Our approach leverages graph compression, a well-established technique in graph mining that reduces complexity while preserving global structure, widely used in tasks like community detection and graph embedding. To our knowledge, no prior graph contrastive learning model has utilized graph compression for view construction.

Our model employs two view encoders: a K-core view encoder and a K-truss view encoder. In the K-core view encoder, we first extract K-core subgraphs, compress them into super nodes, and learn representations from both the original and compressed graphs. The K-truss view encoder follows a similar process, using K-truss subgraphs instead. We optimize these representations via contrastive learning and combine them to generate the final graph representation for classification.

This structure-aware contrastive framework effectively preserves both local and global structural properties, addressing key shortcomings of existing methods while enhancing graph classification performance.

1.2.4 DyGCL: Dynamic Graph Contrastive Learning For Event Prediction

In this project, we tackle the event prediction problem by modeling historical event-related articles as a sequence of dynamic graphs, where each graph captures contextual information at a specific timestamp. Nodes represent words in the articles, and edges indicate word co-occurrence within a fixed-size window. Specifically, one graph is created for each day, and data from k consecutive days is represented as the dynamic graph and used to predict whether an event will occur on $(k + 1)$ th day.

Here, we first define dynamic graphs and then event prediction via dynamic graphs.

Definition 1 (Dynamic Graph). *A dynamic graph \mathbb{G} is defined as a series of T discrete snapshots denoted as $\mathbb{G} = \{G_1, G_2, \dots, G_T\}$, where G_t represents the graph at timestamp t . Each G_t has an adjacency matrix A_t showing the relation between nodes at time t*

Definition 2 (Event Prediction). *Given a training dataset D , where each sample is represented as $D[\mathbb{G}_i]$ for $i = 1, 2, 3, \dots, m$ and each and each \mathbb{G} is a dynamic graph $\mathbb{G} = \{G_1, G_2, \dots, G_T\}$ with initial node feature matrix $X \in R^{N \times d}$ where N is the number of nodes with d dimension at time t , our goal is to learn a graph encoder that maps the input dynamic graph into vector representation and use this representation to predict the future event \hat{y} at time $T + 1$.*

Dynamic GNNs [30, 31, 32, 33] have been developed as extensions to standard GNNs to capture the dynamic patterns within dynamic graphs and used for event prediction [34, 35, 30]. Although these models show promising results, they primarily focus on node-level representation capturing the local structure and overlooking the global structure of the temporal graph. On the other hand, both node-level and graph-level representations should be utilized in a consistent way to capture the dynamic patterns of input graphs for more accurate event prediction.

Addressing these aforementioned challenges, in this paper, we introduce a **Dynamic Graph Contrastive Learning (DyGCL)** method that learns dynamic graph representation for event prediction. In this model, we utilize two novel encoders: a local view encoder and a global view encoder. Our local view encoder captures the local neighborhood structural

information using a Dynamic Graph Convolutional Network, propagating node features over time and applying pooling at the final step to obtain a graph representation for the entire dynamic graph. Meanwhile, our global view encoder captures hierarchical graph structures via dynamic Graph Pooling and aggregates these representations using an LSTM module. Contrastive learning is applied to align the local and global representations, which are then fused via an attention mechanism for event prediction. By integrating both encoders, DyGCL learns a dynamic graph representation that effectively captures both local and global dynamic structural patterns of the input graphs

CHAPTER 2

RELATED WORKS

2.1 Network Embedding

Network embedding plays a significant role in network data analysis, and it has received huge research attention in recent years. Previous researchers consider the graph embedding as a dimensionality reduction [36], such as PCA [37] that captures linear structural information and LE (locally linear embeddings) [38] that preserves the global structure of non-linear manifolds. While these methods are effective on small graphs, scalability is the major concern with them being applied to large-scale networks with billions of vertices, since the time complexity of these methods is at least quadratic in the number of graph vertices [1, 39]. On the other hand, recent approaches in graph representation learning focus on the scalable methods that use matrix factorization [40, 41] or neural networks [42, 43, 44, 45]. Many of these aim to preserve the first and second-order proximity as a local neighborhood with path sampling using short random walks such as DeepWalk and Node2vec [2, 3, 4, 5, 46]. Some recent studies aim to preserve higher-order proximity [47, 48]. In addition to these, some recent works integrate contents to learn better representations [49]. While some studies use network embedding on node and graph classification [6, 48, 50], some others use it on graph clustering [51, 49, 52].

DeepWalk [6] is the pioneering work that uses the idea of word representation learning in [53, 54] for network embedding. While vertices in a graph are considered as words, neighbors are considered as their context in natural language. A graph is represented as a

set of random walk paths sampled from it. The learning process leverages the co-occurrence probability of the vertices that appear within a window in a sampled path. The Skip-gram model is trained on the random walks to learn the node representation [53, 54].

Optimization of a non-convex function in these methods could easily get stuck at a bad local minima as the result of poor initialization. Moreover, while preserving local proximities of vertices in a network, they may not preserve the global structure of the network. As a solution to these issues, a multi-level graph representation learning paradigm has been proposed [48, 55, 56, 57]. HARP, is proposed in [48] as a graph preprocessing step to get better initialization vectors. In this approach, related vertices in the network are hierarchically combined into super-nodes at varying levels of coarseness. After learning the embedding of the coarsened network with a state-of-the-art graph embedding method, the learned embedding is used as an initial value for the next level. The initialization with the embedding of the coarsened network improves the performance of the state-of-the-art methods. One of the limitations of this method is that multi-level compressing and learning results in significant compression and embedding cost. Random edge compressing may put dissimilar nodes into the same super-node which makes their representation similar.

As a more efficient solution, MILE [56] performs multi-level network embedding on large graphs using graph coarsening and refining techniques. It compresses the graph repeatedly based on Structural Equivalence Matching (SEM) and Normalized Heavy Edge Matching (NHEM). After learning the embedding of the compressed graph, they refine it efficiently through a novel graph convolution neural network to get the embedding of the original

graph. This way, it receives embedding for large scale graphs in an efficient and effective way. More recently, GraphZoom [57] proposes a multi-level spectral approach to enhancing both the quality and scalability. It performs graph fusion to generate a new graph that effectively encodes the topology of the original graph and the node attribute information. Then they apply spectral clustering methods to merge the nodes into super-nodes with the aim of retaining the first few eigenvectors of the graph Laplacian matrix. Finally, after getting the embedding of the compressed graph, they refine it by applying projection on it to get the original graph embedding. LouvainNE [55] applies the Louvain clustering algorithm recursively to partition the original graph into multiple subgraphs and construct a Hierarchy partition of the graph, which is represented as a tree. Then they generate different meta-graph from the tree and apply the baseline method i.e., DeepWalk Node2vec. After getting the embedding from different meta-graph, they combine these embeddings to find the final embedding. They use a parameter to regulate the weights of different embedding for combining.

2.2 Graph Neural Networks

In general, GNNs can be categorized via two different approaches: spectral and non-spectral. In the spectral approach [7, 58, 59], convolution operation for the graph is defined by Fourier transform and graph Laplacian. Spectral methods have a basis-dependent problem in that they are not generalizable to graphs with different structures. Using this approach, a model trained on one graph structure cannot be effectively transferred to another. To overcome

this problem, non-spectral methods design convolution operations by directly aggregating neighbor nodes' information [8, 9]. These methods are faster and more easily generalizable to graphs of different structures. GraphSAGE [8] is one of the popular methods of non-spectral methods where it samples a fixed-sized neighborhood and aggregates its information. Moreover, Graph attention network (GAT) [9] uses the attention mechanism to aggregate neighbor information where it gives different weights to neighborhood representation.

2.3 Graph Pooling

Graph pooling operation on GNN is developed for learning hierarchical graph representation learning. There are two different classes of graph pooling in the literature: global pooling and hierarchical pooling. Global pooling methods [60, 61] summarize entire graph in one step. For example, Set2Set [60] implemented the global pooling using LSTMs where it uses iterative content-based attention to calculate the importance of nodes. Another method, SortPool [61] sorts the nodes based on their embedding and feeds them to the next layers. However, the global pooling method does not learn graph representation hierarchically, which is very important for graph-level representation.

Hierarchical pooling methods learn graph representation hierarchically and capture the local substructures of graphs. There are two different hierarchical pooling methods in the literature: node cluster pooling and selection pooling. Node cluster pooling methods [62, 63, 64] do the pooling operation by calculating the cluster assignment matrix using node features and graph topology. After calculating the cluster assignment matrix, they build

the coarse graph by grouping the nodes on the same cluster. For example, while DiffPool[62] calculates the cluster assignment matrix using a graph neural network, MinCutPool[16] calculates the cluster assignment matrix using a multi-label perception.

Selection-based pooling methods [14, 15, 65, 66] compute the importance scores of nodes and select top k nodes based on their scores and drop other nodes from the graph to create the pooled graph. For example, while gPool [14] calculates the score using node feature and a learnable vector, SAGPool [15] uses an attention mechanism to calculate the scores. SUGAR [67] uses subgraph neural network to calculate the score and select top- K subgraph for pooling operation. They use BFS algorithm to sample the subgraph from the input graph. Selection-based pooling methods are generally more memory efficient as they avoid generating dense cluster assignments and select nodes based on their scalar projection values on a trainable vector. However, both types of methods mainly use the node features and normal adjacency matrix with GNNs for pooling operation. On the other hand, higher-order structures, i.e., motifs, are also important for the graph-based task. In our proposed method, we use motif structure for the pooling operation to integrate higher-order graph structure information during learning graph representation.

2.4 Event prediction

In the area of event prediction, there is a broad spectrum of real-world applications that have been explored, such as predicting political events [68, 35], forecasting election results [69], traffic analysis, trends in the stock market [70], and tracking disease outbreaks [71, 72, 73, 74,

75]. Initial approaches in this field often utilized traditional machine learning techniques. For instance, the use of linear regression [76] has been noted for its effectiveness in predicting the timing of future events by analyzing social media data frequency and volume. Furthermore, more complex methods involving paragraph embeddings [77] and the utilization of topic-specific keywords [78] have been investigated.

The advancements in event prediction have seen a notable shift from basic machine learning techniques to the adoption of sophisticated deep learning methods, particularly focusing on the temporal aspects of information diffusion. The application of deep neural networks, notably by Ma et al. [79, 80, 81] using recurrent neural networks, and Liu et al. [82] combined convolutional and recurrent neural networks, exemplifies this advancement. These models excel in understanding the progression and nuances of events as they unfold, particularly in digital and social media contexts. Xia et al. [83] further contributed to this field with a model focused on the detailed detection and segmentation of evolving event states, offering a more granular perspective on event dynamics. Despite these technological advancements, a common limitation persists in the predominant focus on semantic information of input data, potentially overlooking other vital aspects, such as contextual and non-semantic factors, which are crucial for a comprehensive and accurate prediction model.

2.4.1 Graph-based event prediction

Graph-based event prediction has seen significant advancements with the implementation of GNNs, which excel in structuring the relationships among words or entities into graphs. The DynamicGCN [34] exemplifies this approach by applying a static Graph Convolutional

Network (GCN) to input graphs of each time epoch, initializing node features for each epoch with embeddings from the previous one, and starting with word embeddings at the initial time. This model updates node embeddings via a temporal layer, integrating the initial and GCN-derived embeddings, and employs a readout layer to convert these embeddings into a fixed-size vector for event prediction. Similarly, DyGED[35] focuses on learning graph-level representations for each epoch’s graph, updated using a recurrent neural network (RNN). This model applies a static GNN and global pooling for learning representations but does not utilize hierarchical graph pooling, which is crucial for capturing the graph’s global structure. DyGED then merges all snapshot graph embeddings into a single vector representation for event prediction. While both models mark important contributions to graph-based event prediction, leveraging GNNs to capture complex event dynamics, they also highlight areas for potential improvements, such as the integration of more advanced graph pooling techniques and enhanced processing of temporal information for more robust and accurate predictions.

CHAPTER 3

Proximity-based Compression for Network Embedding

3.1 Introduction

This paper is published in *Frontiers in Big Data* [84]. In this work, we propose a novel Network Embedding method, NECL, to generate embedding more efficiently or effectively.

We summarize our contributions as follows,

- New proximity-based graph compressing method: Based on the observation that vertices with similar neighborhood sets get similar random walks and eventually similar representation, we merge these vertices into super-nodes to get a smaller compressed graph that preserves the proximity of nodes in the original large graph.
- Efficient embedding without losing effectiveness: We do random walks and embedding on the compressed graph, which is much smaller than the original graph, efficiently. This method has similar effectiveness with baseline methods by preserving the global and local structure of the graph in the compressed graph.
- Effective embedding without decreasing efficiency: We use the embedding obtained from the compressed graph as initial vectors for the original graph embedding. This combines the global and local structure of the graph and improves the effectiveness. Embedding of a small compressed graph does not take much time with respect to original graph embedding, so it will not increase the embedding time significantly.

- Generalizable: NECL is a general meta-strategy that can be used to improve the efficiency and effectiveness of many state-of-the-art graph embedding methods. We report the results for DeepWalk Node2vec and LINE.

3.2 Methodology

While a desirable network embedding method for real-world networks should preserve the local proximity between vertices and the global structure of the graph, it should also be scalable for large networks. This section presents our novel network-embedding models, NECL and NECL-RF, which satisfy these requirements. We extend the idea of the graph compressing layout to network representation learning methods. After giving some preliminary information, we explain our proximity-based compression method and how we combine compression with network embedding.

In this paper, we consider an undirected, connected, simple graph $G = (V; E)$ where V_G is the set of vertices, and $E \subseteq \{V \times V\}$ is the set of edges. The set of neighbors for a given vertex $v \in V$ is denoted as $N_G(v)$, where $N_G(v) = \{u | u \in V : (u, v) \in E\}$. We now define what a compressed graph is.

Definition 2 (Compressed graph). *A compressed graph of a given graph $G = (V; E)$ is represented as $CG = (S; M)$ where $S = (V_S; E_S)$ is the graph summary with super-nodes V_S and super-edges E_S and M is a mapping from each node v to its super-node in V_S . A super-edge $E = (V_i; V_j)$ in E_S represents the set of all edges between vertices in the super-nodes V_i and V_j .*

Considering a graph G , we define adjacency matrix A that is symmetric for undirected graphs. For an unweighted graph, we have $A_{ij} = 1$ if and only if there exists an edge from v_i to v_j , and $A_{ij} = 0$ otherwise. For a graph with adjacency matrix A , we can define the diagonal matrix, known as degree matrix, as $D_{ij} = \sum_k A_{ik}$ if $i = j$, and $D_{ij} = 0$ otherwise.

3.2.1 Neighborhood similarity-based graph compression

The critical problem for graph compressing with preserving local structures of the graph is to identify vertices that have similar neighborhoods accurately, so they are more likely to have similar representation. In this section, we discuss how to select vertices to merge into super-nodes.

3.2.1.1 Motivation

The motivation of our method is that if two vertices have many common neighbors, many embedding algorithms that preserve local neighborhood information will give similar representations to them. This comes from our following observation that if two vertices, v_i and v_j , of a graph have many common neighbors, they also have similar transition probabilities to other vertices. This means that if A_i and A_j are similar, their transition probability vectors, $T_i = A_i * D_{ii}^{-1}$ and $T_j = A_j * D_{jj}^{-1}$, will be similar as well. Hence they have similar neighborhoods and get similar neighborhood sets from random walks, and as a result, they get similar representations from the learning process.

For example, in the toy graph in Figure 3.1, the neighbor sets of the nodes a and b are the same. Hence, their transition probabilities to the other neighbor vertices are also the

same, i.e., $p(n_i|a) = p(n_i|b) = 1/4$ for all $i \in \{1, 2, 3, 4\}$. Starting on either a or b yields the same or very similar walks, so they have the same or similar representation. Therefore, instead of walking and learning representations for both a and b , it is enough to learn one for both of them. For this, we can merge this node pair (a, b) into one super-node ab . Transition probabilities of this super-node to neighbors of a and b are still the same with a and b , i.e., $p(n_i|ab) = 1/4$ for all $i \in \{1, 2, 3, 4\}$. When we obtain the representation of the super-node ab , we can project it as the representation of each node in this pair. Merging these vertices keeps the preservation of the first and second-order proximity. Thus, this does not affect the results of walking and learning, but it increases efficiency.

Furthermore, compressing may change the transition probability of neighbors of compressed vertices since the number of their neighbor decrease. As a result, the transition probability of each neighbor changes. For example, in the toy graph in Figure 3.1-(a), while the transition probability from n_1 to its neighbors is $\frac{1}{|N(n_1)|}$, after compressing, it becomes $\frac{1}{|N(n_1)|-1}$ since the number of neighbors decrease by one. In order to avoid this problem, we assign weights to edges of super-nodes based on the number of merged edges within the compression. For example, the super-edge between super-node ab and n_1 includes 2 edges which are (a, n_1) and (b, n_1) . Therefore, the weight of the super-edge (ab, n_1) should be 2.

In a real-world graph, it is not expected to have too many vertices sharing exactly the same neighborhood. However, for many graph mining problems, such as node classification and graph clustering, if two vertices share many common neighbors, they are expected to be in the same class or cluster, although their neighbor sets are not completely the same.

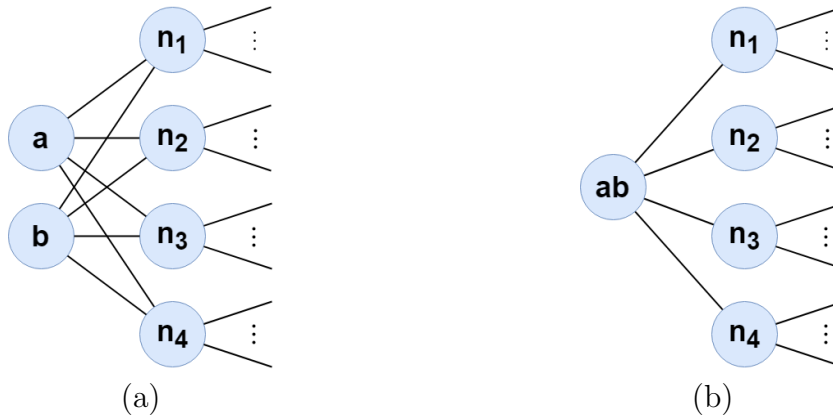


Figure 3.1 Example of graph compressing: a and b are merged into super-node ab connected to the neighbours of both a and b.

Hence, we expect to have similar feature vectors for the vertices in the same class/cluster after embedding. From these observations, we can also apply the same merge operation on these vertices. Following the same idea in the example above, if neighbors of two vertices are similar (but not exactly the same), we can merge them into a super-node and learn one representation for all. While we can project this super-node embedding to original vertices and use the same representation for both, we can also update them in the refinement phase to embed the difference of them into their representation.

3.2.1.2 Proximity Based Graph Compressing

In this section, We define our graph compressing algorithm formally.

For a given graph G , if a set of vertices n_1, n_2, \dots, n_r in V_G have similar neighbors, we merge these vertices into one super-node $n_{12\dots r}$ to get a smaller compressed graph $G'(V_{G'}, E_{G'})$. To decide which vertices to merge, we define the *neighborhood similarity* based on the transition probability. Before defining the neighborhood similarity, we first show that cosine similarity

between transition probabilities of two vertices u and v , T_u and T_v , are determined by the number of their common neighbors.

Theorem 1. *Let T be the 1-step transition probability matrix of vertices V in a graph G and let $u, v \in V$. Let $N(u)$ and $N(v)$ be the neighborhood sets of u and v and T_u and T_v be the transition probability vectors from u and v to other vertices. Then the similarity between T_u and T_v is proportional to the number of common neighbors, $|N(u) \cap N(v)|$.*

Proof. The cosine similarity between T_u and T_v is defined by

$$\text{sim}(T_u, T_v) = \frac{\sum_i T_{ui}T_{vi}}{\|T_u\| \|T_v\|} \quad (3.1)$$

By definition of T , we have $T_u = \frac{A_u}{|N(u)|}$ and $T_v = \frac{A_v}{|N(v)|}$. Furthermore, we have

$$\|T_u\| = 1/\sqrt{|N(u)|}, \quad \|T_v\| = 1/\sqrt{|N(v)|}$$

and

$$\sum_i A_{ui}A_{vi} = |N(u) \cap N(v)|.$$

Hence, if we plug in these into the Equation (1), we get

$$\begin{aligned}
sim(T_u, T_v) &= \frac{\sum_i T_{ui} T_{vi}}{\|T_u\| \|T_v\|} = \frac{\sum_i \frac{A_{ui}}{|N(u)|} \frac{A_{vi}}{|N(v)|}}{\frac{1}{\sqrt{|N(u)|}} \frac{1}{\sqrt{|N(v)|}}} \\
&= \frac{\frac{1}{|N(u)||N(v)|} |N(u) \cap N(v)|}{\frac{1}{\sqrt{|N(u)||N(v)|}}} \\
&= \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)||N(v)|}}.
\end{aligned}$$

Therefore,

$$sim(T_u, T_v) \propto |N(u) \cap N(v)|$$

This finalizes the proof. □

From Theorem 1, we see that the similarity of transition probabilities from two vertices to other vertices depends on the similarity of their neighbors. Therefore, for the compressing, we define the neighborhood similarity between two vertices as follows.

Definition 3. (*Neighborhood similarity*) Given a graph G , the neighborhood similarity between two vertices u, v is given by

$$Nsim(u, v) = \frac{2|N(u) \cap N(v)|}{|N(u)| + |N(v)|} \quad (3.2)$$

In order to normalize the effect of high degree vertices, we divide the number of common neighbors by degree of vertices. The neighborhood similarity is between 0 and 1, where it is

Algorithm 1: Graph Compressing (G, λ) .

Input: $G(V_G, E_G)$, similarity threshold λ
Output: $S(V_{G'}, E_{G'}, W_E)$, mapping \mathbf{M}
 \mathbf{M} is a mapping from super-node to original node
 $S \leftarrow G$
 $NSQ \leftarrow \emptyset$
for $v \in V_G$ **do**
 for $u \in N_G(v)$ **do**
 for $k \in N_G(u)$ **do**
 Compute Neighborhood similarity between v and k as $NSim(v, k)$
 $NSQ \leftarrow NSQ \cup (v, k)$
 end
 end
end
for $(v, k) \in NSQ$ **do**
 if $NSim(v, k) > \lambda$ **then**
 Merge them into a super-node $s_{v,k}$
 $M(s_{v,k}) \leftarrow v; M(s_{v,k}) \leftarrow k$
 Delete v and k from S and add $s_{v,k}$ into S .
 end
 for $ng \in N_S(v)$ **do**
 add edge between $s_{v,k}$ and ng
 $w(ng, s_{v,k}) = w(ng, v)$
 end
 for $ng \in N_S(k)$ **do**
 add edge between $s_{v,k}$ and ng if there is no
 $w(ng, s_{v,k}) = w(ng, s_{v,k}) + w(ng, k)$
 end
end

0 when two vertices have no common neighbor and 1 when both have the same neighbors. According to the neighbor similarity, we merge vertices whose similarity value is higher than a given threshold value.

The neighborhood similarity-based graph compressing algorithm is given in Algorithm 1. It is clear that the vertices with a nonzero neighborhood similarity are 2-step neighbors. Therefore, we do not need to compute the similarity between all pairs of vertices. Instead, we just need to compute the similarity between vertices and their neighbors' neighbors. For

each node $v \in V_G$, we compute the similarity between v and each k as neighbors of neighbors (line 3-10). Then, we check the similarity value of all pairs (u, k) in the list and if it is higher than the given threshold λ (line 12), we merge u and k into a super-node $s_{u,k}$ (line 13). Then we delete edges of u and k and add edges between neighbors of u and k and the new super-node $s_{u,k}$ (line 17-24). We assign weights to the edges of super-nodes based on the number of merged edges within the compression. The threshold λ decides the trade-off between efficiency and effectiveness. If we use a larger value, it will merge a fewer number of vertices. On the other hand, if we use a smaller value, we merge more vertices, and as a side effect, we may merge some dissimilar vertices as well, which may result in an increase in efficiency but cause a decrease in accuracy. Note that since we use original neighborhood similarity, the order of merging does not affect the result, so we randomly select a node and check neighbors for compression. Furthermore, one super-node may include more than two vertices of the original graph. For example, if the similarity between the vertices x and y , $NSim(x, y)$, and the vertices y and z , $NSim(y, z)$, are both bigger than given threshold, we merge x and y in $s_{x,y}$ and then we merge $s_{x,y}$ and z into $s_{x,y,z}$. Therefore, during the merge operation, we check whether the node y is merged with another node, and if so, we get the super-node of the original node x .

3.2.2 Network Embedding

Our NECL framework is adaptable with any embedding method that preserves the neighborhood proximity of nodes, i.e., DeepWalk, Node2vec, and LINE. We get the embedding for the

original graph in two ways.

3.2.2.1 Network Embedding on Compressed Graph

Our main goal in this section is to *improve the efficiency* of the embedding problem while maintaining *similar effectiveness* with the baseline methods. For this goal, instead of embedding the original graph, we embed the compressed graph and employ this embedding for the original graph embedding.

We first start compressing the graph for a given similarity threshold, as explained in the previous section. Then we learn the embedding of super-nodes in the compressed graph. Next, we assign the representation of each super-node in the compressed graph as the representation of the corresponding vertices in each super-node and obtain the embedding of the original graph. Since the size of the compressed graph is much smaller than the original graph, the embedding will be more efficient. The details of our algorithm for network embedding on a compressed graph is given in Algorithm 2.

In the algorithm, after getting the weighted compressed graph S (line 1), we obtain the representation of super-nodes V_S as ϕ_s in the compressed graph with the provided network embedding algorithm (line 2). We apply any random walk-based representation learning algorithm on the compressed graph. We just need to apply weighted random walks to consider the edge weights. As the size of the compressed graph is smaller than the original graph, it is more efficient to get embeddings of super-nodes than single vertices. Finally, we assign the embedding of super-nodes to vertices according to the mapping M obtained

from the compression (line 3-7). While we may lose some local information with assigning the same representation to multiple vertices, we gain efficiency. Also, we may not need to get small differences between nodes for many applications, e.g. classification, as we see in Section 5.3.

3.2.2.2 Network Embedding with Refinement

Our main goal in this section is to *improve the effectiveness* of the embedding problem while still maintaining *similar efficiency* with the baseline methods. For this goal, we employ the embedding of the compressed graph as initialization to the original graph embedding and refine it.

When we compress a graph using the neighborhood similarity score, we can easily capture the global structure of the original graph. On a large original graph, the random walk may get stuck in a local neighborhood. As a result, the embedding method may not capture the global structure of the original graph. However, when we do the random walk on the compressed graph, it visits the globally similar neighbor nodes. Hence, we can capture the global proximity of the nodes. That is why, in this method, we first embed the compressed graph for a given similarity threshold to encode the original graph’s global structure in the representation as in Section 3.2.2.1. Then, for the embedding of the original graph, instead of starting with randomly initialized representations, which happens in the original embedding methods such as DeepWalk and Node2vec, we start with the representations obtained from the compressed graph. In the case of random representations, for example, two similar nodes

Algorithm 2: NECL: Network Embedding on Compressed Graph.

Input: $G(V_G, E_G)$, similarity threshold λ
Output: Representation $\phi(u)$ for all $v \in V_G$
 $S, M \leftarrow \text{GraphCompressing}(G, \lambda)$
 $\phi_S \leftarrow \text{WightedGraphEmbedding}(S)$
for $V_i \in V_S$ **do**
 for $v_j \in M(V_i)$ **do**
 $\phi(v_j) \leftarrow \phi_S(V_i)$
 end
end
 $\phi_{v_j} \leftarrow \text{Refinement}(\phi_{v_j}, G)$ // For Method 2 in 3.2.2.2

are likely to have two very different and distanced representations, hence, the optimization process may not provide an accurate representation, and this may decrease the quality or it may take a longer time to make them similar. However, initializing the representation using the compressed graph embedding provides global structure information as initial knowledge to the embedding. The original graph embedding updates this initial embedding with local information that may be lost with compressing. Therefore, final embeddings have better quality with integrating local and global information in one representation. In Algorithm 2, the original graph embedding is obtained in line 8 by refining the compressed graph embedding given as the initial representation.

3.3 Experiments

We do our experimental studies to compare our methods with different models in terms of efficiency and effectiveness. We evaluate the quality of embeddings through challenging multi-class and multi-label classification tasks on four popular real-world graph datasets. First, in Section 3.3.1, we present our model’s performance based on different parameters.

Then, we compare the results of our models with the results of HARP.

Datasets: We consider four real-world graphs¹, which have been widely adopted in the network-embedding studies. Two of them are single-label, which are Wiki and Citeseer, and two of them are multi-label datasets, which are DBLP and BlogCatalog (BlogC). In single-label datasets, each node in the datasets has a single label from multi-class values. In multi-label datasets, a node can belong to more than one class.

Baseline methods: To demonstrate that our methods can work with different graph embedding methods, we use three popular graph embedding methods, namely DeepWalk, Node2vec and LINE, as the baseline methods in our model. We combine each baseline method with our methods and compare their performance. We give a brief explanation of the baseline methods in Section ???. We named our first method as NECL, which uses a compressed graph embedding as the original graph embedding, and the second method as NECL-RF, which uses the compressed graph embedding as the initial vector for original graph embedding and refine it with the original graph.

Parameter Settings: For DeepWalk, Node2vec, NECL(DW), NECL(N2V), NECL-RF(DW) and NECL-RF(N2V), we set the following parameters: the number of random walks γ , walk length t , window size w for the Skip-gram model and representation size d . The parameter setting for all models is $\gamma = 40$, $t = 10$, $w = 10$, $d = 128$. The initial learning rate and final learning rate are set to 0.025 and 0.001 respectively in all models. Representation size for LINE is $d = 64$ for all model.

Classification We present our results and compare them with the baseline methods and

¹<https://linqs.soe.ucsc.edu/data>

also HARP in single-label and multi-label classification tasks. For the single classification task, the multi-class SVM is employed as the classifier, which uses the one-vs-rest scheme. For the multi-label classification task, we train a one-vs-rest logistic regression model with L_2 regularization on the graph embeddings for prediction. The logistic regression model is implemented with LibLinear [85].

For the evaluation, after getting embeddings for nodes in the graph, we use these embeddings as the features of the nodes. Then, we train a classifier using these features. To train the classifier, we randomly sample a certain portion of labeled vertices from the graph and use the rest of the vertices as the test data. To have a detailed comparison of methods, we vary our training ratio from 1% to 50% on the Citeseer, Wiki, and DBLP datasets and from 10% to 80% for BlogCatalog. To compare to other datasets, we use a larger portion of training data for the BlogCatalog dataset because the number of class labels of BlogCatalog is about 10 times that of other graphs.

We repeat the classification tasks ten times to ensure the reliability of our experiment and report the average macro F_1 and micro F_1 scores and embedding times of our models with different parameter. Since our focus is improving the efficiency of embeddings, we report the time for embedding and do not include compression time. However, as we explain in the methodology section, we just need to compute the similarity between vertices and their neighbors' neighbors and combine them into supernodes. Furthermore, the computation is not multi-level, just one-time computation. Therefore, the compression part does not have high complexity and it does not have an impact on efficiency. All experiments are performed

Table 3.1 Performance comparisons of NECL with baseline methods (BL)

| | | Macro F_1 | | | Micro F_1 | | | Time (s) | | |
|----------|-------------|-------------|-------|------------|-------------|-------|------------|----------|--------|-------------|
| | | NECL | BL | Gain% | NECL | BL | Gain% | NECL | BL | Gain% |
| Citeseer | DW | 0.434 | 0.408 | 6.4 | 0.469 | 0.440 | 6.6 | 9.26 | 16.21 | 42.9 |
| | N2V | 0.439 | 0.437 | 0.5 | 0.475 | 0.472 | 0.6 | 8.95 | 15.46 | 42.1 |
| | Line | 0.317 | 0.320 | -0.9 | 0.355 | 0.359 | -1.1 | 0.67 | 1.43 | 53.1 |
| Wiki | DW | 0.390 | 0.373 | 4.6 | 0.497 | 0.483 | 2.9 | 4.84 | 8.98 | 46.0 |
| | N2V | 0.349 | 0.348 | 1.0 | 0.489 | 0.490 | -0.2 | 9.41 | 19.10 | 50.7 |
| | Line | 0.355 | 0.369 | -3.8 | 0.517 | 0.518 | 0.2 | 1.28 | 3.81 | 66.4 |
| DBLP | DW | 0.625 | 0.603 | 3.6 | 0.656 | 0.635 | 3.3 | 39.97 | 93.96 | 57.5 |
| | N2V | 0.626 | 0.624 | 0.3 | 0.657 | 0.653 | 0.6 | 75.81 | 175.31 | 56.8 |
| | Line | 0.595 | 0.593 | 0.3 | 0.649 | 0.645 | 0.6 | 9.94 | 28.58 | 65.2 |
| BlogC | DW | 0.246 | 0.245 | 0.4 | 0.388 | 0.387 | 0.2 | 71.7 | 99.3 | 27.7 |
| | N2V | 0.252 | 0.251 | 0.3 | 0.391 | 0.389 | -0.5 | 1247 | 1628 | 23.4 |
| | Line | 0.215 | 0.219 | -1.8 | 0.369 | 0.373 | -1.1 | 99.35 | 126.65 | 21.6 |

on a server running Ubuntu 14:04 with 4 Intel 2.6 GHz ten-core CPUs and 48 GB of memory.

All data and code are publicly available through this link: <https://github.com/esraabil/NECL>.

3.3.1 Analysis of NECL

We present our results in Table 3.1 and 3.2. For the similarity threshold $\lambda < 0.5$, the compressed graph becomes very small and gives low macro F_1 and micro F_1 scores. Since it also merges more nodes into super-nodes with a low similarity value, this may result in information loss on the graph. Hence, we set the cutting point of compression at $\lambda = 0.5$. Moreover, to see the effect of the similarity threshold value λ on the compression and accuracy, we vary it from 0.45 to 1. We present the macro F_1 and micro F_1 scores with respect to the fraction of labeled data in Figures 3.2, 3.3, 3.4, 3.5 and embedding times in Figure 3.6. We also report the number of edges and vertices in the compressed graph with respect to similarity threshold λ on Figure 3.7 to see the effectiveness of the graph compression algorithm.

Table 3.2 Compression ratio with the similarity threshold $\lambda = 0.5$.

| | $ V $ | | | $ E $ | | |
|-----------------|------------|----------|-------------|------------|----------|-------------|
| | Compressed | Original | Ratio % | Compressed | Original | Ratio % |
| Citeseer | 1427 | 2708 | 47.3 | 5236 | 10858 | 51.8 |
| Wiki | 1060 | 2405 | 55.9 | 8584 | 23192 | 63 |
| DBLP | 8824 | 27199 | 69.9 | 32984 | 133664 | 75.3 |
| BlogC | 8507 | 10312 | 17.5 | 543872 | 667966 | 18.6 |

Gain on baseline methods: For all datasets, we present macro F_1 and micro F_1 scores for single and multi-label classification tasks and embedding time in Table 3.1 and compression ratio for edge and vertices in Table 3.2. We use 5% training ratio of labeled vertices for Citeseer, Wiki, and DBLP and 40% training ratio for BlogCatalog. As we see from Table 3.1, for DeepWalk, there is a significant gain on macro and micro F_1 in addition to gain on efficiency on Citeseer, Wiki, and DBLP. For Node2vec and LINE, while there is a significant gain on total embedding time as efficiency, there is no (significant) difference between NECL and baseline methods on macro F_1 and micro F_1 . For LINE, we have a higher gain on time for all datasets.

For DBLP, gains of embedding time are much higher than other datasets. On the other hand, for BlogCatalog, gains of embedding times are less with respect to other datasets. As we see from the tables, 1 and 2, the gain of embedding time depends on the compression ratio of the number of edges and vertices. With compression, the number of vertices and edges for DBLP decrease from 27199 to 8824 (70%) and from 133664 to 32984 (75%), respectively. Therefore, embedding becomes more efficient with better or the same accuracy. For BlogCatalog, the compression ratio is lower than the others, around 18%; therefore, the time gain is also lower. The reason for this is that, in DBLP, vertices have many common neighbors, so the neighborhood similarity is higher, and this results in more compression.

On the other hand, in BlogCatalog, vertices have less common neighbors and so a lower similarity, and this results in less compression. We can conclude that while the gain in the effectiveness of our method depends on the baseline method, the gain in the efficiency of our method depends on the characteristics of the dataset.

Detailed Analyses: We compare the performance of NECL framework for different similarity threshold values λ that results in different compression ratios with the performance of the baseline methods. Macro F_1 and micro F_1 scores on different datasets are given on Figures 3.2, 3.3, 3.4, 3.5 for Citeseer, Wiki, DBLP and BlogCatalog datasets, respectively. We observe that for $\lambda > 0.45$, macro F_1 , and micro F_1 scores for NECL are similar with or higher than baseline methods across all datasets except Citeseer. For $\lambda \leq 0.45$, the quality of embedding decreases dramatically and so does the accuracy of classification. The results for Citeseer depend on the baseline methods. While $\lambda = 0.45$ gives better accuracy for DeepWalk and Node2vec, it gives worse for LINE.

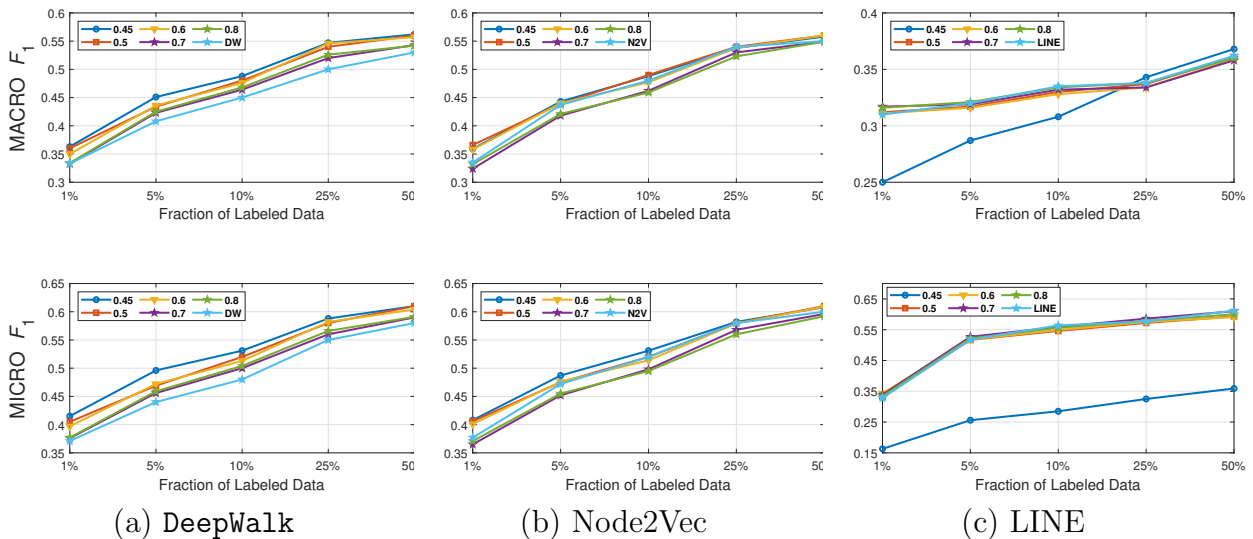


Figure 3.2 Detailed classification results on Citeseer.

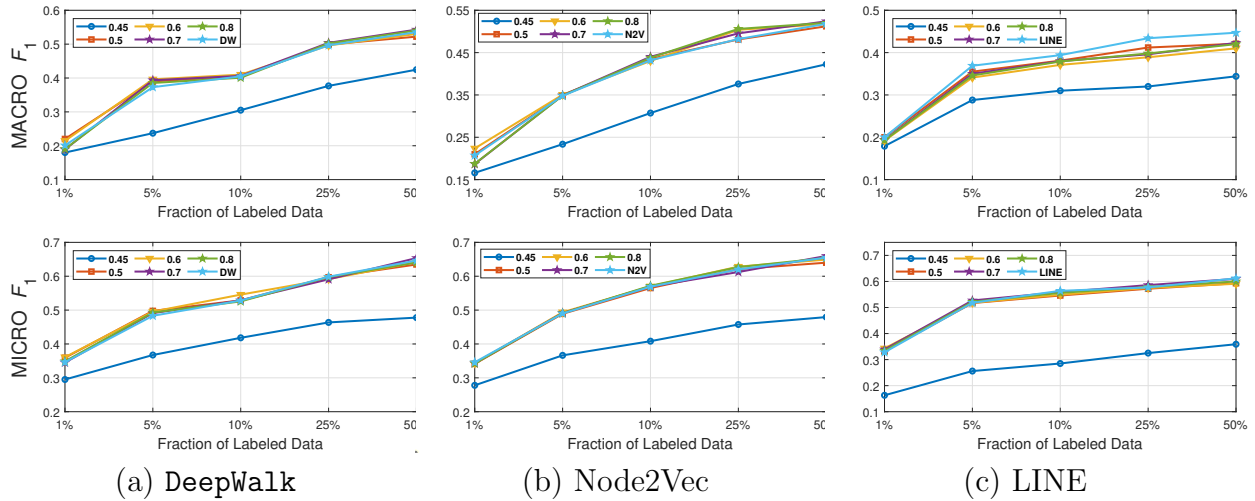


Figure 3.3 Detailed classification results on Wiki.

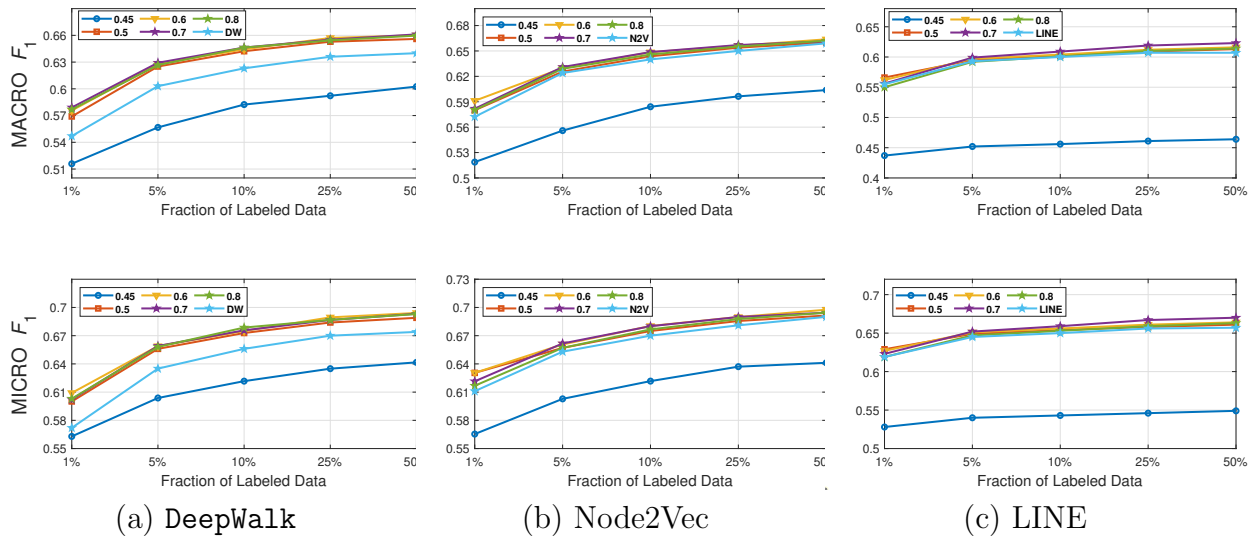


Figure 3.4 Detailed classification results on DBLP.

In addition to the macro F_1 and micro F_1 scores, we also report the embedding time and compression ratio for different similarity threshold values λ in Figure 3.6 and Figure 3.7. From the figures, we see that NECL takes significantly less time compared to the baseline method. As expected, for a lower threshold value λ , the compression ratio increases, and we get a smaller compressed graph and so the embedding time decreases. As BlogCatalog

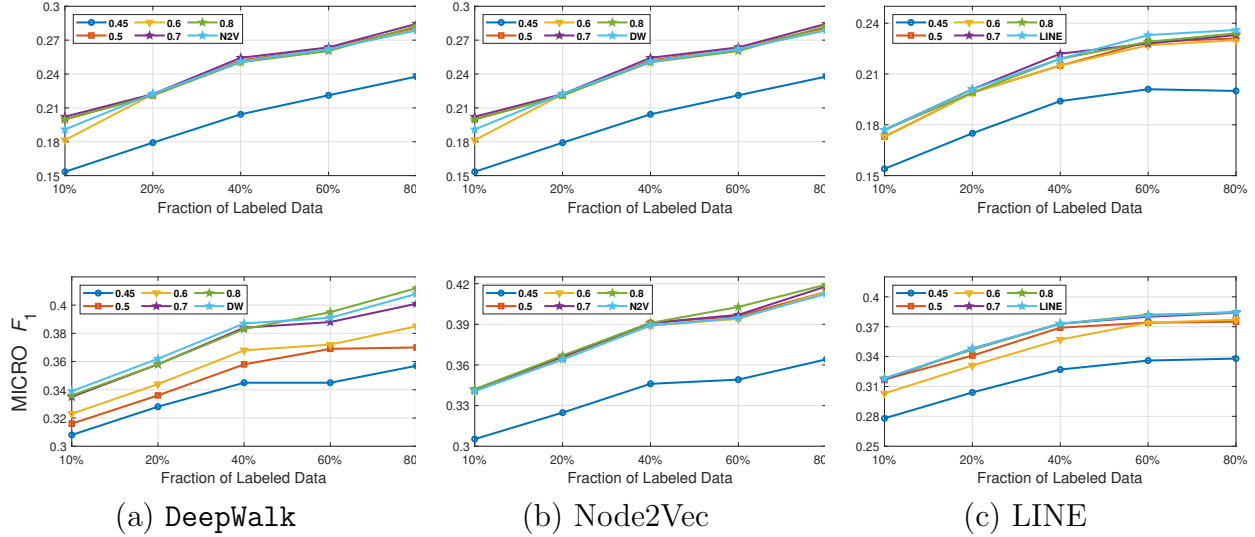
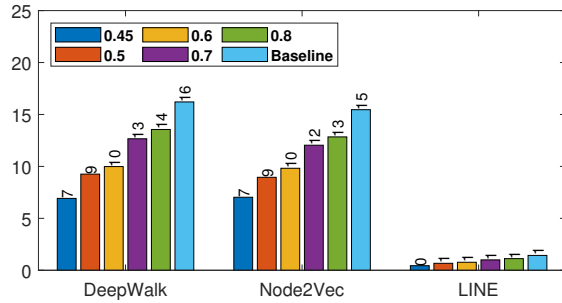


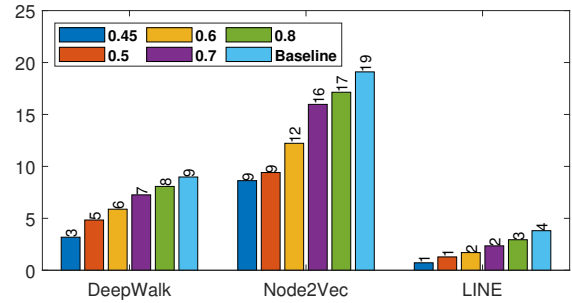
Figure 3.5 Detailed classification results on BlogCatalog.

has a lower compression ratio, the embedding time is less for all three baseline methods. We observe that there is a linear relation between λ and the number of vertices and edges until $\lambda = 0.5$. After this point, graph sizes change dramatically for smaller λ for Citeseer, Wiki, and DBLP, but the decrease is slow for BlogCatalog until $\lambda = 0.7$. One of the reasons for this situation in BlogCatalog is that the sizes of the neighbor sets for some vertices are very large, and it is not easy to get higher similarity for a larger set. For example, for two vertices with 15 edges, 10 common neighbors can be considered to have a higher similarity. On the other hand, two vertices with 150 edges should have 100 common neighbors to get the same similarity value, which is not very common.

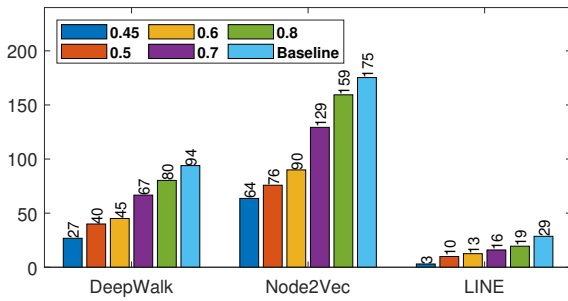
From these detailed analyses, we observe that smaller λ results in smaller compressed graph. As a result, embedding becomes more efficient. However, for $\lambda \leq 0.45$, we start to lose critical information about the graph, hence, while efficiency increases, effectiveness decreases dramatically. As a solution to this problem, we refine our results with our second



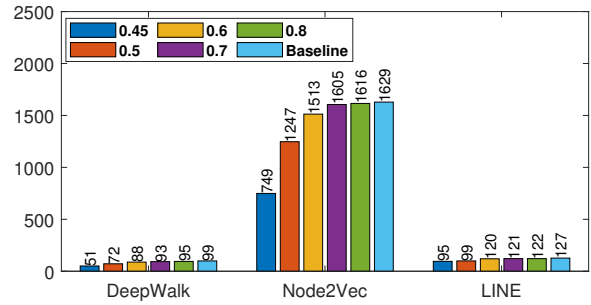
(a) Citeseer



(b) Wiki

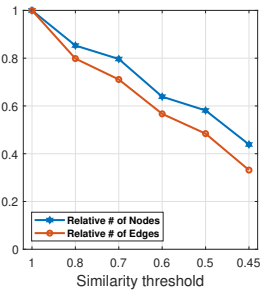


(c) DBLP

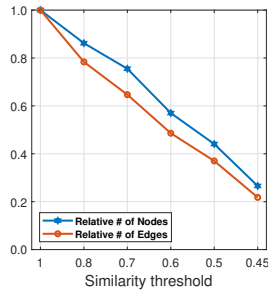


(d) BlogCatalog

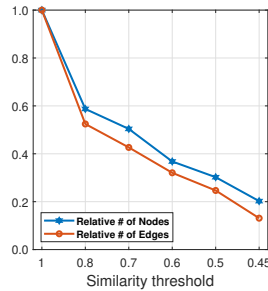
Figure 3.6 Run time analyses for different similarity threshold values λ



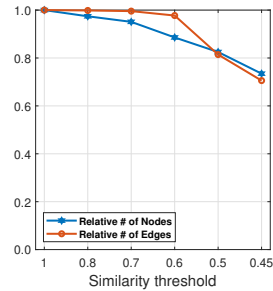
(a) Citeseer



(b) Wiki



(c) DBLP



(c) BlogCatalog

Figure 3.7 The ratio of vertices/edges of the compressed graphs to that of the original graphs.

method, NECL-RF.

Table 3.3 Performance comparisons of all methods

| | Citeseer | | Wiki | | DBLP | | BlogCatalog | |
|-----------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| | Macro F_1 | Micro F_1 | Macro F_1 | Micro F_1 | Macro F_1 | Micro F_1 | Macro F_1 | Micro F_1 |
| Baseline (DW) | 0.408 | 0.440 | 0.373 | 0.483 | 0.603 | 0.635 | 0.245 | 0.387 |
| HARP (DW) | 0.422 | 0.453 | 0.366 | 0.483 | 0.612 | 0.644 | 0.253 | 0.390 |
| NECL (DW) | 0.434 | 0.469 | 0.390 | 0.497 | 0.625 | 0.656 | 0.246 | 0.388 |
| NECL-RF (DW) | 0.422 | 0.457 | 0.408 | 0.549 | 0.617 | 0.649 | 0.285 | 0.405 |
| Baseline (N2V) | 0.437 | 0.472 | 0.348 | 0.490 | 0.624 | 0.653 | 0.251 | 0.389 |
| HARP (N2V) | 0.432 | 0.466 | 0.352 | 0.492 | 0.626 | 0.656 | 0.259 | 0.394 |
| NECL (N2V) | 0.439 | 0.475 | 0.349 | 0.489 | 0.626 | 0.657 | 0.252 | 0.391 |
| NECL-RF (N2V) | 0.430 | 0.464 | 0.372 | 0.513 | 0.628 | 0.661 | 0.260 | 0.398 |
| Baseline (LINE) | 0.320 | 0.359 | 0.369 | 0.518 | 0.593 | 0.645 | 0.219 | 0.373 |
| HARP (LINE) | 0.430 | 0.494 | 0.322 | 0.396 | 0.594 | 0.643 | 0.228 | 0.373 |
| NECL (LINE) | 0.317 | 0.355 | 0.355 | 0.517 | 0.595 | 0.649 | 0.215 | 0.369 |
| NECL-RF (LINE) | 0.444 | 0.513 | 0.353 | 0.493 | 0.619 | 0.661 | 0.252 | 0.377 |

3.3.2 Comparisons of All Methods

In this section, we evaluate the effectiveness of our NECL-RF method and compare the results with NECL, HARP, and baseline methods. From the analysis of NECL, we can see that $\lambda = 0.5$ similarity threshold value gives the best result in terms of efficiency and effectiveness. For this reason, we decided to use the compressed graph for $\lambda = 0.5$ threshold value and get the embedding for the compressed graph. We present the macro F_1 and micro F_1 scores achieved on all datasets in Table 3.1 and 3.2. We use 5% of the labeled vertices for Citeseer, Wiki, and DBLP, 40% for BlogCatalog as training data. To have a detailed comparison between our models, NECL and NECL-RF, HARP and the baseline methods, we vary the fraction of labeled data for classification, and present macro F_1 and micro F_1 scores in Figure 3.8-3.11.

In Table 3.3, we see that NECL or NECL-RF gives the highest macro F_1 and micro F_1 scores for datasets with all baseline methods except for LINE on Wiki. For DBLP, NECL or NECL-RF gives the highest accuracy for all the three baseline models. NECL-RF significantly improves the quality of the embedding for all datasets except Citeseer with Node2vec and Wiki with

LINE.

While HARP has higher accuracy than baseline methods, it does multiple levels of iteration of graph coarsening and representation learning, so it increases the time complexity. On the other hand, we do iteration only one level in NECL-RF. Embedding time for NECL-RF is the total of embedding time for the original graph and compressed graph. As we see in the previous section, the compressed graph is much smaller than an original graph, so the learning time for the compressed graph is significantly less compared to the baseline method. Hence, complexity does not increase significantly as in HARP. As a result, we get similar or better effectiveness than HARP with less time complexity.

Detailed comparisons between all methods using different portions of labeled vertices as training data are presented in Figure 3.8-3.11. In most cases, we see that in most of the cases, NECL and NECL-RF give the highest accuracy compared to other models or give better results than the baseline models. We observe that, for some datasets, refinement decreases the accuracy of NECL. The reason for this decrease might be that, for some classification tasks, learning a global structure with compressed data, which also includes a local structure in the super-nodes, would be enough. So, when we relearn and update the embedding of the compressed graph, it might add noise to the features. As a result, it deteriorates the accuracy of the classification task. Also, as we see from the figures, our method has a better improvement on DeepWalk. The reason is that while Node2vec and LINE may learn higher-order proximity, regular random walks in DeepWalk may not capture higher-order proximity, so it loses the global information. It also depends on the datasets.

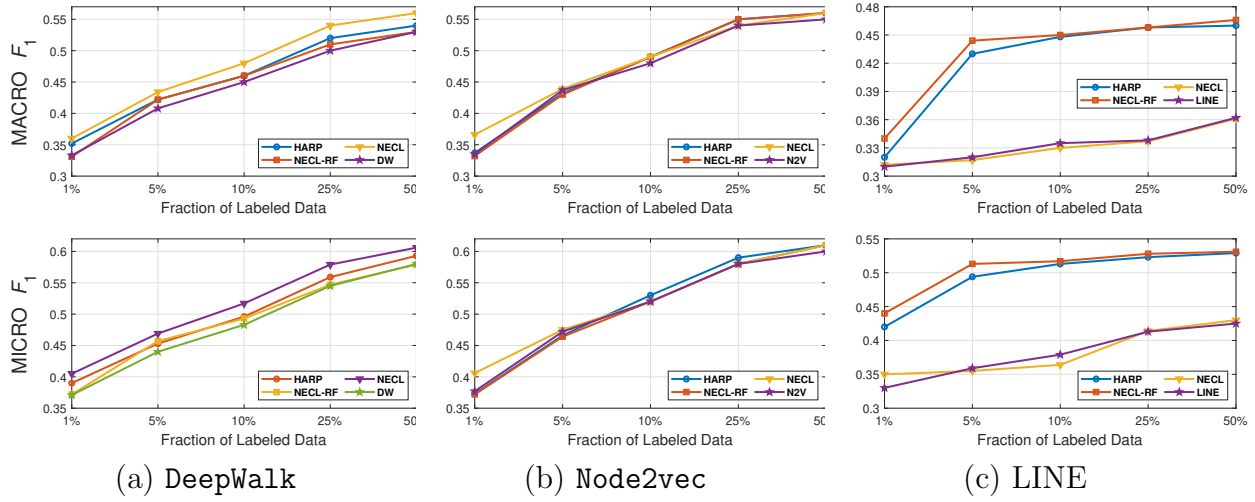


Figure 3.8 Detailed comparisons of classification results on Citeseer

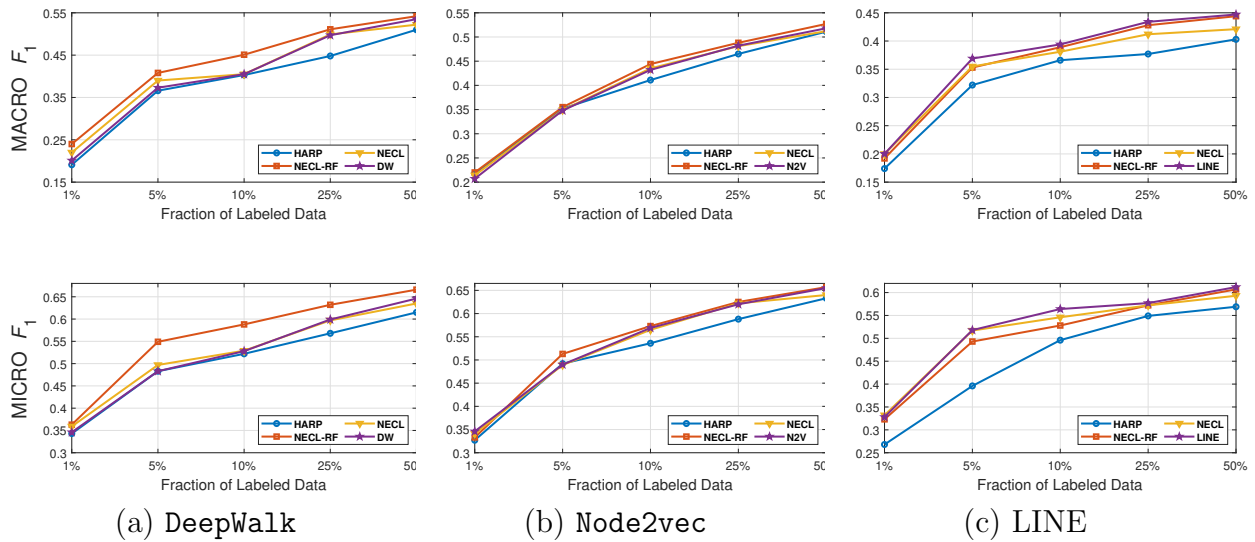


Figure 3.9 Detailed comparisons of classification results on Wiki

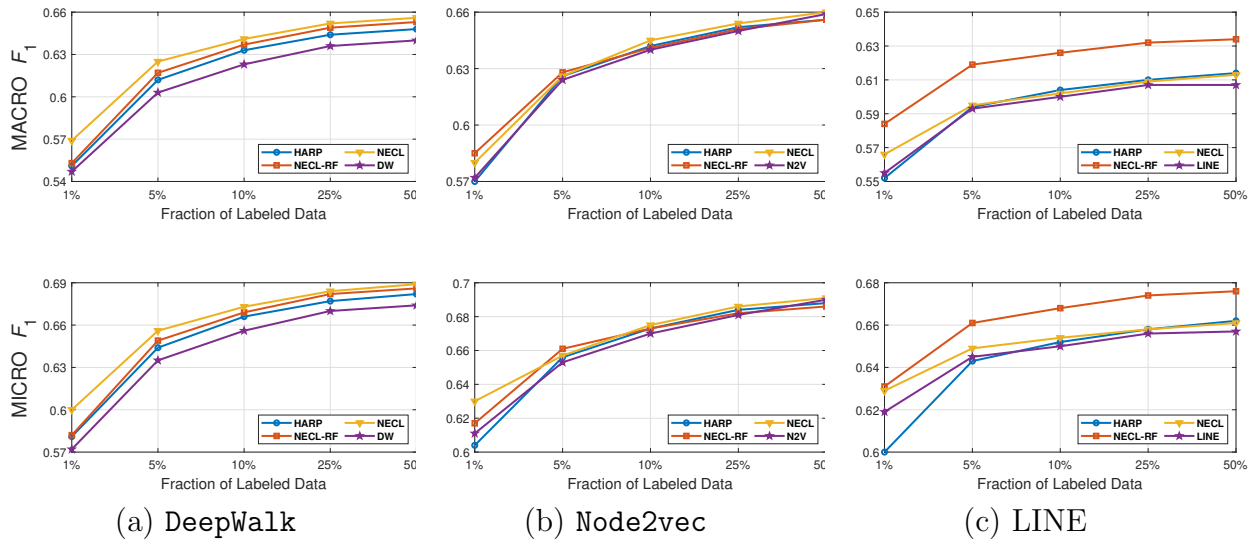


Figure 3.10 Detailed comparisons of classification results on DBLP

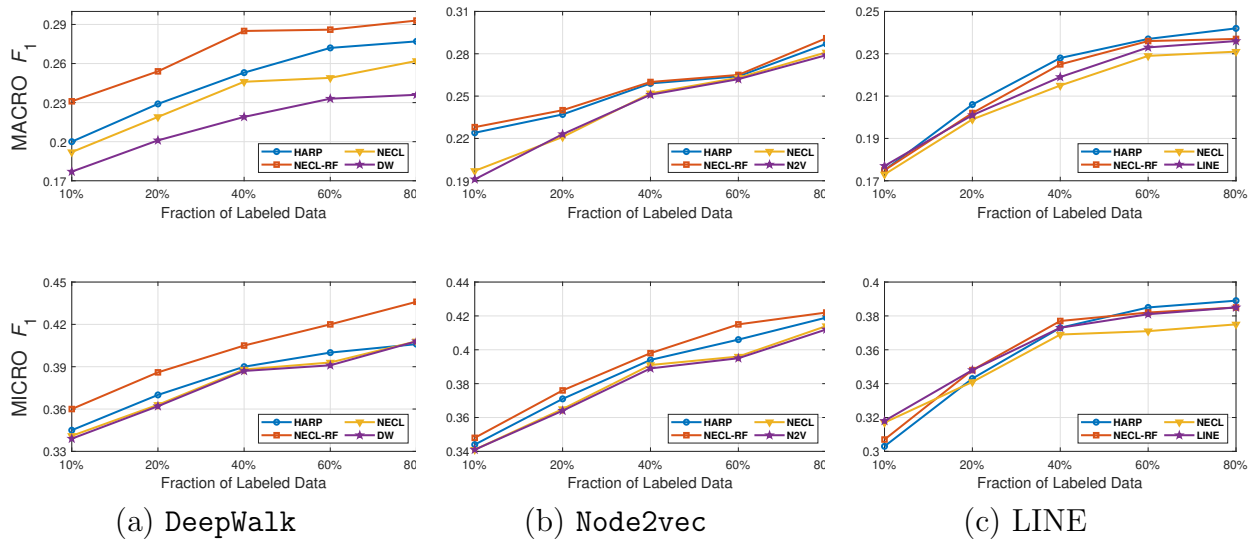


Figure 3.11 Detailed comparisons of classification results on BlogCatalog

CHAPTER 4

MPool: Motif-Based Graph Pooling

4.1 Introduction

Graph Neural Networks (GNNs) have recently become a powerful technique for many graph-related tasks, including graph classification. Current GNN models apply different graph pooling methods that reduce the number of nodes and edges to learn the higher-order structure of the graph in a hierarchical way. All these methods primarily rely on Graph Convolution Networks (GCNs) with layer-wise propagation based on the one-hop neighbors to calculate the assignment matrix in the clustering-based method and score in the node selection-based method. However, they do not consider the higher-order structure of the graph. In this work, we propose a graph pooling method, Motif Pool (MPool), that considers motifs as the higher-order graph structure for pooling operation. We proposed two types of graph pooling. As the first one, we develop node selection-based graph pooling with designing a node ranking model considering the motif adjacency of nodes. As the second one, we develop cluster-based graph pooling by designing a spectral clustering model using motif adjacency. Finally, the result of each channel is aggregated into the final graph representation. This work is published in PAKDD 2023 [86].

4.2 Methodology

In this section, we present our proposed graph pooling method. We first discuss our overall framework. Then, we explain our selection-based and clustering-based motif pooling meth-

ods. We further explain how we incorporate these methods into GNN models. Lastly, we present the complexity analysis of our models.

4.2.1 Framework

In this section, first, we discuss the problem formulation of graph classification and preliminaries. Then we present our motif-based pooling models.

4.2.2 Preliminaries and Problem Formulation

We denote a graph as $G(V, A, X)$ where V is the node-set, $A \in \mathbb{R}^{N \times N}$ is the adjacency matrix, and $X \in \mathbb{R}^{N \times d}$ is the feature matrix with d dimensional node feature and N is the number of nodes in the graph. We denote a graph collection as (\mathcal{G}, Y) where $\mathcal{G} = \{G_0, G_1, \dots, G_n\}$ with G_i 's are graphs and Y is the set of the graph labels. In this paper, we work on the graph classification problem, whose goal is to learn a function $f : \mathcal{G} \rightarrow Y$ to predict the graph labels with a graph neural network in an end-to-end way.

Graph Neural Network for Graph Classification: GNN for graph classification has two modules: message-passing and pooling. For message-passing operations, Graph convolution network (GCN) [7] is the most widely used model where it combines the features of each node from its neighbors as follows:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} \theta^{(l)}) \quad (4.1)$$

where $H^{(l+1)}$ is the node representation matrix for layer $(l+1)$, σ is an activation function, $\tilde{A} = A + I$ is the adjacency matrix with self-loop, $\tilde{D} \in \mathbb{R}^{N \times N}$ is the normalized degree matrix

of \tilde{A} , $\theta^{(l)}$ is trainable weight for $l^{(th)}$ layer and $H^{(l)}$ is the input node representation matrix for $l + 1^{th}$ layer obtained from previous layer. $H_0 = X$ is the initial input node feature matrix of the input graph. We utilize GCN for message-passing operations in our model.

The second module of GNNs for graph classification is the pooling operation that helps to learn the graph features. The main idea behind graph pooling is to coarsen the graph by reducing the number of nodes and edges to encode the information of the whole graph. In the literature, there are two types of hierarchical graph pooling methods: selection-based and clustering-based methods. Selection-based methods calculate a score (attention) using a scoring function for every node that represents their importance. Based on the calculated scores, the top k nodes are selected to construct a pooled graph. They use a classical graph adjacency matrix to propagate information and calculate the score.

Clustering-based pooling methods learn a cluster assignment matrix $S \in R^{N \times K}$ using graph structure and/or node features. Then, they reduce the number of nodes by grouping them into super nodes by $S \in R^{N \times K}$ to construct the pooled graph at $(l + 1)^{th}$ layer as follows

$$A^{(l+1)} = S^{(l)T} A^{(l)} S^{(l)}, \quad H^{(l+1)} = S^{(l)T} H^{(l)}. \quad (4.2)$$

Motifs and Motif-based Adjacency Matrix: Motifs (graphlets) are small, frequent, and connected subgraphs that are mainly used to measure the connectivity patterns of nodes [24]. Motifs of sizes 2-4 are shown in Figure 4.1. To include higher-order structural information between nodes, we create the motif adjacency matrix M_t for a motif t where $(M_t)_{i,j}$ represents the # of the motif containing nodes i and j .

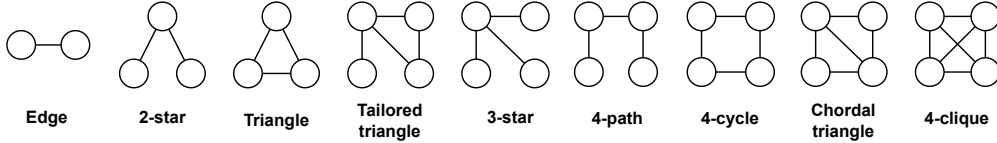


Figure 4.1 Motif Networks with size 2-4.

4.2.3 Motif Based Graph Pooling Models

We propose a hierarchical pooling method based on motif structure. As the first layer, graph convolution (GCN) takes the adjacency matrix A and feature matrix X of the graph as input and then updates the feature matrix by propagating the features through the neighbors and aggregating features coming from adjacent nodes. After getting the updated feature matrix from the convolution layer, our proposed graph pooling layer, MPool , operates coarsen on the graph. These steps are repeated l steps, and outputs of each pooling layer are aggregated with readout function [87] to obtain a fixed-sized graph representation. After concatenating the results of readouts, it is fed to the multi-layer perceptron (MLP) layer for the graph classification task. We develop three types of motif-based graph pooling methods: (1) MPool_S is the selection-based method, (2) MPool_C is the clustering-based method, and (3) MPool_{cmb} is the combined model. These are illustrated in Figure 4.2.

In this work, we adopt the model architectures from SAGPool [15] as the selection-based and MinCutPool [16] as the clustering-based model. On the other hand, our method is compatible with any graph neural network that we show later in our experiment section.

A. Selection-based Pooling via Motifs (MPool_S): Previous selection-based methods [14, 15] do the pooling operation using a classical adjacency matrix. However, higher-order

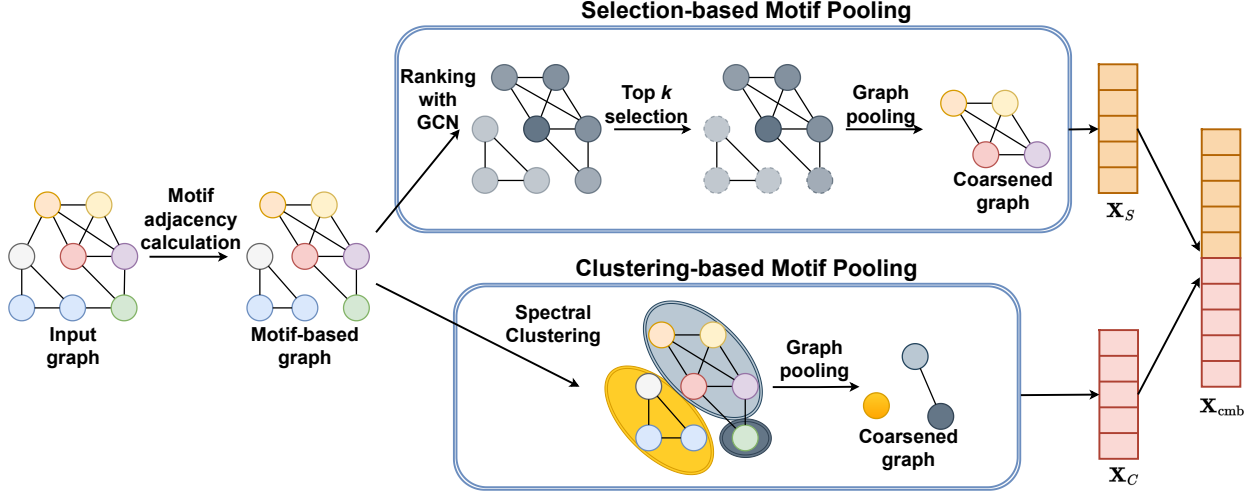


Figure 4.2 An illustration of our motif-based pooling methods.

structures like motifs show great performance on graph convolution network [17] and are also important structures for graph classification. Therefore, in our selection method, we first calculate the motif adjacency matrix for a particular motif type, e.g., triangle, from the original graph as we discuss in Section 5.2.1. Then, we calculate the motif attention score for each node by considering the motif adjacent. Based on these scores, we select the top k nodes for pooling and construct the coarsened graph using the pooling function. Figure 4.2 presents the overview of our selection-based graph pooling method. We use a graph convolution network to calculate the motif attention score for each node where we use node attributes and also motif-based graph topological information instead of pair-wise edge information. *Motif attention* score is defined as follows

$$Z = \sigma(D^{-\frac{1}{2}} \tilde{M} D'^{-\frac{1}{2}} X \theta_{att}) \quad (4.3)$$

where σ is an activation function, $\tilde{M} \in \mathbb{R}^{N \times N}$ is the motif adjacency matrix with self loop where $\tilde{M} = M + I_N$, $D' \in \mathbb{R}^{N \times N}$ is the degree matrix of M , and $\theta_{att} \in \mathbb{R}^{d \times 1}$ is the learnable

parameter matrix.

Based on the motif attention score, we select the top k nodes from the graph following the node selection method in [14]. The top $k = \alpha \times N$ nodes are selected based on the Z value where α is the pooling ratio between 0 and 1. Thus, we obtain the pooling graph as follows

$$idx = topK(Z, [\alpha \times N]) \tag{4.4}$$

$$X_{out} = X_{idx,:} \odot Z_{idx}, \quad A_{out} = A_{idx,idx}$$

where idx is the indices of the top k nodes from the input graph which is returned by $topK$ function, X_{idx} is the features of the selected k nodes, Z_{idx} is the motif attention value for those nodes. \odot is the element-wise broadcasted product, $:$ is the indexed of each node feature. $A_{idx,idx}$ is row and column wised indexed matrix, A_{out} is the adjacency matrix and X_{out} is the new feature matrix of the pooled graph.

Since we use graph features and motif adjacency matrix with convolution, the motif attention score is based on higher-order graph structures and features. So pooling operation gets the important nodes with respect to higher-order structure.

B. Clustering-based Pooling via Motifs (MPool_C): In this paper, as the base for our clustering-based pooling methods, we use MinCutPool [16] that is defined based on Spectral clustering (SC) by minimizing the overall intra-cluster edge weights. MinCUTpool proposes to use GNN with a custom loss function to compute cluster assignment with relaxing the normalized minCut problem. However, they consider only the regular edge-based adjacency matrix to find clusters. On the other hand, considering edge-type relations between nodes

may result in ignoring the higher-order relations. Including higher-order relations like motifs for clustering may produce better groups for pooling.

In our clustering-based method, we calculate the cluster assignment matrix S utilizing motif adjacency information. We adopt spectral clustering method [16] where we use multi-layer perceptron (MLP) by inputting node feature matrix X . We use the softmax function on the output layer of MLP. This function maps each node feature X_i into the i^{th} row of a soft cluster assignment matrix S

$$S = MLP(X; \theta_{MLP}) = softmax(ReLU(XW_1)W_2) \quad (4.5)$$

However, as it is seen in Equation 4.5, we do not use adjacency but use attributes of nodes obtained from the convolution part. Therefore, to include motif information in the pooling layer, we use the motif adjacency matrix in the convolution layer while passing the message to neighbors as $X = GNN(X, \tilde{M}; \theta_{GNN})$ where \tilde{M} is the normalized motif adjacency matrix, and θ_{GNN} and θ_{MLP} are learnable parameter.

We also incorporate motif information in the optimization. Parameters of the convolution layer and pooling layer are optimized by minimizing a loss function \mathcal{L} including the usual supervised loss function \mathcal{L}_s and also an unsupervised loss function[16] \mathcal{L}_u as $\mathcal{L}_u = \mathcal{L}_c + \mathcal{L}_o$ where

$$\mathcal{L}_c = -\frac{Tr(S^T MS)}{Tr(S^T DS)} \quad \text{and} \quad \mathcal{L}_o = \left\| \frac{S^T S}{\|S^T S\|_F} - \frac{I_K}{\sqrt{K}} \right\|_F \quad (4.6)$$

\mathcal{L}_c is the cut loss that encourages strongly connected nodes in motif adjacency to be clustered together. Here, $Tr(S^T MS) = \frac{1}{k} \sum_{k=1}^K S_k^T M S_k$ and $Tr(S^T DS) = \frac{1}{k} \sum_{k=1}^K S_k^T D S_k$ where K

is the number of clusters and D is the degree matrix and M is the motif adjacency matrix. L_o is the orthogonality loss, which helps the clusters to become similar in size. I_K is a (rescaled) clustering matrix $I_K = \widehat{S}^T \widehat{S}$, where \widehat{S} assigns exactly N/K points to each cluster. After calculating the cluster assignment matrix, we compute the coarsened graph adjacency matrix and attribute matrix using Equation 6.4.

C. Combined model (MPool_{comb}): Selection-based models mainly focus on preserving the local structure of the node by selecting top- K representative nodes while cluster-based methods basically focus on the global structure of the graph by assigning nodes into K -clusters. To utilize the benefits of the selection-based and cluster-based models at the same time, we combine our selection-based and cluster-based motif pooling models into one model. As a result graph representation from the combined model encoded local structure information from the selection-based model and the global structure model from the cluster-based model. In this model we concatenate the graph-level representation from the selection-based motif pooling method and cluster-based motif pooling method into one final representation as follows:

$$X_{cmb} = X_S \oplus X_C \tag{4.7}$$

where X_S is the graph-level representation from MPool_S model and X_C from MPool_C method and, \oplus is the concatenation operation.

4.2.4 Readout Function and Output Layer

To get a fixed-sized representation from different layers' pooled graph, we apply a readout function[15] that aggregates the node features as follows: $Z = \frac{1}{N} \sum_{i=1}^N x_i || \max_{i=1}^N x_i$ where N is the number of nodes, x_i is the i^{th} node feature and $||$ denotes concatenation. After concatenating the results of all readout functions as a representation of the graph, it is given as an input to a multilayer perceptron with the softmax function to get the predicted label of the graph as $\hat{Y} = softmax(MLP(Z))$ where Z is the graph representation. For graph classification, parameters of GNNs and pooling layers are optimized by a supervised loss as $\mathcal{L}_s = - \sum_{i=1}^L \sum_{j=1}^C Y_{i,j} \log \hat{Y}_{i,j}$ where Y is the actual label of the graph.

4.2.5 Graph Reconstruction

In addition to using pooling for the graph classification problem, we further use it in autoencoder (AE) to construct graphs from their representations. We design an AE by optimizing two objectives: (1) node attribute reconstruction and (2) edge reconstruction. For the node attribute reconstruction, we use the mean squared error (MSE) as the optimization function. For the edge reconstruction, the predicted adjacency matrix is scored against the target adjacency matrix using binary cross-entropy (BCE) loss. In our model, we change the last step as one MLP, which is optimized with graph classification task, with two MLPs, one predicts the adjacency matrix with a final sigmoid activation and the other one predicts the node attributes. Both the edge and attribute MLPs use the ReLU activation for the first two

layers and are defined as follows

$$\hat{\mathbf{A}} = MLP_{\text{edge}}(V; \theta_{MLP_{\text{edge}}}) \quad \text{and} \quad \hat{\mathbf{X}} = MLP_{\text{attribute}}(V; \theta_{MLP_{\text{attribute}}}).$$

4.2.6 Complexity Analysis

The computational complexity of our methods involves two parts that correspond to motif adjacency matrix calculation and pooling layer. Since we use the method from [88] to calculate the motif adjacency matrix, the computation complexity of motif matrix calculation for dense matrices is $O(|V|^3)$ and for sparse matrices is $O(|V|^2)$ where $|V|$ represent the number of vertices. In our selection-based method, we next do the graph pooling operation similar to SAGPool [15] which has time complexity $O(|V|^2)$ for dense matrices and $O(|E|)$ for sparse matrices where $|E|$ is the number of edges. Since our implementation is sparse for our selection method, the computational complexity is $O(|V|^2 + |E|)$. For the clustering-based method, we adopt MinCutPool [16] that requires $O(K(E + VK))$ computational cost where K is the number of clusters in the cluster assignment matrix. It is further implemented by a dense matrix. Hence, the total computational cost for our node cluster-based method is $O(V^2 + K(E + VK))$.

4.3 Experiment

We evaluate the performance of our models in graph classification problems and compare our results with the baseline methods for selection-based and clustering-based on different datasets. We also give the results for the variation of our model by utilizing different

Table 4.1 Graph statistics. $|G|$, V_{avg} , E_{avg} , and $|C|$ denote the number of graphs, the average number of nodes and edges, the number of classes in each dataset, respectively.

| Datasets | | $ G $ | V_{avg} | E_{avg} | $ C $ |
|---------------------|---------------|-------|-----------|-----------|-------|
| Biochemical network | D&D | 1178 | 284.32 | 715.66 | 2 |
| | NCI1 | 4110 | 29.87 | 32.30 | 2 |
| | NCI109 | 4127 | 29.68 | 32.13 | 2 |
| | PROTEINS | 1113 | 39.06 | 72.82 | 2 |
| | Mutagenicity | 4337 | 30.32 | 30.77 | 2 |
| Social network | IMDB-BINARY | 1000 | 19.77 | 96.53 | 2 |
| | REDDIT-BINARY | 2000 | 429.63 | 497.75 | 2 |
| | COLLAB | 5000 | 74.49 | 2457.78 | 3 |

message-passing models. Further, we analyze the effect of the motif types on the results of the pooling. More experiments can be found on supplements.

Datasets: We use eight benchmark graph datasets 6.1 in our experiments commonly used for graph classification [89]. Among these, three datasets are social networks (SN); IMDB-BINARY, REDDIT-BINARY, and COLLAB, and five other datasets are biological and chemical networks (BN) ;D&D, PROTEINS (PROT), NCI1, NCI109, and Mutagenicity (MUTAG) .

Baseline: We use five graph pooling methods as baseline methods. Among them, gPool [14] and SAGPool [15] are selection-based method and MinCutPool (MCPool) [16], DiffPool [62] and ASAP [63] are clustering-based method. We also combined SAGPool and MCPool models in a single model and use as a baseline model.

Experimental setup: To evaluate our models for the graph classification task, we randomly split the data for each dataset into three parts. We use 80% data for the training set, 10% data for the validation set, and 10% data for the test set. We do the splitting

process 10 times using 10 random seed values. We implement our model using PyTorch and PyTorch Geometric library. For optimizing the model, we use Adam optimizer [90]. In our experiments, we take node representation size as 128 for all datasets. Our hyperparameters are as follows: learning rate in $\{1e-2, 5e-2, 1e-3, 5e-3, 1e-4, 5e-4\}$, weight decay in $\{1e-2, 1e-3, 1e-4, 1e-5\}$, and pooling ratio in $\{1/2, 1/4\}$. We find the optimal hyperparameters using grid search. We run the model for a maximum of 100K epochs, and there is an early stopping condition if the validation loss does not improve for 50 epochs. Our model architecture consists of three blocks, and each block contains one graph convolution layer and one graph pooling layer like [15]. We use the same model architecture and hyperparameters with MinCuT and SAGPool models.

4.3.1 Overall Evaluation

Performance on Graph Classification: In this part, we evaluate our proposed graph pooling methods for the graph classification task on the given eight datasets. Each dataset contains a certain number of input graphs and their corresponding label. In the graph classification task, we classify the input graph by predicting the label of the graph. We use node features of the graph as the initial features of the model. If a dataset does not contain any node feature, we use node degrees as initial features using one-hot encoding. Table 4.2 and Table 5.2 show the average graph classification accuracy, standard deviation, and ranking of our models and other baseline models for all datasets. We can observe from the tables that our motif-based pooling methods consistently outperform other state-of-art models, and our models get the first rank for almost all datasets.

Table 4.2 Comparison of our models with baseline pooling methods for biochemical datasets.

| Model | D&D | NCI1 | NCI109 | PROT | MUTAG | Rank |
|-----------------------------|--------------------|--------------------|-------------------|--------------------|--------------------|----------|
| gPool | 75.0±0.9/7 | 67.0±2.3/8 | 66.1 ±1.6/8 | 71.1 ±0.9/8 | 71.9 ±3.7/8 | 7.8 |
| SAGPool | 75.7±3.7/6 | 68.7±3.0/7 | 71.0±3.4/5 | 72.5 ±4.0/7 | 74.9±3.9/7 | 6.4 |
| MCPool | 76.7±3.0/5 | 73.1 ±1.4/4 | 71.5 ±2.7/4 | 76.3 ±3.6/3 | 75.9 ±2.7/6 | 4.4 |
| DiffPool | 66.9 ±2.4/9 | 62.2 ±1.9/9 | 62.0±2.0/9 | 68.2 ±2.0/9 | 77.6 ±2.6/3 | 7.8 |
| ASAP | 76.9 ±0.7/4 | 71.5±0.4/5 | 70.1 ±0.6/7 | 74.2 ±0.8/5 | - | 4.2 |
| Combined | 74.5±9.8/8 | 74.1 ±1.2/3 | 72.0 ±2.1/3 | 75.6±2.1/4 | 76.5±3.2/4 | 4.4 |
| MPool_{comb} | 81.2 ±2.1/1 | 77.4± 1.9/1 | 73.5±2.5/1 | 79.3 ±3.3/1 | 79.6 ±3.7/1 | 1 |
| MPool _S | 77.2 ±4.6/3 | 71.0±3.4/6 | 70.8±2.1/6 | 72.7 ±4.2/6 | 76.4 ±3.1/5 | 5.2 |
| MPool _C | 78.5 ±3.3/2 | 74.4±1.8/2 | 73.1±2.5/2 | 78.1 ±3.3/2 | 78.8 ±2.1/2 | 2 |

Table 4.2 shows the results for our motif-based models and other graph pooling models on biochemical datasets. We obtain the reported results for gPool and DiffPool from the SAGPool paper since our model architecture and hyperparameters are the same as SAGPool. Also, for the ASAP method, we obtain the results from the initial publication (“-”) means that results are not available for that dataset. As we see from the table, MPool_{comb} gives the highest result for all biochemical networks. In particular, MPool_{comb} achieves an average accuracy of 81.2% on D&D and 77.4% on NCI1 datasets which are around 4% improvements over the MPool_C method as the second-best model. We can also see MPool_{comb} gives very good accuracy compared to baseline models for all biochemical datasets. Especially for D&D, NCI1, and NCI109 datasets MPool_{comb} gives 5.8%, 5.8%, and 3.9% improvements over the best model of baseline models for these datasets. From this result, we can say that incorporating global and local structures of the graph in the combined model gives better results for graph classification on biochemical data. We further calculate the average rank for all models, where our model MPool_{comb} average rank is the lowest at 1 and our model MPool_C is the second lowest.

Table 4.3 Comparison of our models with baseline pooling methods for social network datasets.

| Model | IMDB-B | REDDIT-B | COLLAB | Avg. Rank |
|----------------------------|-----------------------|----------------------|----------------------|-----------|
| gPool | 73.40±3.7 (3) | 74.70±4.5 (7) | 77.58 ±1.6 (3) | 4.3 |
| SAGPool | 73.00±4.06 (4) | 84.66±5.4 (2) | 70.10±2.5 (7) | 4.3 |
| MinCutPool | 70.78±4.7 (8) | 75.67 ±2.7 (6) | 69.91 ±2.3 (8) | 7.3 |
| DiffPool | 68.40 ±6.1 (9) | 66.65 ±7.7(8) | 74.83 ±2.0 (4) | 7 |
| ASAP | 72.74 ±0.9 (5) | - | 78.95 ±0.7 (2) | 3.5 |
| Combined | 71.20±4.50(7) | 88.40±0.22(1) | 71.85±3.73(6) | 4.7 |
| MPool_{cmb} | 74.20 ±2.8 (1) | 84.10± 5.0 (3) | 74.13±2.3 (5) | 2.6 |
| MPool_S | 73.44 ±3.9 (2) | 83.89±4.3 (4) | 68.95±2.7 (9) | 5 |
| MPool_C | 71.44 ±4.0 (6) | 78.77±5.0 (5) | 83.62±5.2 (1) | 4 |

Table 5.2 shows the performance comparison with our models and other baseline models on social network datasets. As we see from the table, our proposed methods outperform all the baseline methods for all datasets except ReDDIT-BINARY, where our model is the third best with giving very close to the second one, SAGPool. For IMDB-BINARY and REDDIT-BINARY MPool_{cmb} model gives better accuracy than the MPool_S and MPool_C model while for COLLAB dataset MPool_C give much higher accuracy than our other two models. For both types of datasets, our selection-based method MPool_S gives better accuracy than the selection-based baseline methods SAGPool and gPool for most of the datasets. In particular, MPool_S achieves an average accuracy of 77.21% on D&D and 76.42% on MUTAG datasets which is around 2% improvement over the SAGPool method which is our base model. Similarly, our cluster-based model outperforms the baseline methods of cluster-based methods for most of the datasets. Especially, MPool_C achieves an average accuracy of 83.62% on COLLAB datasets, which is around 5% improvement over the ASAP method as the second-best model and around 14% improvement over the MinCutPool, which is our base model.

Furthermore, when we compare our selection-based model MPool_S and clustering-based model MPool_C results from Tables, we can see that MPool_C outperforms MPool_S for all biochemical datasets. While MPool_S gives better accuracy for two social networks, IMDB-BINARY and REDDIT-BINARY, MPool_C have 15% better accuracy than MPool_S on the COLLAB dataset.

Ablation Study: While we use GCN as the base model for message passing, our pooling model can integrate other GNN architectures. In order to see the effects of different GNN models in our methods, we utilize the other four most widely used convolutional graph models: Graph convolution network (GCN) [7], Graph-SAGE [8], GAT [9], and GraphConv [91]. Table 4.4 shows average accuracy results for these GNN models using MPool_S , MPool_C and MPool_{cmb} on NCI1 and IMDB-BINARY datasets. As there is no dense version of Graph attention network(GAT), we use it only for selection-based model MPool_S . For this experiment, we use triangle motifs for the motif adjacency matrix calculation. As we see in the table, the effects of GNN models and which model gives the best result depend on the dataset. For the NCI1 dataset, Graph-SAGE gives the highest accuracy on MPool_S and MPool_{cmb} model while GraphConv gives the highest accuracy on MPool_C model. For IMDB-BINARY, all the graph convolutional models give very close results for all of our pooling models. For MPool_C and MPool_{cmb} Graph-SAGE gives better accuracy than the other GNN models while GAT gives the highest accuracy for MPool_S model.

We further study the effect of the motif type for pooling. In this experiment, we use 2-star, triangle, and a combination of 2-star and triangle motifs, as these motifs are observed the

Table 4.4 MPool_S MPool_C and MPool_{cmb} performance with different GNN models.

| GNN Model | MPool _S | | MPool _C | | MPool _{cmb} | |
|----------------------------------|--------------------|--------------|--------------------|--------------|----------------------|--------------|
| | NCI1 | IMDB-B | NCI1 | IMDB-B | NCI1 | IMDB-B |
| <i>MPool_{GCN}</i> | 70.98 | 73.44 | 74.44 | 71.44 | 76.09 | 73.90 |
| <i>MPool_{GraphConv}</i> | 74.20 | 73.50 | 75.93 | 71.90 | 74.7 | 73.00 |
| <i>MPool_{SAGE}</i> | 74.69 | 73.00 | 74.13 | 72.22 | 78.80 | 74.00 |
| <i>MPool_{GAT}</i> | 67.15 | 74.00 | - | - | - | - |

Table 4.5 MPool_S MPool_C and MPool_{cmb} performance with different motifs.

| Model | Motif | DD | NCI1 | Mutag | IMDB-B |
|----------------------|-----------------|--------------|--------------|--------------|--------------|
| MPool _S | 2-star | 77.21 | 69.48 | 70.11 | 73.00 |
| | Triangle | 75.63 | 70.98 | 76.42 | 73.44 |
| | 2-star+triangle | 75.63 | 69.82 | 72.39 | 69.64 |
| MPool _C | 2-star | 78.48 | 73.56 | 73.56 | 71.20 |
| | Triangle | 75.80 | 74.44 | 78.77 | 71.44 |
| | 2-star+triangle | 74.21 | 74.20 | 76.00 | 70.96 |
| MPool _{cmb} | 2-star | 81.20 | 77.36 | 79.60 | 74.20 |
| | Triangle | 80.50 | 76.09 | 77.90 | 73.90 |
| | 2-star+triangle | 79.95 | 76.75 | 78.42 | 73.40 |

most in real-world networks. We present the graph classification accuracy for different motifs using MPool_S MPool_C and MPool_{cmb} in Table 4.5. As we see in the table, we get the highest accuracy for MPool_S and MPool_C with the triangle motif for three datasets NCI1, MUTAG, and IMDB-BINARY. For D&D, we get the highest accuracy with 2-star motif adjacency on MPool_S and MPool_C. We also observe that for D&D, the accuracy of the selection-based model does not vary much compared to the clustering-based model. For MUTAG, different motifs have a large effect on the accuracy, where triangle motif adjacency gives around 4% and 3% higher accuracy than the 2-star motif adjacency for the selection-based method and for the clustering-based model, respectively. For IMDB-BINARY, 2-star and triangle motifs give similar accuracy for both methods, and 2-star+triangle motif adjacency gives less accuracy for the clustering-based method. For our combined model MPool_{cmb} the 2-star motif gives the highest accuracy for all datasets whereas other motifs give very close results to the 2-star motif.

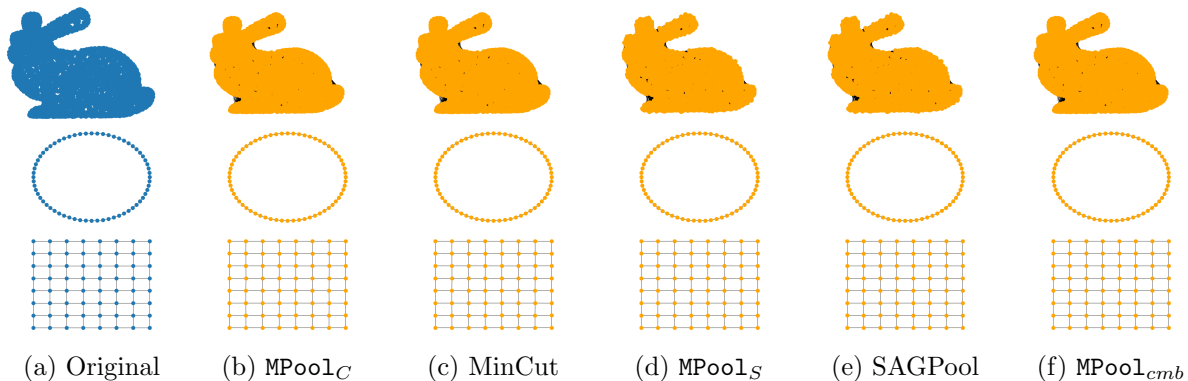


Figure 4.3 Graph reconstruction Results after Pooling.

Performance on Graph Reconstruction:

In addition to using pooling for the graph classification problem, we further use it in autoencoder (AE) to construct graphs from their representations. We build a model for graph reconstruction using an AE. Our goal is to preserve the content information that is coming from node attributes and also structural information that is coming from the adjacency matrix. We design an AE by optimizing two objectives: (1) node attribute reconstruction and (2) edge reconstruction. For the node attribute reconstruction, we use the mean squared error (MSE) as the optimization function. For the edge reconstruction, the predicted adjacency matrix is scored against the target adjacency matrix using binary cross-entropy (BCE) loss. We show our graph reconstruction results for our pooling methods in Figure 4.3. The blue graph is the original, and the orange graph is the reconstructed one. Since the predicted adjacency is probabilistic, if the value is greater than 50%, we put an edge. All methods have 100% accuracy for edges on the ring and grid graph and an edge accuracy of 97.96% on the bunny graph. From these results, we can easily see that our models are able to preserve the content information and graph structure. We also present loss as MSE for node attributes

Table 4.6 Autoencoder position loss.

| Graph | $MPool_S$ | $MPool_C$ | $MPool_{cmb}$ | SAGPool | MinCutPool |
|-------|-----------|-----------|-----------------|----------|------------|
| Bunny | 1.64e-07 | 7.49e-10 | 2.80e-10 | 1.63e-07 | 7.49e-10 |
| Ring | 4.12e-09 | 9.80e-08 | 1.62e-09 | 2.44e-09 | 9.79e-08 |
| Grid | 3.24e-09 | 1.11e-07 | 1.63e-09 | 4.75e-09 | 1.11e-07 |

as position in

CHAPTER 5

Graph Contrastive Learning via Structure-Aware Graph Compression

5.1 Introduction

Graph compression is a powerful technique for reducing graph complexity while preserving essential structural information. Existing graph contrastive learning models often rely on random node and edge dropping for augmentation, which can result in significant information loss. To address this, we propose Graph Contrastive Learning via Structure-Aware Graph Compression, a novel framework for graph classification that constructs compressed graph views to retain both local and global structures.

- Instead of randomly altering graphs, we introduce a structure-aware augmentation strategy using K-core and K-truss decomposition. We propose a contrastive learning framework that leverages K-core and K-truss view encoders to preserve both low-order and high-order structural information.
- Our model optimizes representations from both encoders via contrastive learning, maximizing feature consistency in an unsupervised manner. This enhances structural feature learning and improves classification performance.
- We conduct extensive experiments on six benchmark graph datasets, comparing our model against state-of-the-art GNN-based methods. Results demonstrate that our approach achieves superior performance in graph classification.

5.2 Methodology

In this section, we first present some preliminaries and problem formulation of this paper. Then we discuss some primary concepts of Graph Neural Network and contrastive learning. Then we present our proposed model, graph contrastive learning with graph compression.

5.2.1 Preliminaries and Problem Formulation

We denote a graph as $G(V, A, X)$ where V is the node-set, $A \in \mathbb{R}^{N \times N}$ is the adjacency matrix, and $X \in \mathbb{R}^{N \times d}$ is the feature matrix with d dimensional node feature and N is the number of nodes in the graph. We denote a graph collection as (\mathcal{G}, Y) where $\mathcal{G} = \{G_0, G_1, \dots, G_n\}$ with G_i 's are graphs and Y is the set of the graph labels. In this paper, we work on the graph classification problem, whose goal is to learn a function $f : \mathcal{G} \rightarrow Y$ to predict the graph labels with a graph neural network in an end-to-end way.

5.2.2 Graph Neural Network

Recently, GNNs have been used as benchmark models for static graphs for different types of downstream tasks, including node classification, link prediction, and graph classification. Graph Convolution Network (GCN) [7] is the most widely used GNN model. GCN is a multilayer neural network to processes the graph data where it combines the features of each node from its neighbors while propagating the information through the edges. Given an input graph as $G(V, A, X)$ where V is the node-set, $A \in \mathbb{R}^{N \times N}$ is the adjacency matrix and $X \in \mathbb{R}^{N \times d}$ is the feature matrix with d dimensional node feature and N is the number of

nodes in the graph GCN layer transform the node representation as follows:

$$H^{(1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(0)} \theta^{(0)}) \quad (5.1)$$

where σ is a non linear activation function, $\tilde{A} = A + I$ is the adjacency matrix with self-loop, $\tilde{D} \in \mathbb{R}^{N \times N}$ is the normalized degree matrix of \tilde{A} , $\theta^{(0)}$ is trainable weight and $H^{(0)} = X$ is initial node features. GCN model uses multiple convolutional layers to learn the spatial feature for nodes from the connected nodes as follows:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} \theta^{(l)}) \quad (5.2)$$

where $H^{(l)}$ feature matrix and $\theta^{(l)}$ model parameter for $l^{(th)}$ layer. The underlying intuition for each layer is that nodes gather and aggregate information from their local neighbors. With the l layers GCN model, nodes can get information from l -hops neighborhood information.

In general, while training the model for the classification problem, cross-entropy loss is used as the supervised loss function, which compares the actual label with the predicted label.

5.2.3 Contrastive Learning

Contrastive learning (CL) has become one of the most popular approaches for unsupervised representation learning, which learns through comparisons among different samples. This comparison is typically conducted between positive pairs of "similar" inputs and negative pairs of "dissimilar" inputs. Contrastive learning in the graph domain aims to learn node or graph representation for a given input graph by maximizing the similarity between the

different views of the input graph in their latent space via contrastive loss. In general, there are three parts to graph contrastive learning: i) View construction, ii) View encoder, and iii) Contrastive loss.

- **View Construction:** A view is represented as graph data, denoted as $vi = (A_i, X_i)$, where $A_i \in \mathbb{R}^{N \times N}$ and $H(i) \in \mathbb{R}^{N \times d}$. While it is common to get 2 views, there are also some models that create more than 2 views [92, 93]. In practical terms, view augmentation approaches involve techniques like node dropping [94], edge perturbation [95], attribution masking [96], and subgraph sampling [97].
- **View Encoder:** View encoder maps the node representation or graph representation from different views of input graphs. As encoders for learning representations from views, any GNN models can be used based on the problem and type of graphs.
- **Contrastive loss:** After learning the representations for different views using the view encoder, the contrastive learning model is optimized by a contrastive loss. The objective of the contrastive loss function is to maximize the similarity among different view representations of the same graph and minimize the similarity for view representations of other graphs. In general, the contrastive loss can be described as follows:

$$L_{contra} = -\log\left(\frac{sim(Z_{vi}, Z_{vj})}{\sum_{j'=1}^N sim(Z_{vi}, Z_{vj'})}\right) \quad (5.3)$$

where the $sim(\cdot)$ function measures the similarity between two representations. We can use any similarity function, like cosine similarity and mutual information (MI) in

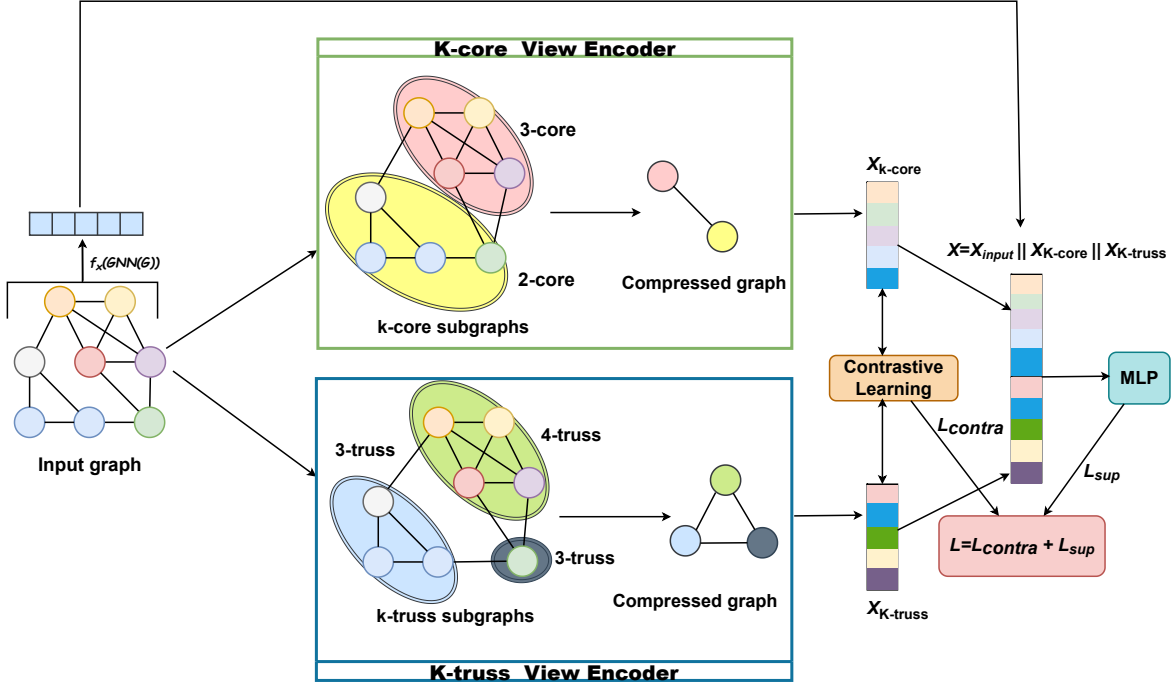


Figure 5.1 An illustration of our Graph Contrastive Learning with Graph Compression.

the contrastive loss. Z_{vi} and Z_{vj} are the i th view and j th view representations of node v .

5.2.4 Graph Contrastive Learning with Graph Compression

Our proposed Graph Contrastive Learning with Graph Compression model aims to enhance graph representations, particularly for graph classification tasks. The overall architecture is illustrated in Figure 5.1 and comprises three main components: (i) **K-core View Encoder**, (ii) **K-truss View Encoder**, and (iii) **Contrastive Learning**. Specifically, the K-core view encoder generates graph representations through K-core graph compression, while the K-truss view encoder employs K-truss-based graph compression for graph representa-

tion learning. Finally, the contrastive learning component optimizes these representations obtained from both encoders.

5.2.4.1 K-core View Encoder

In the K-core view encoder, we first compute K-core subgraphs corresponding to different k values from the input graph. Subsequently, these subgraphs are leveraged to construct compressed graph representations. We then utilize a Graph Convolutional Network (GCN) for message passing on both the subgraphs and the compressed graphs, followed by a pooling operation to obtain the final graph representations.

K-core Subgraph Decomposition: A K-core subgraph is formally defined as follows:

Definition 3. *K-core: Given a graph $G = (V, E)$ and an integer $k \geq 1$, the K-core of graph G is the maximal induced subgraph $G_k = (V_k, E_k)$ in which every vertex $v \in V_k$ has a degree of at least k , formally denoted as:*

$$\forall v \in V_k, \quad \text{deg}(v) \geq k. \quad (5.4)$$

We perform K-core decomposition for multiple values of k to obtain various subgraphs. To ensure the uniqueness of nodes across subgraphs, nodes appearing in multiple K-core subgraphs are retained only in the subgraph corresponding to the highest k -value and removed from lower-order subgraphs. Subsequently, we evaluate the size of each resulting subgraph. If a subgraph contains more nodes than a predefined threshold λ , we partition it evenly into smaller subgraphs using the Breadth-First Search (BFS) algorithm. This process is repeated iteratively until each subgraph’s size is at most λ .

Graph Compression: After extracting the subgraphs, we construct a compressed graph containing fewer nodes and edges, effectively preserving the structural integrity of the original graph. In this compressed representation, each subgraph is encapsulated as a super-node, significantly reducing the complexity while maintaining critical structural attributes. The connections between super-nodes are established based on inter-subgraph connectivity, measured explicitly by edge density:

$$A_{\text{super}}(i, j) = \frac{|E_{ij}|}{\min(|E_i|, |E_j|)} \quad (5.5)$$

where E_{ij} denotes the set of edges connecting subgraphs i and j , and $|E_i|, |E_j|$ represent the total number of edges within each subgraph, respectively. This compression approach effectively retains crucial topological relationships, substantially enhancing computational efficiency, scalability, and the effectiveness of subsequent graph representation learning tasks.

Graph Encoding. We first apply a Graph Neural Network (GNN), specifically a Graph Convolutional Network (GCN), as a message-passing model to learn node-level representations for each decomposed subgraph and the corresponding compressed graph. The node representation learning through GCN is formally expressed as:

$$H^{(1)} = \sigma \left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(0)} \theta^{(0)} \right) \quad (5.6)$$

where σ denotes a non-linear activation function, $\tilde{A} = A + I$ is the adjacency matrix augmented with self-loops, $\tilde{D} \in \mathbb{R}^{N \times N}$ is the degree matrix of \tilde{A} , and $\theta^{(0)}$ represents the trainable parameters.

After obtaining the node embeddings, we utilize a readout layer to aggregate these node-

level representations into fixed-size graph-level embeddings for each snapshot graph. The readout function integrates node features by concatenating the mean and maximum operations, defined formally as:

$$Z_{K-core} = \frac{1}{N} \sum_{i=1}^N [h_i \parallel \max_{i=1}^N h_i] \quad (5.7)$$

where N denotes the total number of nodes, h_i is the embedding of the i^{th} node, and \parallel denotes the concatenation operator. This aggregation effectively consolidates node-level features, thus enhancing the discriminative power of the learned graph representations for downstream classification tasks.

5.2.4.2 *K-truss View Encoder*

The K-truss view encoder follows a structure similar to the K-core encoder but leverages triangle-based edge connectivity for graph decomposition and compression. Specifically, we begin by computing K-truss subgraphs over a range of k values from the input graph. These subgraphs are then used to construct a compressed graph representation. A Graph Convolutional Network (GCN) is applied to both the subgraphs and the compressed graph for message passing, followed by a pooling operation to obtain final graph-level representations.

Definition 4. *Support:* In a graph $G = (V, E)$, the support of an edge $e = (u, v) \in E$, denoted as $sup_G(e)$, refers to the number of triangles in which the edge participates. Formally, $sup_G(e) = |\{ \Delta_{uvw} : w \in V \}|$.

Definition 5. *k-truss Subgraph:* A subgraph $S = (V_S, E_S)$ where $S \subseteq G$, $V_S \subseteq V$, and $E_S \subseteq E$, is a k -truss subgraph if every edge $e \in E_S$ has support of at least $k - 2$, where

$k \geq 2$.

The k -truss structure is inherently based on triangle participation. Every graph is trivially a 2-truss subgraph. The nested hierarchy of k -truss subgraphs follows: $G_3 \subseteq G_2$, $G_4 \subseteq G_3$, and so on, such that $G_k \subseteq G_{k-1}$.

Definition 6. Edge Trussness: For a graph G and a given $k > 2$, an edge $e = (u, v)$ may appear in multiple k -truss subgraphs. The trussness of edge e , denoted as $T_r(u, v)$, is defined as the largest k such that $e \in G_k$ and $e \notin G_{k+1}$.

We extract k -truss subgraphs over a selected range of k values. If a subgraph exceeds a predefined size threshold λ , we partition it using a Breadth-First Search (BFS) strategy, as employed in the K-core view encoder. After decomposition, a compressed graph is constructed where each subgraph is represented as a super-node, and GCN, followed by pooling, is applied on both the subgraphs and the compressed graph. The resulting representations are concatenated to produce the final output of the K-truss view encoder.

5.2.4.3 Contrastive Learning

To optimize the consistency between the representations generated by the K-core and K-truss view encoders, we apply contrastive learning. The objective is to maximize similarity between the embeddings from both views of the same graph while distinguishing them from representations of different graphs. The contrastive loss is formulated as:

$$\mathcal{L}_{contra} = -\log \left(\frac{\text{sim}(Z_{core}, Z_{truss})}{\sum_{j=1}^N \text{sim}(Z_{input}, Z_j)} \right) \quad (5.8)$$

Here, $\text{sim}(\cdot)$ represents the cosine similarity function. Z_{core} and Z_{truss} denote the graph representations from the K-core and K-truss encoders, respectively, and Z_j refers to the embedding of the j -th graph in the batch.

After generating the representations from both encoders, we combine them into a final graph representation, which is then passed through a multi-layer perceptron (MLP) to predict the graph label. The supervised loss is computed using the standard cross-entropy objective:

$$\begin{aligned} \hat{Y} &= \text{MLP}(Z_{final}) \\ \mathcal{L}_{sup} &= - \sum Y \log \hat{Y} \end{aligned} \tag{5.9}$$

where Z_{final} is the fused graph representation, \hat{Y} is the predicted label, and Y is the true graph label.

The overall training objective is a weighted sum of the supervised loss and the contrastive loss:

$$\mathcal{L} = \alpha \mathcal{L}_{sup} + (1 - \alpha) \mathcal{L}_{contra} \tag{5.10}$$

where α is a tunable hyperparameter that balances the contribution of each loss component during training.

5.3 Experiments

We evaluate the effectiveness of our proposed model on the graph classification task using six benchmark datasets. We compare our approach against state-of-the-art baselines for graph classification. Additionally, we conduct ablation studies by removing different components of our model. Also, check the effect of message passing model by varying the message-passing

model used in our framework and examining the influence of contrastive loss by varying the value of α . Additionally, we als.

5.3.1 Datasets:

We use six publicly available datasets summarized in Table 6.1, all of which are commonly used for benchmarking graph classification models [89]. These datasets include three social network datasets (IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, and COLLAB) and two biological or chemical datasets (D&D, and PROTEIN). Each dataset differs in terms of graph size, number of nodes and edges, and class distribution, providing a diverse evaluation scenario.

Table 5.1 Dataset statistics. $|G|$ is the number of graphs, V_{avg} and E_{avg} denote the average number of nodes and edges per graph, and $|C|$ is the number of classes.

| Dataset | $ G $ | V_{avg} | E_{avg} | $ C $ |
|---------------|-------|-----------|-----------|-------|
| D&D | 1178 | 284.32 | 715.66 | 2 |
| PROTEIN | 1113 | 39.06 | 72.82 | 2 |
| IMDB-BINARY | 1000 | 19.77 | 96.53 | 2 |
| IMDB-MULTI | 1500 | 13.00 | 65.94 | 3 |
| REDDIT-BINARY | 2000 | 429.63 | 497.75 | 2 |
| COLLAB | 5000 | 74.49 | 2457.78 | 3 |

The datasets vary significantly in size and domain. For instance, REDDIT-BINARY and COLLAB contain large graphs representing user interactions, while D&D and PROTEIN contain more complex molecular structures with rich biological features. This diversity ensures a robust evaluation of our model across different types of graph data.

Table 5.2 Comparison of our models with baseline methods on Graph Classification.

| Model | IMDB-B | IMDB-M | REDDIT-B | COLLAB | D&D | PROTEIN |
|------------|--------------|--------------|--------------|--------------|--------------|--------------|
| gPool | 73.40 | 50.27 | 74.70 | 77.58 | 75.0 | 71.9 |
| SAGPool | 73.00 | 49.43 | 84.66 | 70.10 | 75.7 | 72.5 |
| MinCutPool | 70.78 | 52.8 | 75.67 | 69.91 | 76.7 | 76.3 |
| DiffPool | 68.40 | 45.62 | 66.65 | 74.83 | 66.9 | 68.2 |
| ASAP | 72.74 | | - | 78.95 | 76.9 | 74.2 |
| MPool | 74.20 | 50.26 | 84.10 | 83.62 | 81.2 | 79.30 |
| InfoGraph | 71.34 | 47.93 | 82.50 | 72.36 | 72.85 | 74.09 |
| GraphCL | 71.48 | 48.11 | 89.53 | 72.36 | 78.62 | 74.32 |
| Our Model | 78.80 | 52.88 | 89.65 | 86.70 | 83.43 | 78.88 |

5.3.2 Baseline:

We use six graph pooling methods and two graph contrastive learning methods as baseline methods. Among them, gPool [14] SAGPool [15], MinCutPool [16], DiffPool [62], ASAP [63] and MPool [86] are hierarchical graph pooling method. InfoGraph [28], and GraphCL [96] are graph contrastive learning methods.

5.3.3 Experimental Setup:

For each dataset, we perform 10 independent random splits using different seed values. The data is divided into 80% for training, 10% for validation, and 10% for testing. We implement our model using PyTorch and PyTorch Geometric. The model is optimized using the Adam optimizer [90]. Node embedding dimensionality is fixed to 128 across all datasets.

We tune hyperparameters using grid search over the following ranges: learning rate $\{1e-2, 5e-2, 1e-3, 5e-3, 1e-4, 5e-4\}$, weight decay $\{1e-2, 1e-3, 1e-4, 1e-5\}$. Each model is trained for a maximum of 100K epochs, with early stopping triggered if the validation loss does not improve over 50 consecutive epochs.

5.3.4 *Result:*

Table 5.2 presents the classification accuracy (%) of our proposed model compared to several state-of-the-art graph pooling and contrastive learning methods across six widely-used benchmark datasets. Our model demonstrates superior performance across all datasets, underscoring the effectiveness of combining K-core and K-truss structural views with contrastive learning.

On the **social network datasets** (IMDB-BINARY, IMDB-MULTI, and REDDIT-BINARY), our model achieves the best accuracy of 78.80%, 52.88%, and 89.65%, respectively. These results indicate that our model effectively captures both local and higher-order structural patterns crucial for social graph analysis. Notably, although GraphCL performs competitively on REDDIT-BINARY (89.53%), our model slightly outperforms it, validating the benefit of dual-view alignment.

For the **collaboration network** dataset COLLAB, our model obtains an accuracy of 86.70%, significantly outperforming strong baselines such as MPool (83.62%) and ASAP (78.95%). This demonstrates the utility of subgraph-based compression in modeling complex, large-scale interaction graphs.

On the **biological and chemical datasets** (D&D and PROTEIN), our model achieves 83.43% and 78.88%, respectively. While MPool attains a slightly higher score on PROTEIN (79.30%), our method provides the most consistent performance across all datasets. The strong results on these structurally diverse datasets confirm the robustness and generalizability of our approach.

Table 5.3 Performance comparisons with the main model (DGCL), the model with supervised loss ($DGCL_{sup}$), the model with K-core view encoder ($DGCL_{core}$), and the model with K-truss view encoder ($DGCL_{truss}$).

| Dataset | GCLC | GCLC _{sup} | GCLC _{core} | GCLC _{truss} |
|----------|--------------|---------------------|----------------------|-----------------------|
| IMDB-B | 78.80 | 75.10 | 74.95 | 75.07 |
| IMDB-M | 52.88 | 50.26 | 50.15 | 49.88 |
| REDDIT-B | 89.65 | 88.20 | 86.32 | 87.54 |
| COLLAB | 86.70 | 84.25 | 83.65 | 83.79 |
| DD | 83.43 | 80.85 | 79.54 | 80.32 |
| PROTEIN | 78.88 | 77.63 | 75.77 | 75.56 |

Overall, these results highlight that our contrastive graph learning framework—leveraging structure-aware K-core and K-truss decompositions—learns more expressive and discriminative representations for graph classification tasks across various domains.

5.3.4.1 Ablation Studies

To further understand the contributions of different components and design choices in our framework, we conduct a series of ablation studies. The results are summarized in Tables 6.6–5.5.

Component Analysis. Table 6.6 compares the performance of our full model (GCLC) with three ablated variants: GCLC_{sup} (trained using only supervised loss), GCLC_{core} (using only the K-core view encoder), and GCLC_{truss} (using only the K-truss view encoder). Across all datasets, the full model outperforms its variants, demonstrating the effectiveness of combining contrastive learning with dual-view structural compression. Both GCLC_{core} and GCLC_{truss} underperform the full model, indicating that integrating both views contributes to richer representation learning. The contrastive objective also plays a key role, as evidenced by the performance gap between GCLC and GCLC_{sup}.

GNN Variants: Table 6.3 examines the impact of different GNN architectures (GCN,

Table 5.4 Performance comparisons of different GNN models on GCLC for DD and IMDB datasets.

| Dataset | GCN | GAT | Graph-SAGE |
|---------|--------------|--------------|-------------------|
| D&D | 83.43 | 82.75 | 83.26 |
| IMDB-B | 78.80 | 78.87 | 78.68 |

Table 5.5 Performance comparisons of different values on α on GCLC for DD and IMDB datasets.

| Dataset | $\alpha = 0.5$ | $\alpha = 0.6$ | $\alpha = 0.7$ | $\alpha = 0.8$ | $\alpha = 0.9$ |
|---------|----------------|----------------|----------------|----------------|----------------|
| D&D | 83.35 | 83.43 | 82.88 | 82.51 | 80.84 |
| IMDB-B | 78.80 | 78.81 | 77.74 | 77.28 | 76.85 |

GAT, and GraphSAGE) on the D&D and IMDB-B datasets. While all three models achieve competitive results, GCN yields the highest accuracy on D&D (83.43%), whereas GAT performs slightly better on IMDB-B (78.87%). These results indicate that our framework is adaptable to various GNN backbones and consistently delivers strong performance.

Effect of Contrastive Loss Weight α . Table 5.5 shows the results for varying values of the loss weight parameter α , which controls the balance between supervised and contrastive learning objectives. The best performance is achieved when $\alpha = 0.6$ on both D&D and IMDB-B datasets. While lower or higher values of α still produce reasonable results, extreme values (e.g., $\alpha = 0.9$) tend to degrade performance, suggesting that an appropriate trade-off between the two objectives is essential for optimal learning.

CHAPTER 6

DyGCL: Dynamic Graph Contrastive Learning For Event Prediction

6.1 Introduction

This paper is published in IEE BigData 2024 [98]. In this work, we introduce a **Dynamic Graph Contrastive Learning (DyGCL)** method that learns dynamic graph representation for event prediction. The main contributions of our work can be outlined as follows:

- We present an innovative framework called Dynamic Graph Contrastive Learning (DyGCL), expressly developed for event prediction. DyGCL considers both the local and global structures of input graphs to comprehend the temporal dependency for event forecasting via the local view and the global view encoders, respectively. The local view encoder learns the graph representation that captures the local structure of the graph in each time-stamp graph, considering the previous time-stamp representations. In contrast, the global view encoder infers a unified representation for each time-stamp graph with dynamic pooling and utilizes them with an RNN model to capture the global structure in all time-stamp graphs.
- The DyGCL framework utilizes contrastive learning while optimizing two representations coming from the local view encoder and the global view encoder. With contrastive learning, while capturing the structural features in an unsupervised way, we maximize the similarity between representations obtained from both encoders for feature consistency, which enhances the overall effectiveness of the model.

- We perform comprehensive experiments to show the effectiveness of our model on six different event datasets. We compare the results of our model against several state-of-the-art deep learning and GNN-based baseline models for event prediction. The experiments demonstrate that our model outperforms these baseline models.

6.2 Methodology

In this section, we first present some primary concepts and terminology about dynamic graphs and contrastive learning and then define the event prediction problem. Then we explain our Dynamic Graph Contrastive Learning (DyGCL) model including local and global view encoders, optimized with contrastive loss.

6.2.1 Preliminaries and Problem Description

For an event at a specific location, historical event-related articles reveal important information about the rising event and they are used to predict future events. In this project, event-related articles are encoded into a sequence of graphs as a dynamic graph where each graph represents the contextual information from a specific timestamp. While nodes in the graph represent words occurring in the articles of that timestamp, edges between nodes represent the occurrence of the words in a predefined fixed-size window. Specifically, one graph is created for each day, and data from k consecutive days is represented as the dynamic graph and used to learn and predict whether an event will occur on $(k + 1)$ th day.

Here, we first define dynamic graphs and then event prediction via dynamic graphs.

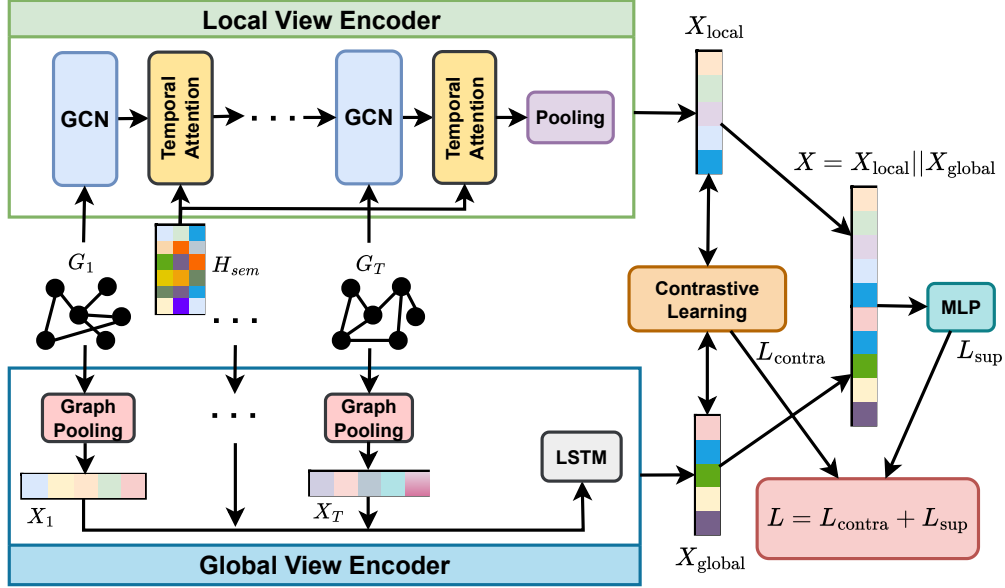


Figure 6.1 An overview of Dynamic Graph Contrastive Learning, (DyGCL) architecture. We feed input graphs into the Local View Encoder and Global View Encoder. Local View encoder learns the dynamic graph representation through the dynamic node representation. Global-View Encoder learns dynamic graph representation through graph pooling and the LSTM model. After optimizing representations by contrastive learning, they are combined by an MLP layer and fed to the predictor for event prediction.

Definition 7 (Dynamic Graph). A dynamic graph \mathbb{G} is defined as a series of T discrete snapshots denoted as $\mathbb{G} = \{G_1, G_2, \dots, G_T\}$, where G_t represents the graph at timestamp t .

Each G_t has an adjacency matrix A_t showing the relation between nodes at time t

We define the event prediction problem as the binary classification problem and predict whether there is an event on day $t + 1$ using data from t previous days. Formal definition of the event prediction via dynamic graphs is given as follows;

Definition 8 (Event Prediction). Given a training dataset D , where each sample is represented as $D[\mathbb{G}_i]$ for $i = 1, 2, 3, \dots, m$ and each and each \mathbb{G} is a dynamic graph $\mathbb{G} = \{G_1, G_2, \dots, G_T\}$ with initial node feature matrix $X \in R^{N \times d}$ where N is the number of nodes

with d dimension at time t , our goal is to learn a graph encoder that maps the input dynamic graph into vector representation and use this representation to predict the future event \hat{y} at time $T + 1$.

For our dynamic graph datasets, which are constructed from the text of event-related articles, we have one global initial node feature matrix H_{sem} representing the semantic meaning of all words appearing over the time obtained with a word embedding model. H_t can be obtained by filtering the words occurring at time t .

Graph Contrastive learning: Contrastive learning (CL) has become one of the most popular approaches for unsupervised representation learning, which learns through comparisons among different views. In general, there are three parts in graph contrastive learning: **i)**

View Construction: A view is represented as graph data, denoted as $vi = (A_i, X_i)$, where $A_i \in \mathbb{R}^{N \times N}$ and $H_{(i)} \in \mathbb{R}^{N \times d}$. Different views of a given graph are created with augmentation approaches like node dropping [94], edge perturbation [95], and subgraph sampling [97]. **ii)**

View Encoder: View encoder learns the node representation or graph representation from different views of input graphs using a GNN model. **iii) Contrastive loss:** The model is optimized by a contrastive loss with an objective of maximizing the similarity among different view representations.

6.2.2 Dynamic Graph Contrastive Learning

Our novel Dynamic Graph Contrastive Learning (DyGCL) model is specifically designed to optimize dynamic graph representation learning, particularly for event prediction tasks. The overall architecture for the model is presented in Figure 6.1. The model consists of three

main components as detailed in Algorithm 3; i) Local view encoder ii) Global view encoder, and iii) Contrastive learning. The local view encoder extracts the local structural information from the dynamic graphs using a Dynamic Graph Convolutional Network. The global view encoder captures the global structure using Dynamic Graph Pooling and a recurrent neural network. Then, the graph representations from both encoders, optimized via contrastive learning to maximize the similarity between them, are combined with an attention mechanism as the final dynamic graph representation and utilized for event prediction.

Algorithm 3: Dynamic Graph Contrastive Learning (DyGCL)

Input: Initial Node Features matrix $H_{(sem)}$, Temporal Graphs, Event Label Y

Output: Predicted Event label \hat{y}

$\mathcal{G} \leftarrow$ randomly sample a batch

for i *in* \mathcal{G} **do**

$A_{T-k, \dots, T}, H_{sem} = \mathbb{G}_i Z_{local}, H_L = LVE(A_{T-k, \dots, T}, H_{sem})$

//Local view graph representation

$Z_{global} = GVE(A_{T-k, \dots, T}, H_L)$

//Global view graph representation

$L_{contra} = \frac{Z_{local} Z_{global}^T}{\|Z_{local}\| \|Z_{global}\|}$ //Contrastive Loss

$Z_{local} = Z_{local} W_l + b_l$

$Z_{global} = Z_{global} W_g + b_g$

$Z_{Final} = MLP([Z_{local} \| Z_{global}])$

$\hat{Y} = \sigma(MLP(Z_{Final}))$

$\mathcal{L}_{sup} = -\sum Y \log \hat{Y}$

$L = \alpha \mathcal{L}_{sup} + (1 - \alpha) L_{contra}$

end

6.2.2.1 Local View Encoder

In general, a graph-level representation is derived from the node-level representations of the graph. Therefore, obtaining an improved graph representation hinges on having superior node representations. To enhance node representation, it is crucial to consider the local

structure of nodes, including neighborhood relationships and interactions with neighbors. To learn better node representation by preserving the local structure of the nodes and getting graph-level representation from the node-level representation, we introduce a local view encoder in Algorithm 4.

Algorithm 4: Local View Encoder (LVE)

Input: Initial Node Features matrix H_{sem} , Temporal Graphs.

Output: Local view Graph Representation Z_{local} , Updated node Features H

for each $t \leftarrow T - k$ **to** T **do**

$$\left| \begin{array}{l} H^{(t)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A}^t \tilde{D}^{-\frac{1}{2}} H^{(t-1)} \theta^{(t-1)}) \\ H^{(t)'} = H^{(t)} W_s^t + b_s^t \\ H_{(sem)'} = H_{sem} W_0^t + b_0^t \\ H^{(t)} = \tanh([H^{(t)} || H_{sem}]) \end{array} \right.$$

end

$$Z_{local} = \frac{1}{N} \sum_{i=1}^N [h_i]$$

In the local view encoder, we use a Dynamic Graph Convolution Network (DyGCN) that learns dynamic node representation, preserving the temporal local structure for dynamic graphs. It passes the information from the previous time step to the next step to capture the dynamically changing neighboring structures among the input graphs. DyGCN comprises two layers: the static GCN layer and the temporal attention layer. For every time step graph, we apply the static GCN to learn node representations based on the current graph structure and also node features coming from the previous time step. This allows us to encapsulate the local structure of nodes at each time step into vector representations. Within each time step, the static GCN learns the node representation using a message-passing approach, which is described as

$$H^{(t)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(t-1)} \theta^{(t-1)}) \quad (6.1)$$

where $H^{(t)}$ represents the node representation matrix for the t^{th} snapshot graph, σ is an activation function, $\tilde{A} = A + I$ is the adjacency matrix with self-loop, $\tilde{D} \in \mathbb{R}^{N \times N}$ is the normalized degree matrix of \tilde{A} , $\theta^{(t)}$ is the trainable weight for $t^{(th)}$ time step and $H^{(t-1)}$ is the input node representation matrix for t^{th} time steps obtained from previous time step’s Dynamic GCN.

While the topological structure of the graph and its changes over time are important features to detect events, semantic meanings of the words are also important for events. On the other hand, when GCN is applied at each time epoch, it updates node representations with the current time epoch neighborhood. Over multiple timestamps, there is a risk of losing essential semantic details of nodes, potentially leading to the over-smoothing issue. To address this challenge, we create a temporal attention layer. We define initial node features, H_{sem} with their word embeddings, which represent their semantic meanings. After obtaining node representations from the current time epochs’ GCN, we refine these representations through a temporal attention layer with initial semantic representations, H_{sem} . In this layer, we combine the node representation from the current timestamp’s GCN and the initial semantic node features using a linear neural network layer. First, we multiply them by learnable weight matrices that signify the importance of each representation for event prediction. Then we concatenate weighted representations and give it to an activation function. Here we use *tanh* as the activation function. By refining the node representation

obtained from GCN with the initial semantic word embedding in each temporal attention layer, we facilitate the fusion of semantic attributes of the nodes. The temporal attention layer applied at each time step is formalized as follows:

$$\begin{aligned}
 H^{(t)'} &= H^{(t)}W_s^t + b_s^t \\
 H_{sem'} &= H_{sem}W_0^t + b_0^t \\
 H^{(t)} &= \tanh([H^{(t)'} || H_{sem'}])
 \end{aligned} \tag{6.2}$$

where $H^{(t)}$ is the embedding matrix from GCN layer at time t , $H_{(sem')}$ is the initial node embeddings or pre-trained word embeddings, W_s^t, b_s^t and W_0^t, b_0^t are learnable parameters for $H^{(t)}$ and $H_{(sem')}$, respectively, and $||$ is the concatenation operation.

After updating node embeddings using the temporal attention layer, we pass it as initial node embedding to the next GCN layer at the time stamp $t + 1$. After getting the final node representation from the Dynamic Graph Convolution layer at the last time step, we convert the node representations into graph representations using global mean pooling, which takes the average of node features defined as follows:

$$Z_{local} = \frac{1}{N} \sum_{i=1}^N [h_i] \tag{6.3}$$

where N is the number of nodes, h_i is the i^{th} node feature.

6.2.2.2 *Global View Encoder*

The local view encoder is primarily engineered to derive graph-level representation from the dynamic node-level representations from temporal graphs. However, this node-centric

approach tends to capture only the local structures of input graphs, often sidelining the global structures. While local structures show close-by connections, global structures show bigger patterns across the entire graph. For predicting events, these big patterns are also important. Sometimes, events happen because of connections that aren't right next to each other. If we only focus on the local structure, we might miss these important clues. Thus, for predicting events, it is very important to consider global structures.

To capture higher-order features in input graphs, we introduce a global view encoder in Algorithm 5 where we use a Dynamic Graph Pooling module. At first, we learn a hierarchical graph representation for each time snapshot graph. These graph representations are then refined by a recurrent neural network, considering both the current and previous temporal graph representations to capture relations between them and also to capture the changes from one to another.

To learn the hierarchical graph representation, selection-based and clustering-based pooling models can be applied. Here, we employ the SAGPool model as a selection-based graph pooling method [15]. Selection-based methods are chosen for their memory efficiency and

Algorithm 5: Global View encoder (GVE)

Input: Node Features matrix H , Temporal Graphs.

Output: Global view graph representation Z_{global}

for each $t \leftarrow T - k$ to T **do**

$S = \sigma(\text{GNN}(H^t, A^t, \theta_{att}^t))$
 $idx = \text{topK}(S, [\alpha \times N])$
 $A^{(l+1)} = A_{idx, idx}$
 $Z_t = \frac{1}{N} \sum_{i=1}^N x_i \| \max_{i=1}^N(x_i)$

end

$Z_{\text{global}} = \text{RNN}(Z_{T-k}, \dots, Z_T)$

emphasis on the global structure of the input graph. In SAGPool, we select the top k important nodes from the input graph that are deemed relevant to the event and keep these nodes and relations between them for the next layer. The selection of the top k nodes is determined by calculating attention scores for all nodes, leveraging node features, the GNN model, and an attention parameter, as follows:

$$S = \sigma(\text{GNN}(H^t, A^t, \theta_{att}^t)) \quad (6.4)$$

Where $S \in \mathbb{R}^{N \times 1}$ represents the node attention scores, H^t denotes the node embeddings at time t , A^t is the adjacency matrix, $\theta_{att}^t \in \mathbb{R}^{d \times 1}$ is the learnable parameter matrix at time t , and N is the number of nodes.

Following the calculation of attention scores, we select the top k nodes with the highest scores. Subsequently, we construct a new coarse graph using the selected nodes, as outlined below:

$$\begin{aligned} idx &= \text{topK}(S, [\alpha \times N]) \\ A^{(l+1)} &= A_{idx, idx} \end{aligned} \quad (6.5)$$

where idx is the indices of top- k nodes, α is the pooling ratio, N is the number of nodes, and $A^{(l+1)}$ is the coarse graph. Multiple graph pooling and GCN layers on each snapshot graph are applied to get a hierarchical graph representation.

As the final layer of the hierarchical graph representation learning, we apply a readout layer to get a fixed-sized graph-level representation for each snapshot graph. The readout

function aggregates the node features as follows:

$$Z_t = \frac{1}{N} \sum_{i=1}^N [h_i || \max_{i=1}^N h_i] \quad (6.6)$$

where N is the number of nodes, h_i is the i^{th} node feature and $||$ denotes concatenation. The graph pooling method effectively captures the static global structure of the current time epoch graph. Yet, to grasp the dynamic evolution of the global structure in temporal graphs, it is crucial to update the current time epoch graph representation with that of the previous time epoch graphs. For this purpose, a Recurrent Neural Network (RNN) layer is employed. RNNs, well-suited for sequential data, update the current data representation by considering the input data from the preceding steps in the sequence. In our approach, we feed graph representations from the pooling layers into the RNN layer in the following manner:

$$Z_{global} = RNN(Z_{t-k}, \dots, Z_t) \quad (6.7)$$

where z_{t-k} is the graph-level representation of k -previous time steps from current time graph. The output of the RNN layer is used as the global graph features.

6.2.2.3 *Contrastive Learning*

While the local view encoder and global view encoder extract the different features of the dynamic graph, their final representation should be similar as they belong to the same data. Therefore, we apply contrastive learning to make them similar. Our contrastive learning objective function aims to maximize the cosine similarity or minimize the cosine distance between the graph representations from local and global view encoders.

We use graph representations from the local view encoder and the global view encoder as positive samples if they belong to the same data. In our model, we do not use any negative sample in contrastive learning as Namkyeong Lee et, al [26] mentioned that contrastive learning on graphs performs better without negative samples. As our objective, we minimize the cosine distance between positive samples. We define our objective function as follows:

$$L_{\text{contra}} = \frac{Z_{\text{local}} Z_{\text{global}}^T}{\|Z_{\text{local}}\| \|Z_{\text{global}}\|} \quad (6.8)$$

Where Z_{local} is the graph-level representation from local view encoder and Z_{global} is the graph-level representation from global view encoder

6.2.2.4 *Output layer*

In our approach, the graph-level outputs from both the local view encoder and global are integrated, allowing us to embed both temporal local and global graph structural information into a singular graph-level embedding. The amalgamation of these embeddings is facilitated by an MLP layer. To combine local and global view representations, we first employ two distinct learnable weights for the two embeddings, allocating specialized attention to each. Once the embeddings are individually multiplied by their respective learnable weights, they are concatenated. Subsequently, an activation function is employed to seamlessly merge them, as outlined below

$$\begin{aligned}
Z_{local} &= Z_{local}W_l + b_l \\
Z_{global} &= Z_{global}W_g + b_g \\
Z_{Final} &= \tanh([Z_{local}||Z_{global}])
\end{aligned} \tag{6.9}$$

where Z_{local} is graph embedding matrix from local view encoder, Z_{global} is graph embedding from global view encoder, W_l, b_l , and W_g, b_g are learnable parameters for Z_{local} and Z_{global} , respectively, $||$ is the concatenation operation and \tanh is an activation function of the linear layer.

After combining both graph representations into one final graph representation, our model gives it as an input to a multilayer perception layer with the sigmoid function to predict the event occurrence and calculate the supervised loss as follows:

$$\begin{aligned}
\hat{Y} &= \sigma(MLP(Z_{Final})) \\
\mathcal{L}_{sup} &= - \sum Y \log \hat{Y}
\end{aligned} \tag{6.10}$$

where Z_{Final} is the graph representation, \hat{Y} is the predicted event occurrence, and Y is the actual event occurrence.

We jointly train our model with a weighted sum of the supervised loss and contrastive loss as follows:

$$L = \alpha L_{sup} + (1 - \alpha) L_{contra} \tag{6.11}$$

where α is a hyperparameter of the model.

Table 6.1 Dataset statistics. $|S|$ is the number of samples, \bar{N} and \bar{E} are the average number of nodes and edges, respectively, and $|Ev|$ is the number of events.

| Datasets | $ S $ | \bar{N} | \bar{E} | $ Ev $ |
|----------|-------|-----------|-----------|--------|
| Thailand | 1883 | 600 | 7281 | 715 |
| Egypt | 3788 | 675 | 9680 | 1469 |
| Russia | 3552 | 645 | 9776 | 1171 |
| India | 12249 | 685 | 12994 | 4586 |
| NYC Cab | 4464 | 263 | 3717 | 162 |
| Twitter | 2557 | 1000 | 10312 | 287 |

Table 6.2 Performance comparisons of our model with baseline models on event prediction

| Method | Model | Thailand | Egypt | Russia | India | NYC Cab | Twitter Weather |
|---------|--------------|--------------|--------------|--------------|--------------|--------------|-----------------|
| Static | GCN | 76.13 | 75.8 | 76.63 | 67.45 | 84.91 | 77.21 |
| | TopKpool | 77.03 | 85.28 | 78.45 | 65.53 | 86.45 | 78.65 |
| | SAGPool | 77.74 | 86.12 | 80.86 | 68.50 | 90.82 | 80.78 |
| | DiffPool | 76.13 | 82.8 | 79.63 | 67.48 | 88.70 | 76.6 |
| | MPool | 77.85 | 85.64 | 80.52 | 69.10 | 89.88 | 82.58 |
| Dynamic | GCN+GRU | 79.28 | 83.88 | 79.66 | 67.48 | 85.00 | 76.50 |
| | GCN+LSTM | 78.13 | 83.05 | 79.38 | 68.10 | 85.07 | 76.55 |
| | EvolveGCN | - | - | - | - | 84.20 | 78.24 |
| | DynamicGCN | 80.92 | 84.71 | 84.71 | 68.70 | 81.00 | 71.30 |
| | DDGCN | 79.10 | 85.25 | 86.32 | 70.70 | 80.92 | 78.15 |
| | DyGED | 73.50 | 85.41 | 81.43 | 68.88 | 91.20 | 81.00 |
| | DyGCL (Ours) | 86.57 | 89.28 | 88.95 | 76.85 | 95.80 | 90.68 |

6.3 Experiment

In assessing our model’s performance for the event prediction task, treating it as a dynamic graph classification problem, we conduct a comprehensive evaluation. We compare our model’s performance against six distinct baseline models. Additionally, we delve into the analysis of the impact of the number of historical days on event prediction. Furthermore, we present results for variations of our model, incorporating different message-passing models and graph-pooling methods. We also visualize the global structure of temporal graphs.

6.3.1 Datasets

In our experiments, we use six datasets. Among them, four of them are social event datasets, one is a weather event dataset and the other is a traffic event dataset. Table 6.1 shows the

statistics of six datasets where $|S|$ represents the number of samples for each dataset. For all datasets, each sample contains seven snapshot graphs where \bar{N} and \bar{E} are the average number of nodes and edges of each snapshot graph.

Thailand, Egypt, Russia, and India event datasets are collected from the Integrated Conflict Early Warning System (ICEWS)[99, 34]. These datasets include information on political events and are designed for assessing both national and international crises. We concentrate on data sourced from major cities such as Delhi, Mumbai, Kolkata, and others in India, Bangkok in Thailand, Cairo in Egypt, and Moscow in Russia. Rallies, strikes, violent protests, and passage obstructions are frequent event types on these datasets.

The **NYC Cab** represents a mobility network that contains geo-tagged mass-gathering events, such as concerts and protests. The **Twitter Weather** dataset showcases user-generated content pertaining to weather events, including storms and earthquakes.

6.3.2 Baseline

We use several graph neural network methods as baseline methods which mainly focus on Dynamic graphs or static graph classification. We divide our baseline methods into two classes: (1) Static methods, and (2) Dynamic methods.

6.3.2.1 Static Methods

We compare our model with GCN[7], TopKpool[14], SAGPool[15], DiffPool [62] and MPool [86], which are most common graph representation learning methods. For the static methods, we combine all time step graphs in a dynamic graph into one graph for each sample.

- *GCN*: Graph Convolutional network is the most popular GNN model for node representation learning. GCN aggregates the neighborhood information to update the node representation. After getting node representation we apply global pooling described in equation 6.3 to get graph-level representation.
- *TopKPool*: TopKPool is a hierarchical graph pooling method that selects top- k nodes for pooling operation. It considers the topological structure of the graph to select the top- k nodes.
- *SAGPool*: SAGPool is also a hierarchical graph pooling method that selects top- k nodes using the self-attention method to select top- k nodes for pooling operation.
- *DiffPool*: DiffPool is a cluster-based hierarchical graph pooling method. It uses a Graph Neural Network to learn a cluster assignment matrix of the input graph. Then it uses the cluster assignment matrix for graph pooling where it combines the nodes in each cluster into supernodes for the next layer.
- *MPool*: MPool hierarchical graph pooling method where it uses motif structure to capture the higher-order structure of input graph. It uses both clustering and selection approaches to learn the graph-level representation.

6.3.2.2 Dynamic Methods

For the dynamic baseline models, we use GCN+LSTM [100], GCN+GRU [100], DynamicGCN [34], EvolveGCN [101], DDGCN[102] and DyGED [35].

- *GCN+LSTM*: This model learns the dynamic graph representation using GCN and LSTM models. GCN+GRU is a variation of this model where it replaces LSTM with the GRU model. It is designed as a temporal graph neural network to predict traffic conditions in the traffic network.
- *DynamicGCN*: This model applies GCN and a temporal layer for each snapshot to learn dynamic node representations. It converts dynamic node representation to graph representation using a mask linear layer. It is designed for event prediction on social media data.
- *EvolveGCN*: EvolveGCN is a dynamic graph convolution network that uses GCN and Recurrent Neural Network (RNN) to learn the representation of dynamic networks. It transfers the parameter matrix from one timestamp's GCN to another timestamp's GCN. It is a more general model that is applied to edge prediction, edge classification, and node classification for dynamic graphs. Here we also use the global pooling method in the last layer of the model to convert node representation to graph representation.
- *DDGCN*: DDGCN utilizes a dual channel dynamic graph convolutional network where it uses dynamic graph and dynamic knowledge graph to learn the dynamic graph level representation.
- *DyGED*: DyGED is the most recent event prediction model that uses global graph pooling on each time step graph and applies RNN models to include macro-level graph dynamics for graph representation learning.

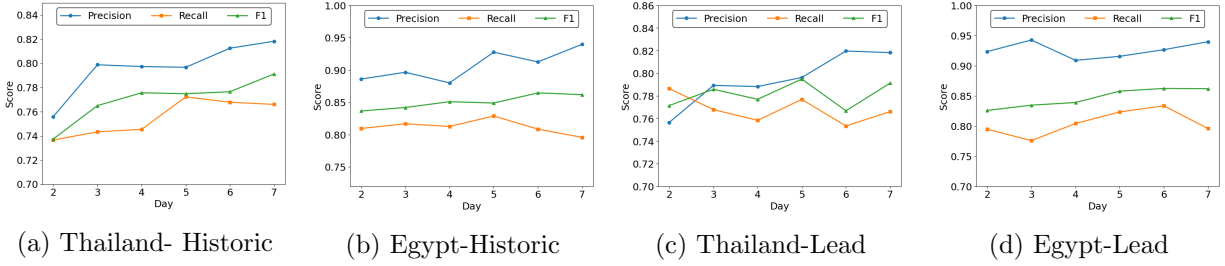


Figure 6.2 Detailed Event prediction results of DyGCL model for the different number of historic and lead days.

6.3.3 Experimental Settings

To assess our model for the event prediction task, we partition the data into three segments: 70% for training, 15% for validation, and 15% for testing. This splitting process is repeated ten times using ten different random seeds. We take the average of 10 different runs as the final results. Our model is implemented using PyTorch and PyTorch Geometry library, with the optimization performed by the Adam Optimizer. In order to determine optimal hyperparameters, we conduct grid search within the specified ranges: learning rate in $\{1e - 2, 5e - 2, 1e - 3, 5e - 3, 1e - 4, 5e - 4\}$, weight decay in $\{1e - 2, 1e - 3, 1e - 4, 1e - 5\}$, pooling ratio in $\{1/2, 1/4\}$, dropout ratio fixed at 0.2, and hidden size in $\{16, 32, 64, 128\}$. The training process halts if the validation loss fails to improve for 50 epochs. For the initial node embeddings of the dynamic graph, we use a 100-dimensional word embedding vector obtained through the Word2Vec method.

6.3.4 Result

In Table 6.2, we present a summary of our event prediction accuracy results, with results of Static and Dynamic baseline models. We treat the event prediction task as a binary graph

classification problem, where class labels are $\{0, 1\}$ indicating whether an event occurs on that day or not. The EvolveGCN model is applicable to datasets where all samples have the same number of nodes. This constraint limits the availability of results to only NYC Cab and Twitter weather datasets that meet this requirement. For other datasets, the number of nodes is different for different samples.

As we see from Table 6.2, our model consistently works well and outperforms all baseline models across all datasets. Notably, our model improves the highest accuracy of baselines by 6.9% for the Thailand dataset with 86.57% accuracy, 3.1% for Egypt with 89.95% accuracy, 3.0% for Russia with 88.95% accuracy, 11.5% for India datasets with 76.85 accuracy, 4.6% for NYC Cab dataset with 95.80 accuracy and 9.6% for Twitter Weather dataset with 90.68 accuracy. For the Russia and India datasets, the DDGCN model achieves the second-highest accuracy. For the Thailand dataset, DynamicGCN gives better accuracy than the other baselines whereas SAGPool demonstrates the second-highest accuracy for the Egypt dataset. For NYC Cab and Twitter weather datasets, DyGED gets the second-highest accuracy.

Additionally, we observe that dynamic methods generally outperform static baseline methods for all datasets, with the exception of the Egypt dataset where SAGPool, as the static graph pooling method, performs better than other baselines. This result suggests the importance of global structural information for the Egypt dataset.

Sensitivity Analysis: In our previous experiment, we used the previous 7 days' data to make a prediction on Day 8. In this experiment, we investigate the prediction performance by varying both the number of prior days and lead time. Historical days denote the previous

number of days used as the training data. We want to see whether there is an effect of the number of historical days preceding the event on the event prediction task by changing it from 2 to 7. In Figure 6.2, we present precision, recall, and F1-score for different numbers of days for the event using the DyGCL model for the Thailand and Egypt datasets. In this figure, “day 2” denotes the use of data from the last two preceding days (6th and 7th days) for predicting events on the 8th day. We can see that for both datasets the Precision and F1-score increase gradually when we increase the number of historic days for event prediction. For both datasets, we get the highest score for “day 7”.

In our experiment, we also explore the effect of the lead day on the model performance by varying the number of lead days from 1 to 7. We present precision, recall, and F1-score for different numbers of days for the event using the DyGCL model in Figure 6.2. The lead day signifies how many days in advance our model predicts the event. As an example, when we refer to “day 6”, it implies the utilization of data from the 1st day to the 6th day for predicting the event on the 8th day, indicating a two-day advance prediction.

In Figure 6.2, for Precision and F1-score, DyGCL with “day 7” yields the highest scores for both datasets. In this scenario, DyGCL performs better with a higher number of lead days’ information compared to fewer days. For the Thailand dataset, we get the highest Recall score for “day 5” and for The Egypt dataset DyGCL gives the highest Recall score for “day 6”.

Ablation Study: In our ablation study, we investigate the impact of different parts of the model including contrastive learning, the GNN model in the local view, the RNN layer,

Table 6.3 Performance comparisons of different GNN models on DyGCL for Thailand and Egypt datasets.

| Dataset | Method | Precision | Recall | F1 |
|----------|-----------|---------------|--------------|---------------|
| Thailand | GCN | 0.8182 | 0.766 | 0.7912 |
| | GAT | 0.75 | 0.798 | 0.773 |
| | GraphSAGE | 0.779 | 0.787 | 0.783 |
| Egypt | GCN | 0.9397 | 0.7957 | 0.8618 |
| | GAT | 0.834 | 0.855 | 0.814 |
| | GraphSAGE | 0.864 | 0.835 | 0.847 |

and the pooling layer in the global view, on the model performance. We present results for all ablation studies for Thailand and Egypt datasets as the representative except contrastive learning where we present results for all datasets.

We first experiment with GNN models. In addition to GCN as the default model, we apply two other popular GNN models, which are GAT and GraphSage in the local view encoder to see the effect of them on the model performance. Table 6.3 shows the Precision, Recall, and F1 scores for different GNN models. From the table, we can see that the GNN model has a big impact on the model performance. For both datasets, the GCN model performs much better than the other models, especially with respect to precision.

We also change the hierarchical graph pooling method in the global view encoder to see the effect of different graph pooling methods. In addition to SAGPool pooling as the default one, we select Top- K as the selection-based graph pooling method and DiffPool as the cluster-based pooling method. Table 6.4 shows the Precision, Recall, and F1-score for the Thailand and Egypt datasets. From the table, we can see that selection-based methods outperform the cluster-based method. For the Thailand dataset, the Top- k graph pooling method gives the highest accuracy and for the Egypt dataset, the SAGPool performs better than other methods.

Table 6.4 Performance comparisons of different graph pooling models on DyGCL for Thailand and Egypt datasets.

| Dataset | Method | Precision | Recall | F1 |
|----------|----------|---------------|---------------|---------------|
| Thailand | Top- K | 0.839 | 0.787 | 0.805 |
| | SAGPool | 0.8182 | 0.766 | 0.7912 |
| | DiffPool | 0.8161 | 0.7553 | 0.7845 |
| Egypt | Top- K | 0.847 | 0.826 | 0.836 |
| | SAGPool | 0.9397 | 0.7957 | 0.8618 |
| | DiffPool | 0.8664 | 0.8553 | 0.8608 |

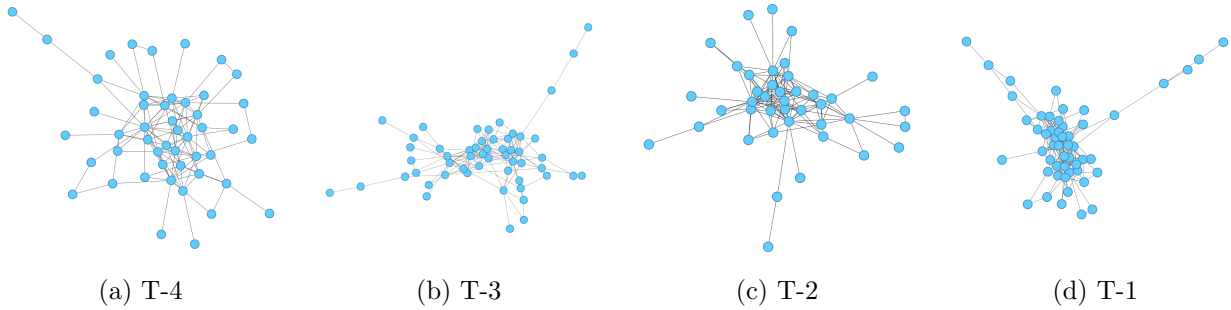


Figure 6.3 Temporal pooled graphs in Global View Encoder for a sample event in NYC cab dataset.

In addition, we use different Recurrent Neural Networks (RNN) in the global view encoder to see their effect on them for the event prediction task. While LSTM is the original RNN model in the global view, we use two different popular RNN models, which are GRU, and Transformer in the global view encoder. Table 6.5 shows the Precision, Recall, and F1 scores for different RNN models. As we can see from the table, the LSTM model gives better results for the Thailand dataset whereas the Transformer model performs better for the Egypt dataset.

Furthermore, we investigate the impact of contrastive loss, Local view encoder, and Global view encoder on event prediction. We evaluate our main model, alongside three variations: DyGCL, DyGCL_{sup}, DyGCL_L, and DyGCL_G. The DyGCL model is our primary approach, whereas DyGCL_{sup} excludes the contrastive loss and directly merges the Local and Global

Table 6.5 Performance comparisons of different RNN models on DyGCL for Thailand and Egypt datasets.

| Datasets | Method | Precision | Recall | F1 |
|----------|-------------|---------------|---------------|---------------|
| Thailand | LSTM | 0.8182 | 0.766 | 0.7912 |
| | GRU | 0.8514 | 0.6702 | 0.75 |
| | Transformer | 0.7765 | 0.7021 | 0.7374 |
| Egypt | LSTM | 0.9397 | 0.7957 | 0.8618 |
| | GRU | 0.9151 | 0.8255 | 0.8680 |
| | Transformer | 0.9112 | 0.8298 | 0.8686 |

Table 6.6 Performance comparisons with the main model (DyGCL), the model with supervised loss (DyGCL_{sup}), the model with local encoder (DyGCL_L), and the model with global encoder (DyGCL_G).

| Dataset | DyGCL | DyGCL _{sup} | DyGCL _L | DyGCL _G |
|----------|--------------|----------------------|--------------------|--------------------|
| Thailand | 86.57 | 85.87 | 80.48 | 74.20 |
| Egypt | 89.28 | 86.64 | 84.20 | 85.77 |
| Russia | 88.95 | 88.37 | 84.58 | 82.45 |
| India | 76.71 | 76.85 | 69.12 | 68.91 |
| NYC | | | | |
| Cab | 90.68 | 89.88 | 80.65 | 93.20 |
| Twitter | | | | |
| weather | 95.80 | 95.74 | 72.50 | 88.70 |

view encoders’ representations using a Multi-Layer Perceptron (MLP) layer. In DyGCL_L only the Local view encoder’s graph representation is utilized for event prediction, and in DyGCL_G only the Global view encoder’s graph representation is used. Unlike other ablation studies, we present the results for all datasets in Table 6.6. The table illustrates that our main model DyGCL which incorporates contrastive loss, achieves superior accuracy compared to the other variations. Additionally, the combined model DyGCL_{sup} which integrates both the Local and Global view encoders, outperforms the single-view models (DyGCL_L and DyGCL_G). These findings justify the significance of combining both view encoders and employing contrastive learning to refine the representation for event prediction.

Visualization: We also visualize the temporal graphs to see the global structure of the temporal graph before an event. For this experiment, we select a sample from the NYC Cab

dataset where an event has occurred. We took four days' temporal graph before that event and applied our pre-trained DyGCLmodel. Then we get the pooled graph from the output of hierarchical graph pooling layers in the global view encoder and present the graphs in Figure 6.3. As we can see from the figure, the global structure of temporal graphs before the event is changing a lot and our model is capturing those structures with the local and global view encoders.

CHAPTER 7

FUTURE WORK

The research presented in this dissertation has significantly advanced graph representation learning in areas such as node classification, graph classification, event prediction, graph pooling, and contrastive learning. However, several promising directions remain for future exploration.

- **Scalability and Efficiency** While models like MPool and DyPool achieve state-of-the-art performance, their computational costs can be high. Future research should focus on optimizing these models for greater scalability and efficiency. Techniques such as model pruning, quantization, and distributed training could reduce resource consumption while maintaining performance. Additionally, designing lightweight yet powerful architectures can further enhance their applicability to large-scale graphs.
- **Explainability and Interpretability** A key challenge in deep learning, including Graph Neural Networks (GNNs), is model interpretability. Future work should prioritize developing methods that enhance the transparency of graph-based models, making their predictions more understandable to users. This could involve designing novel visualization tools, interpretability algorithms, and integrating explainable AI (XAI) techniques into graph learning frameworks.
- **Robustness and Adversarial Resistance** Ensuring the robustness of GNNs against adversarial attacks is crucial for their deployment in critical applications. Future research should investigate vulnerabilities in existing models and develop defense mech-

anisms to improve their resilience. This includes exploring adversarial training techniques tailored for complex graph structures and designing models that can detect and mitigate adversarial manipulations.

By addressing these challenges, future work can further refine graph representation learning, making it more scalable, interpretable, and robust for real-world applications.

CHAPTER 8

CONCLUSION

This dissertation explores various aspects of graph representation learning and its applications across multiple domains. Through three interconnected research projects, we advance the state of the art in graph mining by emphasizing the importance of both local and global structures in static and dynamic graphs. The key contributions and findings from each project are summarized below.

In the first project, We present a novel method for network embedding that preserves the local and global structure of the network. To capture the global structure and accelerate the efficiency of state-of-the-art methods, we introduce a neighborhood similarity-based graph compression method. We build two models, NECL and NECL-RF, where NECL provides efficiency with learning on the small compressed graph, and NECL-RF provides effectiveness with incorporating global information into embedding with the compressed graph. Experimental results on various real-world graphs show the effectiveness and efficiency of our methods on challenging multi-label and multi-class classification tasks.

In the second project, we introduce a novel motif-based graph pooling method, MPool, that captures the higher-order graph structures for graph-level representation. Our proposed method includes hierarchical graph pooling models for both selection-based and clustering-based methods. Additionally, we combine these methods to develop a hybrid model. Our experiments demonstrate that our proposed methods outperform the baseline models on a majority of the datasets.

In our third project, we develop a novel graph contrasting learning model for the graph classification task. In this model, we use K-core and k-truss based graph compression for the view generation for contrastive learning. Graph compression technique captures the global and local structures of the graph. Our experiments demonstrate that it achieves the state-of-the-art performance on the graph classification task, where we compare our model with graph pooling and graph contrastive learning models.

In our last work, we propose a Dynamic Graph Contrastive Learning model DyGCL for event prediction. There are two view encoders in our model one is a local view encoder, and another one is a global view encoder. The local view encoder learns dynamic graph representation by capturing the temporal local structure of input graphs, and the global view encoder learns dynamic graph representation that encodes the temporal global structure of the input graph. In the experiment, we show that our model outperforms existing Dynamic GNN methods on event prediction tasks on various real-world datasets.

These projects collectively highlight the importance of integrating local and global structures in graph representation learning for both static and dynamic graphs across various domains. Our research enhances the capabilities of graph representation learning, demonstrating superior performance over existing methods in diverse applications. The advancements presented in this dissertation lay a strong foundation for future exploration, driving the development of more robust, interpretable, and practical models in graph-based learning.

REFERENCES

- [1] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network Representation Learning: A Survey. *IEEE Transactions on Big Data*, PP(c):1, 2017.
- [2] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.
- [3] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.
- [4] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.
- [5] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 2018.
- [6] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the SIGKDD'14*, pages 701–710. ACM, 2014.
- [7] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the ICLR*, 2017.
- [8] Will Hamilton, Zhitaoying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.
- [9] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning*

Representations, 2018.

- [10] Khaled Mohammed Saifuddin, Bri Bumgardnerr, Farhan Tanvir, and Esra Akbas. Hygmn: Drug-drug interaction prediction via hypergraph neural network. *arXiv preprint arXiv:2206.12747*, 2022.
- [11] Tanvir Hossain, Khaled Mohammed Saifuddin, Muhammad Ifte Khairul Islam, Farhan Tanvir, and Esra Akbas. Tackling oversmoothing in gnn via graph sparsification. In *ECML/PKDD (8)*, 2024.
- [12] Tanvir Hossain, Esra Akbas, and Muhammad Ifte Khairul Islam. End: Enhanced dedensification for graph compressing and embedding. In *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 674–681. IEEE, 2022.
- [13] Hamdi Selim Akin, Mehmet Emin Aktas, Muhammed Ifte Islam, Tanvir Hossain, and Esra Akbas. Exploring similarity-based graph compression for efficient network analysis and embedding. In *2024 33rd International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6. IEEE, 2024.
- [14] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.
- [15] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, volume 97, 2019.
- [16] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *International Conference on Machine Learning*, pages 874–883. PMLR, 2020.

- [17] John Boaz Lee, Ryan A Rossi, Xiangnan Kong, Sungchul Kim, Eunye Koh, and Anup Rao. Graph convolutional networks with motif-based attention. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 499–508, 2019.
- [18] Austin R Benson, David F Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.
- [19] Austin R Benson, David F Gleich, and Desmond J Higham. Higher-order network analysis takes off, fueled by classical ideas and new data. *arXiv preprint arXiv:2103.05031*, 2021.
- [20] Mehmet Emin Aktas, Thu Nguyen, Rakin Riza, Muhammad Ifte Islam, and Esra Akbas. Hypergraph classification via persistent homology. *arXiv preprint arXiv:2306.11484*, 2023.
- [21] Mehmet Emin Aktas, Thu Nguyen, Sidra Jawaaid, Rakin Riza, and Esra Akbas. Identifying critical higher-order interactions in complex networks. *Scientific reports*, 11(1):1–11, 2021.
- [22] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594), 2002.
- [23] Tijana Milenković, Jason Lai, and Nataša Pržulj. Graphcrunch: a tool for large network analyses. *BMC bioinformatics*, 9(1):70, 2008.
- [24] Rasha Elhesha and Tamer Kahveci. Identification of large disjoint motifs in biological

- networks. *BMC bioinformatics*, 17(1):1–18, 2016.
- [25] Pei-Zhen Li, Ling Huang, Chang-Dong Wang, and Jian-Huang Lai. Edmot: An edge enhancement approach for motif-aware community detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.
- [26] Namkyeong Lee, Junseok Lee, and Chanyoung Park. Augmentation-free self-supervised learning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7372–7380, 2022.
- [27] Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *International conference on machine learning*, pages 4116–4126. PMLR, 2020.
- [28] Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *International Conference on Learning Representations*, 2020.
- [29] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.
- [30] Zhihao Wen and Yuan Fang. Trend: Temporal event and node dynamics for graph representation learning. In *Proceedings of the Web Conference 2022*, 2022.
- [31] Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD conference on knowledge*

- discovery and data mining*, pages 2358–2366, 2022.
- [32] Andrea Cini, Ivan Marisca, Filippo Maria Bianchi, and Cesare Alippi. Scalable spatiotemporal graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 7218–7226, 2023.
- [33] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th international conference on web search and data mining*, pages 519–527, 2020.
- [34] Songgaojun Deng, Huzefa Rangwala, and Yue Ning. Learning dynamic context graphs for predicting social events. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1007–1016, 2019.
- [35] Mert Kosan, Arlei Silva, Sourav Medya, Brian Uzzi, and Ambuj Singh. Event detection on dynamic graphs. *arXiv preprint arXiv:2110.12148*, 2021.
- [36] Haochen Chen, Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. A tutorial on network embeddings. *arXiv preprint arXiv:1808.02590*, 2018.
- [37] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.
- [38] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.
- [39] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Graphgan: Graph representation learning with generative

- adversarial nets. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [40] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the WSDM'18*, pages 459–467, 2018.
- [41] Jiankai Sun, Bortik Bandyopadhyay, Armin Bashizade, Jiongqian Liang, P Sadayappan, and Srinivasan Parthasarathy. Atp: Directed graph embedding with asymmetric transitivity preservation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 265–272, 2019.
- [42] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the WWW'15*, pages 1067–1077, 2015.
- [43] Shaosheng Cao, Wei Lu, and Qionгкаi Xu. Deep neural networks for learning graph representations. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- [44] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the NIPS'18*, pages 4805–4815, 2018.
- [45] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the WWW'18*, pages 539–548, 2018.
- [46] Bishal Thapaliya, Esra Akbas, Jiayu Chen, Ram Sapkota, Bhaskar Ray, Pranav Suresh, Vince D Calhoun, and Jingyu Liu. Brain networks and intelligence: A graph neural

- network based approach to resting state fmri data. *Medical Image Analysis*, 101:103433, 2025.
- [47] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the KDD'16*, pages 1105–1114, New York, NY, USA, 2016. ACM.
- [48] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. Harp: Hierarchical representation learning for networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [49] Esra Akbas and Peixiang Zhao. Graph clustering based on attribute-aware graph embedding. In *From Security to Community Detection in Social Networking Platforms*, pages 109–131. Springer International Publishing, 2019.
- [50] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the ICML'16*, pages 2014–2023, 2016.
- [51] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the CIKM '15*, pages 891–900, 2015.
- [52] Esra Akbas and Peixiang Zhao. Attributed graph clustering: An attribute-aware graph embedding approach. In *Proceedings of the ASONAM'17*, pages 305–308, 2017.
- [53] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of Workshop at ICLR, 2013.*, 2013.

- [54] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
- [55] Maximilien Danisch Ayan Kumar Bhowmick, Koushik Meneni. Louvainne: Hierarchical louvain method for high quality and scalable network embedding. In *In WSDM*, 2020.
- [56] Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. Mile: A multi-level framework for scalable graph embedding. *arXiv preprint arXiv:1802.09612*, 2018.
- [57] Yongyu Wang Zhiru Zhang Zhuo Feng Chenhui Deng, Zhiqiang Zhao. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. In *In ICLR 2020*, 2020.
- [58] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.
- [59] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.
- [60] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.
- [61] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-second AAAI conference on*

- artificial intelligence*, 2018.
- [62] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *NeurIPS*, 31, 2018.
- [63] Ekagra Ranjan, Soumya Sanyal, and Partha Talukdar. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5470–5477, 2020.
- [64] Davide Bacciu, Alessio Conte, Roberto Grossi, Francesco Landolfi, and Andrea Marino. K-plex cover pooling for graph neural networks. *Data Mining and Knowledge Discovery*, 35(5), 2021.
- [65] Xing Gao, Wenrui Dai, Chenglin Li, Hongkai Xiong, and Pascal Frossard. ipool—information-based pooling in hierarchical graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.
- [66] Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954*, 2019.
- [67] Qingyun Sun, Jianxin Li, Hao Peng, Jia Wu, Yuanxing Ning, Philip S Yu, and Lifang He. Sugar: Subgraph neural network with reinforcement pooling and self-supervised mutual information mechanism. In *Proceedings of the Web Conference 2021*, pages 2081–2091, 2021.
- [68] Hannes Mueller and Christopher Rauh. Reading between the lines: Prediction of

- political violence using newspaper text. *American Political Science Review*, 112(2):358–375, 2018.
- [69] Andranik Tumasjan, Timm Sprenger, Philipp Sandner, and Isabell Welp. Predicting elections with twitter: What 140 characters reveal about political sentiment. In *Proceedings of the international AAAI conference on web and social media*, volume 4, pages 178–185, 2010.
- [70] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of computational science*, 2(1):1–8, 2011.
- [71] Courtney D Corley, Laura L Pullum, David M Hartley, Corey Benedum, Christine Noonan, Peter M Rabinowitz, and Mary J Lancaster. Disease prediction models and operational readiness. *PloS one*, 9(3):e91989, 2014.
- [72] Khaled Mohammed Saifuddin, Muhammad Ifte Khairul Islam, and Esra Akbas. Drug abuse detection in twitter-sphere: Graph-based approach. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 4136–4145. IEEE, 2021.
- [73] Farhan Tanvir, Muhammad Ifte Khairul Islam, and Esra Akbas. Predicting drug-drug interactions using meta-path based similarities. In *2021 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 1–8. IEEE, 2021.
- [74] Farhan Tanvir, Khaled Mohammed Saifuddin, Muhammad Ifte Khairul Islam, and Esra Akbas. Predicting drug-drug interactions using heterogeneous graph attention networks. In *Proceedings of the 14th ACM International Conference on Bioinformatics*,

- Computational Biology, and Health Informatics*, pages 1–6, 2023.
- [75] Bishal Thapaliya, Esra Akbas, Ram Sapkota, Bhaskar Ray, Vince Calhoun, and Jingyu Liu. Self-clustering graph transformer approach to model resting state functional brain activity. In *2025 IEEE 22nd International Symposium on Biomedical Imaging (ISBI)*, pages 1–5, 2025.
- [76] Zeng Xiao-Jun. Twitter mood predicts the stock market. *Journal of Computational Science*. 1–8., 2011.
- [77] Yue Ning, Sathappan Muthiah, Huzefa Rangwala, and Naren Ramakrishnan. Modeling precursors for event forecasting via nested multi-instance learning. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1095–1104, 2016.
- [78] Xiaofeng Wang, Matthew S Gerber, and Donald E Brown. Automatic crime prediction using events extracted from twitter posts. *SBP*, 12:231–238, 2012.
- [79] Jing Ma, Wei Gao, Zhongyu Wei, Yueming Lu, and Kam-Fai Wong. Detect rumors using time series of social context information on microblogging websites. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 1751–1754, 2015.
- [80] Jing Ma, Wei Gao, Prasenjit Mitra, Sejeong Kwon, Bernard J Jansen, Kam-Fai Wong, and Meeyoung Cha. Detecting rumors from microblogs with recurrent neural networks. 2016.
- [81] Bishal Thapaliya, Robyn Miller, Jiayu Chen, Yu Ping Wang, Esra Akbas, Ram Sap-

- kota, Bhaskar Ray, Pranav Suresh, Santosh Ghimire, Vince D Calhoun, et al. Dsam: A deep learning framework for analyzing temporal and spatial dynamics in brain networks. *Medical Image Analysis*, 101:103462, 2025.
- [82] Yang Liu and Yi-Fang Wu. Early detection of fake news on social media through propagation path classification with recurrent and convolutional networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.
- [83] Rui Xia, Kaizhou Xuan, and Jianfei Yu. A state-independent and time-evolving network for early rumor detection in social media. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pages 9042–9051, 2020.
- [84] Muhammad Ifta Islam, Farhan Tanvir, Ginger Johnson, Esra Akbas, and Mehmet Emin Aktas. Proximity-based compression for network embedding. *Frontiers in big Data*, 3:608043, 2021.
- [85] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.
- [86] Muhammad Ifta Khairul Islam, Max Khanov, and Esra Akbas. Mpool: motif-based graph pooling. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 105–117. Springer, 2023.
- [87] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.
- [88] William G Underwood, Andrew Elliott, and Mihai Cucuringu. Motif-based spectral

- clustering of weighted directed networks. *Applied Network Science*, 5(1):1–41, 2020.
- [89] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.
- [90] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015.
- [91] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 2019.
- [92] Jialu Chen and Gang Kou. Attribute and structure preserving graph contrastive learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 7024–7032, 2023.
- [93] Uno Fang, Jianxin Li, Naveed Akhtar, Man Li, and Yan Jia. Gomic: Multi-view image clustering via self-supervised contrastive heterogeneous graph co-learning. *World Wide Web*, 26(4):1667–1683, 2023.
- [94] Jiaqi Zeng and Pengtao Xie. Contrastive self-supervised learning for graph classification. In *Proceedings of the AAAI conference on Artificial Intelligence*, volume 35, pages 10824–10832, 2021.
- [95] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*, 2020.

- [96] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33:5812–5823, 2020.
- [97] Shichang Zhang, Ziniu Hu, Arjun Subramonian, and Yizhou Sun. Motif-driven contrastive learning of graph representations. *arXiv preprint arXiv:2012.12533*, 2020.
- [98] Muhammed Ifta Khairul Islam, Khaled Mohammed Saifuddin, Tanvir Hossain, and Esra Akbas. Dygcl: Dynamic graph contrastive learning for event prediction. In *2024 IEEE International Conference on Big Data (BigData)*, pages 559–568. IEEE, 2024.
- [99] Elizabeth Boschee, Jennifer Lautenschlager, Sean O’Brien, Steve Shellman, James Starz, and Michael Ward. Icews coded event data, 2015.
- [100] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858, 2020.
- [101] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5363–5370, 2020.
- [102] Mengzhu Sun, Xi Zhang, Jiaqi Zheng, and Guixiang Ma. Ddgc: Dual dynamic graph convolutional networks for rumor detection on social media. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 4611–4619, 2022.