LEARNING EFFECTIVE EMBEDDINGS FOR DYNAMIC GRAPHS AND QUANTIFYING GRAPH EMBEDDING INTERPRETABILITY

SHIMA KHOSHRAFTAR

A DISSERTATION SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

JANUARY 2023

©SHIMA KHOSHRAFTAR, 2023

Abstract

Graph representation learning has been a very active research area in recent years. The goal of graph representation learning is to generate representation vectors that accurately capture the structure and features of large graphs. This is especially important because the quality of the graph representation vectors will affect the performance of these vectors in downstream tasks such as node classification and link prediction. Many techniques have been proposed for generating effective graph representation vectors. These methods can be applied to both static and dynamic graphs. A static graph is a single fixed graph, while a dynamic graph evolves over time, and its nodes and edges can be added or deleted from the graph. We surveyed the graph embedding methods for both static and dynamic graphs. The majority of the existing graph embedding methods are developed for static graphs. Therefore, since most real-world graphs are dynamic, developing novel graph embedding methods suitable for evolving graphs is essential.

This dissertation proposes three dynamic graph embedding models. In previous dynamic methods, the inputs were mainly adjacency matrices of graphs which are not memory efficient and may not capture the neighbourhood structure in graphs effectively. Therefore, we developed Dynnode2vec based on random walks using the static model Node2vec. Dynnode2vec generates node embeddings in each snapshot by initializing the current model with previous embedding vectors and training the model using a set of random walks obtained for nodes in the snapshot. Our second model, LSTM-Node2vec, is also based on random walks. This method leverages the LSTM model to capture the long-range dependencies between nodes in combination with Node2vec to generate node embeddings. Finally, inspired by the importance of substructures in the graphs, our third model TGR-Clique generates node embeddings by considering the effects of neighbours of a node in the maximal cliques containing the node. Experiments on real-world datasets demonstrate the effectiveness of our proposed methods in comparison to the state-of-the-art models. In addition, motivated by the lack of proper measures for quantifying and comparing

graph embeddings interpretability, we proposed two interpretability measures for graph embeddings using the centrality properties of graphs.

Acknowledgments

My Ph.D. was made possible thanks to many wonderful people. First, I would like to express my sincere appreciation to my advisor, Prof. Aijun An, for her incredible guidance, support, encouragement and understanding throughout my Ph.D. studies. Aijun's curiosity, immense knowledge and passion for science have continually inspired me. I am very grateful for her confidence in me and for helping me develop my skills as a researcher.

I would like to thank my committee members, Prof. Manos Papagelis and Prof. Michael Brown, for their guidance, time and valuable comments. I'd like to acknowledge past and present members of the Data Mining lab, who have enriched my graduate experience as labmates, mentors, collaborators and friends. Especially, I am grateful to Dr. Sedigheh Mahdavi for our close collaboration during several parts of my thesis. It was a pleasure to work with her and learn from her during my Ph.D. I also would like to thank Dr. Nastaran Babanejad for our collaboration in my thesis's last part and helpful feedback and advice.

I will always be indebted to my dearest parents, who showered me with love and support in every aspect of my life. I am grateful for all the sacrifices they have made to enable me to follow my own path. I also thank my two brothers and their families for being there whenever I needed them. Finally, I wish to thank my husband and the love of my life for his love and motivation and for patiently standing by my side. His presence in my life alone brings out the best in me.

Table of Contents

Abstract ii

Acknowledgments iv

Table of Contents v

List of Figures ix

List of Tables xii

List of Abbreviations xv

1 Introduction 1

1.1 Motivation 1

1.2 Contributions 3

1.3 Publications and the Statement of Co-authorships 5

1.4 Dissertation Organization 6

2 Literature Review 7

2.1 Graphs 7

2.1.1 Graph Embedding 8

2.1.2 Graph Embedding Applications 10

2.2 Traditional Graph Embedding 12

2.2.1 Traditional Static Graph Embedding 12

2.2.2 Traditional Dynamic Graph Embedding 18

2.3 GNN based graph embedding 24

2.3.1 Introduction to GNNs . . . 25
2.3.2 Static Graph Neural Nets . . . 28
2.3.3 Spatial-Temporal Graph Neural Net (STGNN) . . . 37
2.3.4 Dynamic Graph Neural Net (DGNN) . . . 38
2.3.5 GNN-based method's real-world applications . . . 41
2.3.6 The limitation of GNNs and the proposed solutions . . . 41
2.4 Interpretability of node embeddings . . . 52

3 Dynnode2vec: Scalable Dynamic Network Embedding 54
3.1 Introduction . . . 54
3.2 *Dynnode2vec*: Scalable dynamic network embedding . . . 55
3.2.1 Description of *dynnode2vec* steps . . . 55
3.3 Experiments . . . 57
3.3.1 Datasets . . . 57
3.3.2 Baselines . . . 58
3.3.3 Link Prediction . . . 58
3.3.4 Node Classification . . . 59
3.3.5 Anomaly Detection . . . 59
3.3.6 Effects of evolving walk generation . . . 60
3.4 Summary . . . 61

4 Dynamic Graph Embedding via LSTM History Tracking 62
4.1 Introduction . . . 62
4.2 Problem Statement . . . 64
4.3 Dynamic Network Embedding Method . . . 64
4.3.1 Overview of LSTM-Node2vec . . . 64
4.3.2 Temporal neighbor walk generation. . . . 65
4.3.3 LSTM Autoencoder . . . 66
4.3.4 Node2vec . . . 68
4.4 Experiments . . . 69
4.4.1 Baselines . . . 69

4.4.2 Experiment settings . . . 70
4.4.3 Anomaly Detection . . . 70
4.4.4 Node Classification . . . 72
4.4.5 Link Prediction . . . 73
4.4.6 Effect of length of history parameter L . . . 75
4.4.7 Effects of changes in model structure . . . 75
4.4.8 Time Analysis . . . 76
4.5 Summary . . . 77

5 Temporal Graph Representation Learning via Maximal Cliques 78
5.1 Introduction . . . 78
5.2 Preliminaries . . . 80
5.3 Problem definition . . . 82
5.4 Proposed Method . . . 83
5.4.1 Method overview . . . 83
5.4.2 Temporal walk generation . . . 84
5.4.3 Model . . . 85
5.4.4 Training . . . 86
5.4.5 Inference . . . 87
5.4.6 Justification of using maximal cliques . . . 89
5.5 Experiments . . . 90
5.5.1 Baselines . . . 90
5.5.2 Settings . . . 91
5.5.3 Link prediction . . . 92
5.5.4 Node classification . . . 93
5.5.5 Ablation study . . . 94
5.5.6 Parameter sensitivity . . . 95
5.5.7 Time analysis . . . 96
5.6 Comparison with our other methods proposed in Chapters 3 and 4 . . . 98
5.7 Summary . . . 100

6 Interpretability measures for Graph Embeddings **102**

6.1 Introduction . . . 102

6.2 Preliminaries . . . 104

6.3 Problem definition . . . 105

6.4 Method . . . 105

6.4.1 Approach overview . . . 105

6.4.2 The definition of interpretability scores . . . 106

6.4.3 Example . . . 108

6.5 Experiments . . . 110

6.5.1 Datasets . . . 110

6.5.2 Node embedding methods . . . 111

6.5.3 Quantitative measuring of interpretability of graph embedding methods . . . 112

6.5.4 Validation . . . 112

6.5.5 Interpreting embedding dimensions with visualization . . . 114

6.5.6 Downstream tasks . . . 115

6.5.7 Parameter sensitivity . . . 117

6.6 Summary . . . 118

7 Conclusions and Future Directions **119**

7.1 Summary of Contributions . . . 119

7.2 Future Directions . . . 121

Bibliography **123**

List of Figures

2.1 Facebook as an example of social networks 8

2.2 Example of finance networks 9

2.3 The graph on the left hand side consists of 6 nodes $\{a, b, c, d, e, i\}$ and 8 edges. Graph embedding methods map each node of the graph into an embedding vector with dimension d. For the demonstration purpose, the node a is embedded into an embedding vector z_a of dimension 4 with given values. 10

2.4 Categories of Graph Representation Learning Methods. 12

2.5 A general supervised framework for training Graph neural net layers. Two GNN layers are applied on an input graph to compute the node representation vectors for its nodes. The colors on arrows show neighbors of a target node that are aggregated to generate the target node representation. x_a is the feature vector of node a and h_a^1 and h_a^2 are the representation vectors generated for the node a after applying the first and second GNN layers. The generated embeddings are used in a node classification task. y'_a is the predicted label for the node a. . 26

3.1 Anomaly detection results 60

4.1 LSTM training for word representation. s_j is the jth word in a sentence which is represented by a one-hot encoding vector. W_i is the weights of input layer in LSTM. 66

4.2 The process of temporal walk generation. a, b, c are three time points of a temporal network. Given $L = 3$ the temporal walks for node 1 is generated by sampling from its neighborhood nodes. 66

4.3 Framework for LSTM-Node2vec for creating embedding of G_t using both temporal and static sequences 69

4.4 A temporal network at two time steps. (a) the graph at time $t-1$. (b) the graph at time t with an star shaped anomaly, one previously normal node starts attacking multiple nodes shown in red. 71

4.5 Analysis of effect of parameter L on link prediction for Radoslaw and St-Ov datasets 74

4.6 Analysis of effect of parameter L on node classification for Acm and Dblp datasets 74

4.7 Time complexity Analysis for 4 time steps in 3 datasets 76

5.1 A graph with five nodes {1, 2, 3, 4, 5} with a subgraph depicted in red: a) the red subgraph is not a clique as there is no edge between nodes 1 and 3. b) the red subgraph is a clique but not a maximal one because node 4 can be added to the clique. c) the red subgraph is a maximal clique as there is an edge between every pair of nodes and it does not have a supergraph that is also a clique. 81

5.2 The general framework of TGR-Clique for the node classification task. The input is the temporal graph. We generate the embedding for the grey node. First, a set of temporal walks are generated for the node from the maximal cliques containing the node. For the illustration, the number and length of walks are 3 and 2. Then, these walks are input to the BiLSTM, attention and mean layers to generate the node embedding (block with the purple dot). x_v, x_e, t_e are the node, edge and time features associated with each node in the walks. The framework for link prediction is similar except that embeddings from two nodes are concatenated to input to the FC layer for link prediction. 83

5.3 Examples of temporal random walks for node a, at time 7 on the graph's maximal cliques. For instance, a, d, c is a temporal random walk of length 2 for node a. In this walk, node d is a neighbor of node a and node c is a neighbor of node d. The nodes in the walk belong to the maximal clique $\{a, b, c, d\}$ which contains node a. The edges in the walk are sampled in an anti-chronological order such that $7 > 6 > 2$. No other edge can be added to the walk after node c. . . 84

5.4 An MRF over random variables $X = \{x_1, x_2, x_3, x_4, x_5\}$ with three maximal cliques MC_1, MC_2 and MC_3. 89

5.5 Parameter sensitivity analysis on the transductive link prediction task. 97

5.6 Training time on the SMS dataset over one epoch and different numbers of edges. 98

6.1 An example of BCI calculation for one embedding dimension with respect to one category. I. A sample graph with nodes $\{a, b, c, d, e, f\}$. Nodes $\{a, b, c, d\}$ belong to category 1 (C_1) and $\{e, f\}$ belong to category 2 (C_2). II. Node embedding matrix of the graph where D = 3. III. Computation of BCI score for embedding dimension 1 with respect to node category 1. 109

6.2 Dimension and Category decomposition of BCI scores on the Wisconsin dataset using Node2vec embedding vectors of 128 dimensions: Left side BCI scores of embedding dimensions 96, 25 and 79 over all categories. Right side BCI values over all dimensions for the Project, Faculty and Course category. 114

6.3 Visualization of the embedding of Wisconsin dataset using t-SNE for top two most populated classes. 116

List of Tables

2.1 Graph embedding methods in both traditional and GNN-based categories. Trad and GNN stand for Traditional and GNN-based graph embedding. 13

2.2 Comparison of traditional and GNN-based graph representation learning. 25

2.3 Some of the real-world applications of graph neural nets **deployed in production** 41

2.4 A summary of major solutions proposed to increase the expressive power of GNNs 43

2.5 A summary of major solutions proposed to alleviate oversmoothing of GNNs. Z is the node prediction label, H is the node representation, K_l is the convolution of the l-th layer, s is a projection vector, G_o is the discrete gradient operator on the graph, $\hat{A}_{drop}$ is the symmetric normalized adjacency matrix with certain number of edges dropped, INFLATION(.,e) = Normal(Power(Softmax(.),e)), f_l is the number of features at layer l, S^l is the clustering assignment matrix, C is number of groups, μ_i, σ_i are the mean and standard deviation of group i. $\gamma_i, \beta_i, \lambda$ are hyper parameters. TPSD is a total pairwise squared distance measure, X_{drop} is the perturbed feature matrix, p is an step size, G_v is an extracted subgraph for a node v. Φ_l is a learnable parameter, $\tilde{L}$ is symmetric normalized Laplacian matrix. psi is a learnable function, X is the feature matrix. T is time step. 45

2.6 Time and memory complexities of GCN and some of the proposed scalable GNN models. L is the number of layers, F is the hidden dimension of the model, n is the number of nodes, m is the number of neighbors, b is the batch size, d is the average degree of the graph, s_n is the number of sampled nodes per node,s_l is the number of sampled nodes per layer, δ is the ratio of blocked nodes, $\tilde{s}_n = s_n \times (1 - \delta)$, m_g is the number of remaining edges, m_θ is the number of remaining connections in the model, k is number of codewords 46

2.7 A summary of major solutions proposed to make GNNs suitable for both hemophily and heterophily 49

2.8 The formulas for the major solutions proposed to help GNNs capture substructures. i is a neighborhood , r is a relationship, $\tau(z_v, z_u)$ is a function that defines a relationship from node v to node u in a latent space, x_v^V are the combined structural features of node v, $e_{u,v}$ are the edge (u, v) features, h_p is the embedding of a simplex p, $m_B(p), m_C(p), m_\downarrow(p), m_\uparrow(p)$ are the aggregation of messages from the boundary, co-boundary, lower and upper adjacent simplices of the simplex p, R_v is a row in a node-topic matrix representing the probabilities of a node v in a graph belonging to the graph's structural topics, $S_{n_{v,t}}^{k-BFS}$ is the set of permutation of subset of nodes in egonet of node v of depth t compatible with k-truncated breadth first search, α_π is a learnable normalization factor for π, f can be an MLP layer, $B_{v,t}^{[ego]}$ is the tensor representation of the egonet of node v, ϕ is an anonymous walk,$R(\phi)$ is the concatenation of the attributes of ϕ, $Z = [R(\phi_i)]_i$, Φ_l is the set of anonymous walks of length l, $h_{x,c}$ is the representation of a subgraph c, γ is a learnable similarity measure, A_x is a subgraph at channel x, a_x is the representation of A_x. 50

3.1 Link prediction results using four operators a) Weighted-L1 b) Weighted-L2 c) Hadamard d) Average 59

3.2 Node classification results 60

3.3 Running time comparison 61

3.4 Comparison of dynnode2vec vs dynnode2vec-all 61

4.1 AUC results of anomaly detection 71

4.2 Anomaly detection AUC scores for three graphs to analyze the effect of previously seen anomalies 72

4.3 Macro-f1 and Micro-f1 scores for node classification 73
4.4 AUC scores for link prediction 75
4.5 Difference in performance by adding one more layer to the encoder . . 76

5.1 An example showing potential steps for generating maximal clique $\{5, 4, 2, 3\}$ in the graph in Figure 5.1 using the original recursive method in [1]. 82
5.2 The datasets statistics. #MC is the number of maximal cliques in the training set, #F is the number of node and edge features. 91
5.3 The transductive link prediction results. 93
5.4 The inductive link prediction results. 94
5.5 AUC scores for the node classification. 95
5.6 AUC scores for the ablation study. NC and LP are short for node classification and link prediction respectively. 95
5.7 Comparison of our 3 proposed dynamic graph embedding methods. Dyn is short for Dynamic, Cat is Category, Trad is Traditional, Continu is Continuous and Feat is Feature. 100
5.8 Link prediction results in terms of AUC. 100

6.1 Betweenness centrality importance and Closeness centrality importance scores 111
6.2 Macro-F1 and Micro-F1 scores for node classification task using node degree labels 113
6.3 Macro-F1 and Micro-F1 scores for node classification task using betweenness centrality node labels 113
6.4 Macro-F1 and Micro-F1 scores for node classification task using closeness centrality node labels 113
6.5 Downstream link prediction task auc scores 117
6.6 Downstream node classification Macro-F1 and Micro-F1 scores 117
6.7 CCI score using different λ values on the Wisconsin dataset 117

List of Abbreviations

AP	Average Precision
AUC	Area Under (ROC) Curve
BC	Betweenness Centrality
BCI	Betweenness Centrality Importance
BFS	Breadth First Search
BiLSTM	Bidirectional Long Short Term Memory
CC	Closeness Centrality
CCI	Closeness Centrality Importance
CNN	Convolutional Neural Network
ConvGNN	Convolutional Graph Neural Network
DFS	Depth First Search
ENC	Encoder
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
GRU	Gated Recurrent Unit
KL-divergence	Kullback–Leibler divergence
LSTM	Long Short Term Memory
MLP	Multilayer Perceptron

MRF	Markov Random Field
NLP	Natural Language Processing
RecGNN	Recurrent Graph Neural Network
RNN	Recurrent Neural Network
TGR-Clique	Temporal Graph Representation via maximal Cliques
WL test	Weisfeiler-Lehman test

Chapter 1

Introduction

1.1 Motivation

Graphs are powerful data structures to represent networks that contain entities and relationships between entities. There are very large networks in different domains including social networks, financial transactions and biological networks. For instance, in social networks people are the nodes and their friendships constitute the edges. In financial transactions, the nodes and edges could be people and their money transactions. One of the strengths of a graph data structure is its generality, meaning that the same structure can be used to represent different networks. In addition, graphs have strong foundations in mathematics, which could be leveraged for analyzing and learning from complex networks.

In order to use graphs in different downstream applications, it is important to represent them effectively. Graph can be simply represented using the adjacency matrix which is a square matrix whose elements indicate whether pairs of vertices are adjacent or not in the graph, or using the extracted features of the graph. However, the dimensionality of the adjacency matrix is often very high for big graphs, and the feature extraction based methods are time consuming and may not represent all the necessary information in the graphs. Recently, the abundance of data and computation resources paves the way for more flexible graph representation methods. Specifically, graph embedding methods have been very successful in graph representation. These methods project the graph elements (such as nodes, edges and subgraphs) to a lower dimensional space and preserve the properties of graphs.

Graph embedding methods can be categorized into traditional graph embedding and graph neural net (GNN) based graph embedding methods. Traditional graph embedding methods capture the information in a graph by applying different tech-

niques including random walks, factorization methods and non-GNN based deep learning. These methods can be applied to both static and dynamic graphs. A static graph is a single fixed graph, while a dynamic graph evolves over time and its nodes and edges can be added or deleted from the graph. For example, a molecule can be represented as a static graph, while a social network can be represented by a dynamic graph. Graph neural nets are another category of graph embedding methods that have been proposed recently. In GNNs, node embeddings are obtained by aggregating the embeddings of the node's neighbors. Early works on GNNs were based on recurrent neural networks. However, later convolutional graph neural nets were developed that are based on the convolution operation. In addition, there are spatial-temporal GNNs and dynamic GNNs that leverage the strengths of GNNs in evolving networks. We reviewed the advancements in graph embedding methods in both traditional and GNN-based categories for both static and dynamic settings in a survey on graph embedding methods [2].

The majority of the existing graph embedding methods are developed for static graphs as modeling the characteristics of dynamic graphs is more challenging than modeling static graphs. Some of the traditional dynamic graph embedding models are DynGEM [3], dyngraph2vecAE [4] and CTDNE [5]. Considering the evolving nature of most of the real-world graphs and the need to develop effective methods to model these graphs, we developed two traditional dynamic graph embedding methods called Dynnode2vec [6] and LSTM-Node2vec [7] and one GNN-based dynamic method named TGR-Cliques [8].

Our first model Dynnode2vec is the dynamic version of the node2vec method for static graphs. It is a fast and scalable method based on evolving random walks. Compared to previous embedding methods that mainly take the adjacency matrix of a graph as input, dynnode2vec utilizes random walks because they are more memory efficient and better capture the neighbourhood of nodes. Furthermore, this method combines the history of interactions of nodes into their current states using the weights from the previous timestamps models. The second model LSTM-Node2vec is also based on random walks. Inspired by the success of LSTM in natural language processing, LSTM-Node2vec takes advantage of the LSTM model to capture the long-term dependencies between nodes. Then, it combines this information with the node2vec model to generate the graph embeddings.

Our third model is TGR-Clique which is a GNN-based temporal graph embedding method. This model takes into account the maximal cliques containing a node in generating the node embedding. The graph substructures such as maximal cliques and triads can reveal important information about the groups of nodes that com-

monly interact and potentially have the most influence on each other. However, a few dynamic methods consider graph substructures. Examples of these methods are MTSN [9] and CAW-N [10]. We develop TGR-Clique that generates temporal node embedding by aggregating the embeddings of nodes that co-occur in maximal cliques with the node.

In addition to designing three novel dynamic graph embedding methods, we proposed two interpretability measures for graph embeddings [11]. Despite the success of graph embedding methods in different applications, not much attention has been given to interpreting the graph embeddings. In traditional feature engineering methods, each dimension represents a particular piece of information, such as the degree or importance of a node, which is meaningful and thus potentially useful in explaining the decision made in the application. Graph embedding dimensions, however, are usually not explainable. Inspired by advances in interpreting word embeddings [12, 13], we introduced two interpretability measures for graph embeddings using the network centrality properties.

1.2 Contributions

In this dissertation, we first review the advances in graph embedding methods. Then, two traditional dynamic graph embedding methods and one GNN-based temporal graph emebdding method are introduced. Furthermore, we proposed two interpretability measures for graph embeddings. A summary of our contributions are as follow:

- **Dynnode2vec [6]**: We proposed *dynnode2vec*, a scalable dynamic network embedding for large evolving networks. In order to handle dynamic networks, *dynnode2vec* modifies the well-known static embedding method, *node2vec* by employing the previous learned embedding vectors as initials weights for the skip-gram model. This is motivated by dynamic language models, especially dynamic Skip-gram models [14, 15]. In addition, we utilize evolving random walks for updating the trained skip-gram from previous timestamp. The evolving random walks are only generated for nodes that have changed in consecutive times. As random walk generation is the time consuming part of graph embeddings, we are able to significantly reduce the running time. We evaluate the performance of our proposed method in variety of tasks including link prediction, node classification and anomaly detection on large real-world graphs.

- **LSTM-Node2vec** [7]: Inspired by the effectiveness of LSTM in preserving long term dependencies between elements in a sequence in natural language processing [16, 17, 18], we propose LSTM-Node2vec for computing node embeddings in dynamic graphs. Our key contribution is using an autoencoder LSTM for keeping the history of nodes and training it with a special kind of temporal random walks that capture the evolving patterns in the structure of the graphs. In our method, the embeddings obtained from history are used as initial weights for a node2vec model. Afterwards, node2vec considers the local information from the current graph and produces an embedding that is the combination of both temporal and static information for the nodes of the graph. In addition, for aligning the node embeddings over time, the weights of the model at previous time points is passed along to the model for the next graphs. In our experiments, we evaluate the performance of LSTM-Node2vec in anomaly detection, link prediction and node classification on datasets from various domains.

- **TGR-Clique** [8]: We propose a novel temporal graph representation learning method based on maximal cliques, called **TGR-Clique** short for **T**emporal **G**raph **R**epresentation learning via maximal **Clique**s. Maximal cliques are important graph substructures that represent groups of densely connected nodes. These subgraphs have been successfully used in community detection in social networks [19, 20], clustering in wireless networks [21] and protein-protein interaction detection in biological networks [22, 23]. TGR-Clique generates the node embeddings using maximal cliques. To capture the evolution of temporal graphs, we generate temporal random walks on the maximal cliques. The embedding of a node is obtained by aggregation of the temporal random walks starting from the node. We showed that the embeddings generated in TGR-Clique can lead to better predictive performance in downstream tasks and that our method is faster than CAW-N and some other state-of-the-art methods. In addition, our method can be used in both link prediction and node classification tasks since we produce both node and link embeddings.

- **Interpretability metrics for graph embeddingss** [11]: We quantify and compare the interpretability of different graph embedding methods effectively by adapting the techniques originally designed for quantifying interpretability of word embeddings [12, 24, 25, 13]. We quantify the interpretability of a node embedding method based on the extent to which its embedding dimensions can represent the centrally-located or important nodes in a category with extreme values. For this purpose, we use centrality properties of nodes which have

proven to be very successful in graph clustering and community detection tasks [26, 27, 28, 29]. In addition, we provide a method for comparing the interpretabilities of different embedding methods. Our method is general and can be applied to any graph embedding method based on different techniques such as random walks, matrix factorization and deep learning.

- **A Survey on Graph Representation Learning Methods** [2]: In this survey, we conduct a review of both traditional and GNN-based graph embedding methods in static and dynamic settings and include over 300 papers consisting of papers published in reputable venues in data mining, machine learning and artificial intelligence. In addition, we summarize a number of limitations of GNN-based methods and the proposed solutions to these limitations. These limitations are expressive power, over-smoothing, scalability, over-squashing, capturing long-range dependencies, design space, neglecting substructures, homophily assumptions, and catastrophic forgetting. Such a summary was not provided in previous surveys. We also provide a list of the real-world applications of GNN-based methods that are deployed in production and suggest a list of future research directions including new ones that are not covered by previous surveys.

1.3 Publications and the Statement of Co-authorships

The list of the publications/submissions during my Ph.D. study under the supervision of Prof. Aijun An is as follows:

1. ***S. Khoshraftar***, A. An, A Survey on Graph Representation Methods, under review by a journal. (Chapter 2 of this dissertation is based on this paper.)

2. S. Mahdavi, ***S. Khoshraftar***, A. An, Dynnode2vec: Scalable dynamic network embedding, pp. 3762-3765, IEEE International Conference on Big Data, 2018. (Chapter 3 of this dissertation is based on this paper.)

3. ***S. Khoshraftar***, S. Mahdavi, A. An, Y. Hu, J. Liu, Dynamic Graph Embedding via LSTM History Tracking, 119-127, IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2019. (Chapter 4 of this dissertation is based on this paper.)

4. ***S. Khoshraftar***, A. An, N. Babanejad, Temporal Graph Representation Learning via Maximal Cliques, IEEE International Conference on Big Data

(IEEE Big Data), Dec 17-20, 2022, Online, Regular paper. (Chapter 5 of this dissertation is based on this paper.)

5. ***S. Khoshraftar***, S. Mahdavi, A. An, Centrality-based Interpretability Measures for Graph Embeddings, 1-10, IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2021. (Chapter 6 of this dissertation is based on this paper.)

6. S. Mahdavi, ***S. Khoshraftar***, A. An, Dynamic Joint Variational Graph Autoencoders, 385-401, European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD) Workshops, 2019.

In the publications where I am listed as the first author, I was responsible for proposing and implementing the methodology, setting up and running the experiments, analyzing the results and writing the manuscript. My co-authors assisted in providing valuable feedback to improve the methodology and experiments and editing the manuscripts. In [5], Sedigheh Mahdavi contributed to proposing the methodology. In addition, she prepared the Dblp and Acm datasets. In the papers in which I am listed as the second author, I assisted in brainstorming the methodology, setting up and running some of the experiments and editing the manuscript.

1.4 Dissertation Organization

In the following chapters, we will elaborate on our contributions. In Chapter 2, we first provide a literature review on graph emebdding methods. In Chapters 3, 4 and 5, we explain the Dynnode2vec, LSTM-Node2vec and TGR-Clique models. At the end of Chapter 5, we compare the design and performance of these three methods. Chapter 6 describes the proposed interpretability measures for graph embeddings. Finally, Chapter 7 concludes the dissertation and provides the future directions.

Chapter 2

Literature Review

2.1 Graphs

Graphs are powerful tools for representing entities and relationships between them. Graphs have applications in many domains including social networks, E-commerce and citation networks. In social networks such as Facebook, nodes in the graph are the people and the edges represent the friendship between them. In E-commerce, the Amazon network is a good example, in which users and items are the nodes and the buying or selling relationships are the edges. Figure 2.1 and 2.2 show examples of social networks and finance networks.

Definition 1 *Formally, a graph G is defined as a tuple $G = (V, E)$ where $V = \{v_0, v_1, ..., v_n\}$ is the set of n nodes/vertices and $E = \{e_0, e_1, ..., e_m\} \subseteq V \times V$ is the set of m edges/links of G, where an edge connects two vertices.*

A graph can be *directed* or *undirected*. In a directed graph, an edge $e_k = (v_i, v_j)$ has a direction with v_i being the starting vertex and v_j the ending vertex. Graphs can be represented by their adjacency, degree and Laplacian matrices, which are defined as follows:

Definition 2 *The adjacency matrix A of a graph G with n vertices is an $n \times n$ matrix, where an element a_{ij} in the matrix equals to 1 if there is an edge between node pair v_i and v_j and is 0 otherwise. An adjacency matrix can be weighted in which the value of an element represents the weight (such as importance) of the edge it represents.*

Definition 3 *The degree matrix D of a graph G with n vertices is an $n \times n$ diagonal matrix, where an element d_{ii} is the degree of node v_i for $i = \{1, ..., n\}$ and all other*

Figure 2.1: Facebook as an example of social networks

$d_{ij} = 0$. *In undirected graphs, where edges have no direction, the degree of a node refers to the number of edges attached to that node. For directed graphs, the degree of a node can be the number of incoming or outgoing edges of that node, resulting in an in-degree or out-degree matrix, respectively.*

Definition 4 *The Laplacian matrix L of a graph G with n vertices is an $n \times n$ matrix, defined as $L = D - A$, where D and A are G's degree and adjacency matrix, respectively.*

2.1.1 Graph Embedding

In order to use graphs in downstream machine learning and data mining applications, graphs and their entities such as nodes and edges need to be represented using numerical features. One way to represent a graph is its adjacency matrix. However, an adjacency matrix is memory-consuming for representing very large graphs because its size is $|V| \times |V|$. We can represent a graph and its elements using their features. Especially, a node in the graph can be represented with a set of features that could help the performance of the representation in a particular application. For example, in anomaly detection application, the nodes with the densest neighborhood have the potential to be anomalous. Therefore, if we include the in-degree and out-degree of nodes in the node representation, we can more likely detect the anomalous nodes with high accuracy because the anomalous nodes often have larger degrees. However,

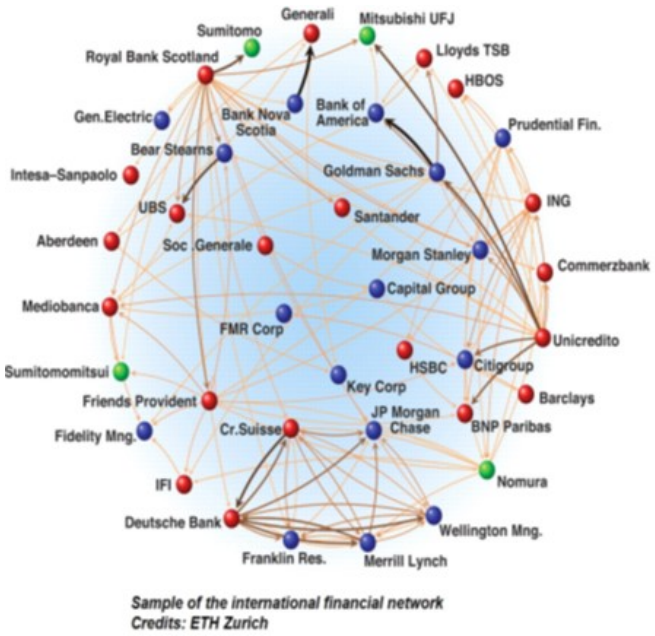


Figure 2.2: Example of finance networks

it could be hard to find features that are important in different applications and can also represent the entire structure of the graph. In addition, it is time consuming to extract these features manually. Therefore, the graph embedding methods have been proposed, which study the issue of automatically generating representation vectors for the graphs. These methods formulate the graph representation learning as a machine learning task and generate embedding vectors leveraging the structure and properties of the graph as input data. Graph embedding techniques include node, edge and subgraph embedding techniques, which are defined as follows.

Definition 5 *(Node embedding). Let $G = (V, E)$ be a graph, where V and E are the set of nodes and the set of edges of the graph, respectively. Node embedding learns a mapping function $f : v_i \rightarrow \mathbb{R}^d$ that encodes each graph's node v_i into a low dimensional vector of dimension d such that $d << |V|$ and the similarities between nodes in the graph are preserved in the embedding space.*

Figure 2.3 shows a sample graph and that an embedding method maps node a in the graph to a vector of dimension 4.

Definition 6 *(Edge embedding). Let $G = (V, E)$ be a graph, where V and E are the set of nodes and the set of edges of the graph, respectively. Edge embedding converts each edge of G into a low dimensional vector of dimension d such that $d << |V|$ and the similarities between edges in the graph are preserved in the embedding space.*

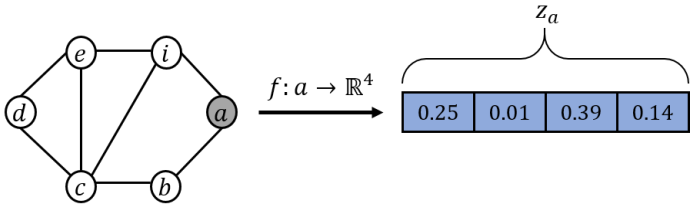


Figure 2.3: The graph on the left hand side consists of 6 nodes $\{a, b, c, d, e, i\}$ and 8 edges. Graph embedding methods map each node of the graph into an embedding vector with dimension d. For the demonstration purpose, the node a is embedded into an embedding vector z_a of dimension 4 with given values.

While edge embeddings can be learned directly from graphs, most commonly they are derived from node embeddings. For example, let $(v_i, v_j) \in E$ be an edge between two nodes v_i and v_j in a graph G and z_i, z_j be the embedding vectors for nodes v_i, v_j. An embedding vector for the edge (v_i, v_j) can be obtained by applying a binary operation such as hadamard product, mean, weighted-L1 and weighted-L2 on the two node embedding vectors z_i and z_j [30].

Definition 7 *(Subgraph embedding). Let $G = (V, E)$ be a graph. Subgraph embedding techniques in machine learning convert a subgraph of G into a low dimensional vector of dimension d such that $d << |V|$ and the similarities between subgraphs are preserved in the embedding space.*

A subgraph embedding vector is usually created by aggregating the embeddings of the nodes in the subgraph using aggregators such as a mean operator.

As node embeddings are the building block for edge and subgraph embeddings, almost all the graph embedding techniques developed so far are node embedding techniques. Thus, the embedding techniques we describe in this survey are mostly node embedding techniques unless otherwise stated.

2.1.2 Graph Embedding Applications

The generated embedding vectors can be utilized in different applications including node classification, link prediction and graph classification. Here, we explain some of these applications.

Node Classification. Node classification task assigns a label to the nodes in the test dataset. This task has many applications in different domains. For instance, in social networks, a person's political affiliation can be predicted based on his friends in the network. In node classification, each instance in the training dataset is the

node embedding vector and the label of the instance is the node label. Different regular classification methods such as Logistic Regression and Random Forests can be trained on the training dataset and generate the node classification scores for the test data. Similarly, Graph classification can be performed using graph embedding vectors.

Link Prediction. Link prediction is one of the important applications of node embedding methods. It predicts the likelihood of an edge formation between two nodes. Examples of this task include recommending friends in social networks and finding biological connections in biological networks. Link prediction can be formulated as a classification task that assigns a label for edges. Edge label 1 means that an edge is likely to be created between two nodes and the label is 0 otherwise. For the training step, a sample training set is generated using positive and negative samples. Positive samples are the edges the exist in the graph. Negative samples are the edges that do not exist and their representation vector can be generated using the node vectors. Similar to node classification, any classification method can be trained on the training set and predict the edge label for test edge instances.

Anomaly Detection. Anomaly detection is another application of node embedding methods. The goal of anomaly detection is to detect the nodes, edges, or graphs that are anomalous and the time that anomaly occurs. Anomalous nodes or graphs deviate from normal behavior. For instance, in banks' transaction networks, people who suddenly send or receive large amounts of money or create lots of connections with other people could be potential anomalous nodes. An anomaly detection task can be formulated as a classification task such that each instance in the dataset is the node representation and the instance label is 0 if the node is normal and 1 if the node is anomalous. This formulation needs that we have a dataset with true node labels. One of the issues in anomaly detection is the lack of datasets with true labels. An alleviation to this issue in the literature is generating synthetic datasets that model the behaviors of real world datasets. Another way to formulate the anomaly detection problem, especially in dynamic graphs, is viewing the problem as a change detection task. In order to detect the changes in the graph, one way is to compute the distance between the graph representation vectors at consecutive times. The time points that the value of this difference is far from the previous normal values, a potential anomaly has occurred [3].

Graph Clustering. In addition to classification tasks, graph embeddings can be used in clustering tasks as well. This task can be useful in domains such as social networks for detecting communities and biological networks to identify similar groups of proteins. Groups of similar graphs/node/edges can be detected by apply-

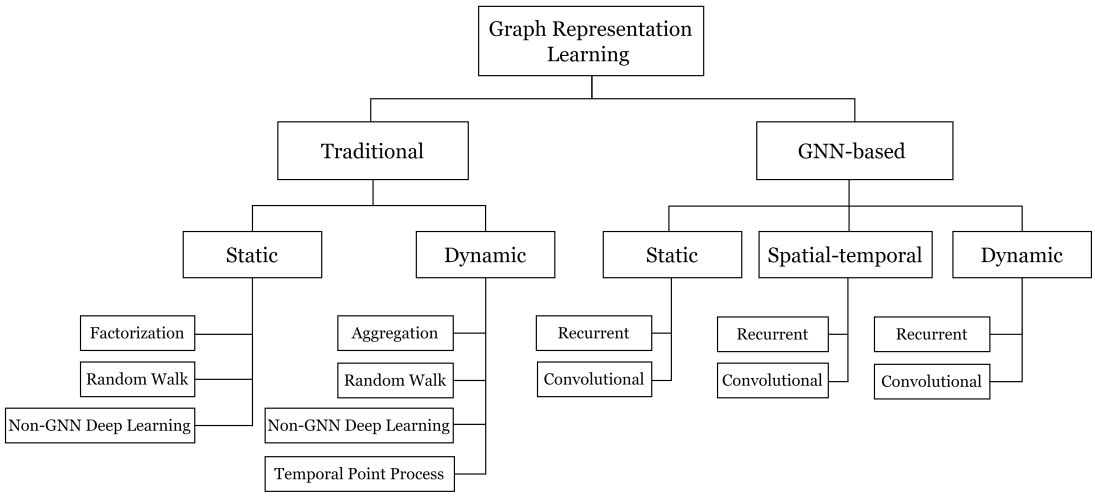


Figure 2.4: Categories of Graph Representation Learning Methods.

ing clustering methods such as the Kmeans method [31] on the graph/node/edge embedding vectors.

Visualization. One of the applications of node embedding methods is graph visualization because node embedding methods map nodes in lower dimensions and the nodes, edges, communities and different properties of graphs can be better seen in the embedding space. Therefore, graph visualization is very helpful for the research community to gain insight into graph data, especially very large graphs that are hard to visualize.

2.2 Traditional Graph Embedding

The first category of graph embedding methods are traditional graph embedding methods. These methods map the nodes into the lower dimensions using different approaches such as random walks, factorization methods, and temporal point processes. We review these methods in static and dynamic settings in this section. Figure 2.4 shows the categories of static and dynamic traditional embedding methods. The upper part of Table 2.1 lists all the methods that we survey in this category.

2.2.1 Traditional Static Graph Embedding

The traditional static graph embedding methods are developed for static graphs. The static graphs do not change over time and have a fixed set of nodes and edges. Graph embedding methods preserve different properties of nodes and edges in graphs

Table 2.1: Graph embedding methods in both traditional and GNN-based categories. Trad and GNN stand for Traditional and GNN-based graph embedding.

Type	Graph	Methods
Trad	Static	Node2vec [30], Deepwalk [32], Graph Factorization [33], GraRep [34], HOPE [35], STRAP [36], HARP [37], LINE [38], SDNE [39], DNGR [40], VGAE [41], AWE [42], PRUNE [43], E[D] [44], ULGE [45], APP [46], CDE [47], GNE [48], DNE [49], DANE [50], RandNE [51], SANE [52], BANE [53], LANE [54], VERSE [55], ANECP [56], NOBE [57], AANE [58], Reinforce2vec [59], REFINE [60], M-NMF [61], struct2vec [62], SNEQ [63], PAWINE [64], FastRP [65], SNS [66], InfiniteWalk [67], EFD [68], NetMF [69], Lemane [70], AROPE [71], NetSMF [72], SPLITTER [73], Ddgk [74], GVNR [75], LouvainNE [76], HONE [77], CAN [78], Methods in [79, 80, 81]
	Dynamic	CTDNE [5], DynNode2vec [6], LSTM-Node2vec [7], EvoNRL [82], DynGEM [3], Dyn-VGAE [83], DynGraph2vec [84], HTNE [85], DynamicTriad [86], DyRep [87], MTNE [88], DNE [89], Toffee [90], HNIP [91], tdGraphEmbed [92], DRLAN [93], TIMERS [94], M2DNE [95], DANE [96], TVRC [97], tNodeEmbed [98], NetWalk [99], DynamicNet [100], Method in [101]
GNN	Static	RecGNN [102], GGNN [103], IGNN [104], Spectral Network [105], GCN [106], GraphSAGE [107], DGN [108], ElasticGNN [109], SGC [110], GAT [111], MAGNA [112], MPNN [113], GN block [114], GNN-FiLM [115], GRNF [116], EGNN [117], BGNN [118], MuchGNN [119], TinyGNN [120], GIN [121], RP-GNN [122], k-GNN [123], PPGN [124], RingGNN [125], F-GNN [126], DEGNN [127], GNNML [128], rGIN [129], DropGNN [130], PEG [131], GraphSNN [132], NGNN [133], ID-GNN [134], CLIP [135], APPNP [136], JKNET [137], GCN-PN [138], DropEdge [139], DGN-GNN [140], GRAND [141], GCNII [142], GDC [143], PDE-GCN [144], SHADOW-SAGE [145], ClusterGCN [146], FastGCN [147], LADIES [148], GraphSAINT [149], VR-GCN [150], GBP [151], RevGNN [152], VQ-GNN [153], BNS [154], GLT [155], H2GCN [156], GPR-GNN [157], WRGNN [158], DMP [159], CPGNN [160], U-GCN [161], NLGNN [162], GPNN [163], HOG-GCN [164], Polar-GNN [165], GBK-GNN [166], Geom-GCN [167], GSN [168], MPSN [169], GraphSTONE [170], DeepLPR [171], GSKN [172], SUBGNN [173], DIFFPOOL [174], PATCHY-SAN [175], SEAL [176], DGCNN [177], AGCN [178], DGCN [179], CFANE [180], AdaGNN [181], MCN [182], Method in [183]
	Spatial-temporal	GCRN [184], Graph WaveNet [185], SFTGNN [186], CoST-Net [187], DSTN [188], LightNet [189], DSAN [190], H-STGCN [191], DMSTGCN [192], PredRNN [193], Conv-TT-LSTM [194], ST-ResNet [195], STDN [196], ASTGCN [197], DGCNN [198], DeepETA [199], SA-ConvLSTM [200], STSGCN [201], FC-GAGA [202], ST-GDN [203], HST-LSTM [204], STGCN [205], PCR [206], GSTNet [207], STAR [208], ST-GRU [209], Tssrgcn [210], Test-GCN [211], ASTCN [212], STP-UDGAT [213], STAG-GCN [214], ST-GRAT [215], ST-CGA [216], STC-GNN [217], STEF-Net [218], FGST [219], PDSTN [220], STAN [221], GraphSleepNet [222], DCRNN [223], CausalGNN [224], SLCNN [225], MRes-RGNN [226], Method in [227]
	Dynamic	DyGNN [228], EvolveGCN [229], TGAT [230], CAW-N [10], DySAT [231], EHNA [232], TGN [233], MTSN [9], SDG [234], VGRNN [235], MNCI [236], FeatureNorm [237]

such as node proximities. Here, we define first-order and second-order proximities. Higher order of proximities can be similarly defined.

Definition 8 *(First-order proximity). Nodes that are connected with an edge have first-order proximity. Edge weights are the first-order proximity measures between nodes. Higher weights for edges show more similarity between two nodes connected by the edges.*

Definition 9 *(Second-order proximity). The second-order proximity between two nodes is the similarity between their neighborhood structures. Nodes sharing more neighbors are assumed to be more similar.*

The traditional static graph embedding methods can be categorized into three categories: *factorization based*, *random walk based* and *non-GNN based deep learning* methods [238, 239]. Below, we review these methods and the techniques they used.

Factorization based. Matrix factorization methods are the early works in graph representation learning. These methods can be summarized in two steps [79]. In the first step, a proximity-based matrix is constructed for the graph where each element of the matrix denoted as P_{ij} is a proximity measure between two nodes i, j. Then, a dimension reduction technique is applied in the matrix to generate the node embeddings in the second step. In the Graph Factorization algorithm [33], the adjacency matrix is used as the proximity measure and the general form of the optimization function is as follows:

$$\min_{z_i, z_j} \sum_{v_i, v_j \in V} |z_i^T z_j - a_{ij}| \tag{2.1}$$

where z_i and z_j are the node representation vectors for node v_i and v_j. a_{ij} is the element in the adjacency matrix corresponding to nodes v_i and v_j. In GraRep [34] and HOPE [35], the value of a_{ij} is replaced with other measures of similarity including higher orders of adjacency matrix, Katz index [240], Rooted page rank [241] and the number of common neighbors. STRAP [36] employs the personalized page rank as the proximity measure and approximates the pairwise proximity measures between nodes to lower the computation cost. In [79], a network embedding update algorithm is introduced to approximately compute the higher order proximities between node pairs. In [70], it is suggested that using the same proximity matrix for learning node representations may limit the representation power of the matrix factorization based methods. Therefore, it generates node representations in a framework that

learns the proximity measures and SVD decomposition parameters in an end-to-end fashion. Methods in [69, 71, 45, 49, 51, 53, 57, 58, 60, 61, 70, 72, 73, 75, 77, 81, 79, 43, 56, 54, 47] are other examples of factorization based methods.

Random walk based. Random walk based methods have attracted a lot of attention because of their success in graph representation. The main concept that these methods utilize is generating random walks for each node in the graph to capture the structure of the graph and output similar node embedding vectors for nodes that occur in the same random walks. Using co-occurrence in a random walk as a measure of similarity of nodes is more flexible than fixed proximity measures in earlier works and showed promising performance in different applications.

Definition 10 *(Random walk). In a graph $G = (V, E)$, a random walk is a sequence of nodes $v_0, v_1, ..., v_k$ that starts from node v_0. $(v_i, v_{i+1}) \in E$ and $k+1$ is the length of the walk. Next node in the sequence is selected based on a probabilistic distribution.*

DeepWalk [32] and Node2vec [30] are based on the Word2vec embedding method [242] in natural language processing (NLP). Word2vec is based on the observation that words that co-occur in the same sentence many times have a similar meaning. Node2vec and DeepWalk extend this assumption for graphs by considering that nodes that co-occur in random walks are similar. Therefore, these methods generate similar node embedding vectors for neighbor nodes. The algorithm of these two methods consists of two parts. In the first part, a set of random walks are generated, and in the second part, the random walks are used in the training of a SkipGram model to generate the embedding vectors. The difference between DeepWalk and Node2vec is in the way that they generate random walks. DeepWalk selects the next node in the random walk uniformly from the neighbor nodes of the previous node. Node2vec applies a more effective approach to generating random walks. In this section, we first explain the Node2vec random walk generation and then the SkipGram.

1. *Random Walk Generation.* Assume that we want to generate a random walk $v_0, v_1, ..., v_k$ where $v_i \in V$. Given that the edge (v_{i-1}, v_i) is already passed, the next node v_{i+1} in the walk is selected based on the following probability:

$$P(v_{i+1}|v_i) = \begin{cases} \frac{\alpha_{v_i v_{i+1}}}{Z} & \text{if } (v_{i+1}, v_i) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

   where Z is a normalization factor and $\alpha_{v_i v_{i+1}}$ is defined as:

$$\alpha_{v_i v_{i+1}} = \begin{cases} 1/p & \text{if } d_{v_{i-1}v_{i+1}} = 0 \\ 1 & \text{if } d_{v_{i-1}v_{i+1}} = 1 \\ 1/q & \text{if } d_{v_{i-1}v_{i+1}} = 2 \end{cases} \tag{2.3}$$

where $d_{v_{i-1}v_{i+1}}$ is the length of the shortest path between nodes v_{i-1} and v_{i+1} and takes values from $\{0, 1, 2\}$. The parameters p and q guide the direction of the random walk and can be set by the user. A large value for parameter p encourages global exploration of the graph and avoids returning to the nodes that are already visited. A large value for q on the other hand biases the walk toward local exploration. With the use of these parameters, Node2vec creates a random walk that is a combination of breadth-first search (BFS) and depth-first search (DFS).

2. *SkipGram.* After generating random walks, the walks are input to a SkipGram model to generate the node embeddings. SkipGram learns a language model, which maximizes the probability of sequences of words that exist in the training corpus. The objective function of SkipGram for node representation is:

$$\max_{\Phi} \sum_{v_i \in V} \log P(N(v_i)|\Phi(v_i)) \tag{2.4}$$

where $N(v_i)$ is the set of neighbors of node v_i generated from the random walks. Assuming independency among the neighbor nodes, we have

$$P(N(v_i)|\Phi(v_i)) = \prod_{v_k \in N(v_i)} P(\Phi(v_k)|\Phi(v_i)) \tag{2.5}$$

The conditional probability of $P(\Phi(v_k)|\Phi(v_i))$ is modeled using a softmax function:

$$P(\Phi(v_k)|\Phi(v_i)) = \frac{\exp(\Phi(v_k)\Phi(v_i))}{\sum_{v_j \in V} \exp(\Phi(v_j)\Phi(v_i))} \tag{2.6}$$

The softmax function nominator is the dot product of the node representation vectors. Since the dot product between two vectors measures their similarity, by maximizing the softmax function for neighbor nodes, the generated node representations for neighbor nodes tend to be similar. Computing the denominator of the conditional probability is time consuming between the target node and all the nodes in the graph. Therefore, DeepWalk and Node2vec approximate it using hierarchical softmax and negative sampling, respectively.

In HARP [37] a graph coarsening algorithm is introduced that generates a hierarchy of smaller graphs as $G_0, G_1, ..., G_L$ such that $G_0 = G$. Starting from the G_L which is the smallest graph, the node embeddings that are generated for G_i are used as initial values for nodes in G_{i-1}. This method avoids getting stuck in the local minimum for DeepWalk and Node2vec because it initializes the node embeddings with better values in the training process. The embedding at each step can be created using DeepWalk [32] and Node2vec [30] methods. LINE [38] is not based on random walks but because it is computationally related to DeepWalk and Node2vec, its results are usually compared with them. LINE generates node embeddings that preserve the first-order and second-order proximities in the graph using a loss function that consists of two parts. In the first part L_1, it minimizes the reverse of the dot product between connected nodes. In the second part L_2, for preserving the second-order proximity, it assumes that nodes that have many connections in common are similar. LINE trains two models that minimize L_1 and L_2 separately and then the embedding of a node is the concatenation of its embeddings from two models. Methods in [52, 59, 62, 42, 67, 68, 76, 80, 46, 66, 64] are some other variants of random walk based methods.

Non-GNN based deep learning. SDNE [39] is based on an autoencoder which tries to reconstruct the adjacency matrix of a graph and captures nodes' first-order and second-order proximities. To that end, SDNE jointly optimizes a loss function that consists of two parts. The first part preserves the second-order proximity of the nodes and minimizes the following loss function:

$$L_1 = \sum_{v_i \in V} |(x_i - x_i') \odot b_i| \tag{2.7}$$

where x_i is the row corresponding to node v_i in the graph adjacency matrix and x_i' is the reconstruction of x_i. b_i is a vector consisting of b_{ij}s for j from 1 to n (the number of nodes in the graph). If $a_{ij} = 0, b_{ij} = 1$; otherwise, $b_{ij} = \beta > 1$. a_{ij} is the element corresponding to nodes v_i and v_j in the adjacency matrix. Using b_i, SDNE assigns more penalty for the error in the reconstruction of the non-zero elements in the adjacency matrix to avoid reconstructing only zero elements in sparse graphs. The second part capturs the first-order similarity and optimizes $L2$:

$$L_2 = \sum_{(v_i, v_j) \in E} a_{ij} |(z_i - z_j)| \tag{2.8}$$

where z_i and z_j are the embedding vectors for nodes v_i and v_j, respectively. In this way, a higher penalty is assigned if the difference between the embedding vectors

of two nodes connected by an edge is higher, resulting in similar embedding vectors for nodes connecting with an edge. This loss is based on ideas from Laplacian Eigenmaps [243]. SDNE jointly optimizes L_1 and L_2 to generate the node embedding vectors. The embedding method DNGR [40] is also very similar to SDNE with the difference that DNGR uses pointwise mutual information of two nodes co-occurring in random walks instead of the adjacency matrix values. VGAE [41] is a variant of variational autoencoders [244] on graph data. The variational graph encoder encodes the observed graph data including the adjacency matrix and node attributes into low-dimensional latent variables.

$$q(Z|A,X) = \prod_{i=1}^{N} q(z_i|A,X),$$
$$with\ q(z_i|A,X) = N(z_i|\mu_i, diag(\sigma_i^2))$$

where z_i is the embedding vector for node v_i, μ_i is a mean vector and σ_i is the log standard deviation vector of node v_i. A and X are the adjacency matrix and attribute matrix of the graph, respectively. The variational graph decoder decodes the latent variables into the distribution of the observed graph data as follows:

$$p(A|Z) = \prod_{i=1}^{N}\prod_{j=1}^{N} p(a_{i,j}|z_i, z_j),$$
$$with\ p(a_{i,j} = 1|z_i, z_j) = sigmoid(z_i^T, z_j)$$

The model generates embedding vectors that minimize the distance between the p and q probability distributions using the KL-divergence measure, SGD and reparametrization trick. Other works in [55, 50, 74, 78, 65, 44, 48, 63], also learn node embeddings using non-GNN based deep learning models.

2.2.2 Traditional Dynamic Graph Embedding

Most real-world graphs are dynamic and evolve, with nodes and edges added and deleted from them. Dynamic graphs are represented in two ways in the dynamic graph embedding studies: discrete-time and continuous-time.

Definition 11 *(Discrete-time dynamic graphs). In discrete-time dynamic graph modeling, dynamic graphs are considered a sequence of graphs' snapshots at consecutive time points. Formally, dynamic graphs are represented as* $G = G_0, G_1, ..., G_T$ *which* G_i *is a snapshot of the graph* G *at timestamp* i*. The dynamic graph is divided*

into graph snapshots using a time granularity such as hours, days, months and years depending on the dataset and applications.

Definition 12 *(Continuous-time dynamic graphs). In continuous-time dynamic graph modeling, the time is continuous, and the dynamic graph can be represented as a sequence of edges over time. The dynamic graph can also be modeled as a sequence of events, where events are the changes in the dynamic graphs, such as adding/deleting edges/nodes.*

Definition 13 *(Dynamic graph embedding). We can use either the discrete-time or the continuous-time approach for representing a dynamic graph. Let* $G_t = (V_t, E_t)$ *be the graph at time* t *with* V_t, E_t *as the nodes and edges of the graph. Dynamic graph embedding methods map nodes in the graph to a lower dimensional space* d *such that* $d << |V|$.

Dynamic graph embedding methods are more challenging than static graph embedding methods because of the challenges in modeling the evolution of graphs. Different methods have been proposed for dynamic graph embedding recently. Here, we provide an overview of the dynamic embedding methods and categorize these methods into four categories: Aggregation based, Random walk based, Non-GNN based deep learning and Temporal point process based [245, 246].

Aggregation based. Aggregation based dynamic graph embedding methods aggregate the dynamic information of graphs to generate embeddings for dynamic graphs. These methods can fall into two groups:

1) Aggregating the temporal features. In these methods, the evolution of the graph is simply collapsed into a single graph and the static graph embedding methods are applied on the single graph to generate the embeddings. For example, the aggregation of the graph over time could be the sum of the adjacency matrices for discrete-time dynamic graphs [247] or the weighted sum which gives more weights to recent graphs [97]. One drawback of these methods is that they lose the time information of graphs that reveals the dynamics of graphs over time. For instance, there is no information about when any edge was created. *Factorization-based models* can also fit into the aggregation based category. The reason is that factorization based models save the sequence of graphs over time in a three dimensional tensor $\in \mathbb{R}^{|V|\times|V|\times T}$ (T is a time dimension) and then apply factorization on this tensor to generate the dynamic graph embeddings [248, 93, 94, 96].

2) Aggregating the static embeddings. These aggregation methods first apply static embedding methods on each graph snapshot in the dynamic graph sequence. Then, these embeddings are aggregated into a single embedding matrix for all the nodes in the graph. These methods usually aggregate the node embeddngs by considering a decay factor that assigns a lower weight to older graphs [100, 101]. In another type of these methods, the sequence of graphs from time 0 to $t-1$ are fit into a time-series model like ARIMA that predicts the embedding of the next graph at time $t+1$ [249].

Random walk based. Random walk based approaches extend the concept of random walks in the static graphs for dynamic graphs. Random walks in dynamic graphs capture the time dependencies between graphs over time in addition to the topological structure of each of the graph snapshots. Depending on the definition of random walks, different methods include the temporal information of the graphs differently. CTDNE [5] defines a temporal walk to capture time dependencies between nodes in dynamic graphs. CTDNE considers a continuous-time dynamic graph such as graph $G = (V, E_T, T)$ which V, E_T, T are nodes and edges of the graph and time $T : E \rightarrow \mathbb{R}^+$. Each edge e in this graph is represented by a tuple (u, v, t) which u, v are the nodes connected by the edge and t is the time of occurrence of that edge.

Definition 14 *(Temporal walk). A temporal walk is a sequence of nodes* $v_0, v_1, ..., v_k$ *such that* $(v_i, v_{i+1}) \in E_T$ *and* $t_{(v_{i-1},v_i)} \leq t_{(v_i,v_{i+1})}$.

An important concept in CTDNE is that time is respected in selecting the next edge in a temporal walk. In order to generate these temporal walks, first a time and a particular edge in that time, $e = (u, v, t_e)$ is selected based on one of three probability distributions: uniform, exponential and linear. The uniform probability for an edge e is $p(e) = 1/|E_T|$. The exponential probability is:

$$p(e) = \frac{exp(t_e - t_{min})}{\sum_{e' \in E_T} exp(t'_e - t_{min})} \tag{2.9}$$

where t_{min} is the minimum time of an edge in the graph. Using exponential probability distribution, edges that appear at a later time are more likely to be selected. After selecting $e = (u, v, t_e)$, the next node in the temporal walk is selected from the neighbors of node v in time $t_e + k$ where $k > 0$ again using one of the uniform, exponential or linear probability distributions. The generated temporal walks are then input to a SkipGram model and the temporal node representation vectors are generated.

DynNode2vec [6] is a dynamic version of Node2vec [30] and uses a discrete-time approach for dynamic graph representation learning. This method represents the dynamic graph as a sequence of graph snapshots over time as $G_0, G_1, ..., G_T$. The embedding for the graph at time 0, G_0 is computed by applying Node2vec on G_0. Then, for next time points, the SkipGram model of G_{t+1} is initialized using node representations from G_t for nodes that are common between consecutive time points. New nodes will be initialized randomly. In addition, dynnode2vec does not generate random walks at each time step i from scratch. Instead, it uses random walks from the previous time $i-1$ and only updates the ones that need to be updated. This method has two advantages. First, it saves time because it does not generate all the walks in each step. Second, since the SkipGram model at time t is initialized with weights from time $t-1$, embedding vectors of consecutive times are in the same embedding space, embedding vectors of nodes change smoothly over time and the model converges faster. LSTM-Node2vec [7] captures both the static structure and evolving patterns in graphs using an LSTM autoencoder and a Node2vec model. The dynamic graph is represented as a sequence of snapshots over time as $G_0, G_1, ..., G_T$. For each graph G_i at time t_i, first, a set of temporal walks is generated for each node in the graph. Each temporal random walk of a node v is represented as $w_0, w_1, ..., w_L$ of length L which w_j is a neighbor of the node v at time t_j in graph G_j and $t_j < t_{j+1}$. These temporal walks demonstrate changes in the neighborhood structure of the node before time t_i. EvoNRL [82] focuses on maintaining a set of valid random walks for the graph at each time point so that the generated node embeddings using these random walks stay accurate. To that end, EvoNRL updates the existing random walks from previous time points instead of generating random walks from scratch. Specifically, it considers four cases of edge addition, edge deletion, node addition and node deletion for evolving graphs and updates the affected random walks accordingly. For instance, in the edge addition case, EvoNRL finds random walks containing the nodes which are connected by the updated link and updates those walks. However, updating random walks is time consuming, especially for large graphs. Therefore, EvoNRL proposed an indexing mechanism for fast retrieval of random walks. Other examples of these category include [90, 89, 99, 92].

Non-GNN based deep learning. This type of dynamic graph embedding methods use deep learning models such as RNNs and autoencoders. DynGEM [3] is based on the static deep learning based graph embedding method SDNE [39]. Let the dynamic graph be a sequence of graph snapshots $G_0, G_1, ..., G_t$. The embeddings for graph G_0 are computed using a SDNE model. The embedding of G_i is obtained by running a SDNE model on G_i that is initialized with the embeddings from G_{i-1}.

This initialization leads to generating node embeddings at consecutive time points that are in the same embedding space and can reflect the changes in the graph at consecutive times accurately. As the size of the graph can change over time, DynGEM uses Net2WiderNet and Net2DeeperNet to account for bigger graphs [250]. Dyn-VGAE [83] is a dynamic version of VGAE [41]. The input to dyn-VGAE is the dynamic graph as a sequence of graph snapshots, $G_0, G_1, ..., G_T$. At each time point, the embedding of the graph snapshot G_i is obtained using VGAE. However, the loss of the model at time t has two parts. The first part is related to VGAE loss and the second loss is a KL divergence measure that minimizes the difference between two distributions as follows:

$$L_s^t = KL[q_t(Z_t|X_t, A_t)||N(Z_{t-1}, \sigma^2)] \tag{2.10}$$

where $q_t(Z_t|X_t, A_t)$ is the distribution of latent vectors at time t and $N(Z_{t-1}, \sigma^2)$ is a normal distribution with mean Z_{t-1} and standard deviation σ. This loss places the current latent vectors Z_t near latent vectors of previous time point Z_{t-1}. The loss function of all the models for the graph at time 0 to T are jointly trained. Therefore, the generated representation vectors preserve both the structure of the graph at each time point and evolutionary patterns obtained from previous time points. Dyngraph2vec [84] generates embeddings at time t using an autoencoder. This method inputs adjacency matrices of previous times $A_0, A_1, ..., A_{t-1}$ to the encoder and using the decoder reconstructs the input and generates the embeddings at time t. Dyngraph2vec proposes several variants using a fully connected model or a RNN/LSTM model for the encoder and the decoder: dyngraph2vecAE, dyngraph2vecAERNN and dyngraph2vecRNN. Dyngraph2vecAE uses an autoencoder, dyngraph2vecAERNN is based on an LSTM autoencoder and dyngraph2vecAERNN has a LSTM enocoder and a fully connected decoder. Other examples of non-GNN based deep learning methods include [91, 98].

Temporal point process based. This class of the dynamic graph embedding methods assumes that the interaction between nodes for creating the graph structure is a stochastic process and models it using temporal point processes. HTNE [85] generates embeddings for dynamic graphs by modeling the neighborhood formation of nodes as a hawkes process. In a hawkes process modeling, the occurrence of an event at time t is influenced by events that occur before time t and a conditional intensity function characterizes this concept. Let $G = (V, E, A)$ be the temporal network which V, E, A are the nodes, edges and events. Each edge (v_i, v_j) in this

graph is associated with a set of events $a_{ij} = \{a_1 \rightarrow a_2 \rightarrow ...\} \subset A$ where each a_i is an event at time i.

Definition 15 *(Neighborhood formation sequence). A neighborhood formation sequence for a node v_i is a series of neighborhood arrival events $\{v_i : (u_0, t_0) \rightarrow (u_1, t_1)... \rightarrow (u_k, t_k)\}$ where u_i is a neighbor of v_i that occurs at time t_i.*

HTNE models the neighborhood formation for a node v using the neighborhood formation sequence H_v. The probability that an edge forms between node v and a target neighbor u at time t is represented using the following formula:

$$p(u|v, H_v) = \frac{\lambda_{u|v}(t)}{\sum_{u'} \lambda_{u'|v}(t)} \tag{2.11}$$

where $\lambda_{y|x}(t)$ is defined:

$$\lambda_{u|v}(t) = exp(\mu_{u,v} + \sum_{h,u} \alpha_{h,u} \kappa(t - t_h)) \tag{2.12}$$

$\lambda_{u|v}(t)$ is the conditional intensity function of a hawkes process which is the arrival rate of target neighbor u for node v at time t given the previous neighborhood formation sequence. $\mu_{u,v}$ is a base rate of edge formation between u, v and it is equal to $|z_u - z_v|$. h is a historical neighbor of the node v in the neighborhood formation sequence in a time before t. $\alpha_{h,u}$ is the degree that the historical neighbor h is important for u and it equals $|z_h - z_u|$. $\kappa(t - t_h)$ is a decay factor to control the intensity of influence of a historical node on u. HTNE generates embedding vectors that maximize the $\sum_{v \in V} \sum_{u \in H_v} p(u|v, H_v)$ for all the nodes using SGD and negative sampling to deal with a large number of computations in the denominator of the probability function. DyRep [87] captures the dynamic of graphs using two temporal point process models. DyRep argues that in the evolution of a graph two types of events occur: communication and association. Communication events are related to node interactions and association events are the topological evolution and these events occur at different rates. For instance, in a social network, a communication event such as liking a post from someone happens much more frequently than an association event like creating a new friendship. DyRep represents these two events as two temporal point process models. MTNE [88] is based on two concepts of triad motif evolution and the hawkes process. This method considers the evolution of graphs as the evolution of motifs in the graphs and models that evolution using a hawkes process. MTNE argues that a model such as HTNE based on neighborhood formation processes considers network evolution at edge and node levels and can

not reflect network evolution very well. Therefore, MTNE models dynamics in a graph as a subgraph (motif) evolution process. M2DNE [95] is another example of temporal point process based dynamic embedding methods.

Other methods. DynamicTriad [86] generates dynamic graph embeddings by modeling the triad closure process, which is a fundamental process in the evolution of graphs.

Definition 16 *(Triad closure process). Let (v_i, v_j, v_k) be an open triad in the graph at time t which means that there are two edges (v_i, v_j) and (v_j, v_k) in the graph but no edge exists between v_i and v_k. It is likely that an edge forms between v_i and v_k at time $t+1$ because of the influence of node v_j and closes the open triad.*

DynamicTriad computes the probability that an open triad (v_i, v_j, v_k) evolves into a closed triad under the influence of v_j at time t as $p^t_{tr}(i, j, k)$. An open triad can evolve in two ways: 1) It becomes closed because of the influence of any one of the neighbors. 2) Stays open because no neighbor could influence the creation of the open link. These two evolution traces are reflected in DynamicTriad loss function by maximizing $(p^t_{tr}(i, j, k))^{\alpha_{ijk}} \times (1 - p^t_{tr}(i, j, k))^{(1-\alpha_{ijk})}$ for open triad samples that close under the influence of a neighbor and $1 - p^t_{tr}(i, j, k)$ for those samples that do not close. $\alpha_{ijk} = 1$ if an open triad closes at time $t+1$. The loss function also utilizes social homophily and temporal smoothness regularizations. *Social homophily smoothness* assumes that nodes that are highly connected are more similar and should have similar embeddings. *Temporal smoothness* assumes that network evolves smoothly and therefore, the distance between embeddings of a node at consecutive times should be small.

2.3 GNN based graph embedding

Graph Neural Net (GNN) based graph embedding methods are the second category of graph embedding methods, which employ GNNs to generate embeddings. These methods are different from traditional methods in that the GNN-based methods generalize well to unseen nodes. In addition, they can better take advantage of node/edge attributes. Table 2.2 shows the advantages and disadvantages of the different categories of graph embedding methods. In this section, we first introduce GNNs. Then, three categories of GNN-based methods including static, spatial-temporal and dynamic GNNs (see Figure 2.4 for subcategories and Table 2.1 for the list of methods in each category) and their real-world applications are surveyed.

Table 2.2: Comparison of traditional and GNN-based graph representation learning.

Category	Advantages	Disadvantages
Traditional	Higher expressive power, scalable in some categories	Not generalizable to unseen nodes, not considering node/edge attributes easily
GNN-based	Generalize to unseen nodes, consider node/edge attributes, can do both task-specific and node similarity based training	Expressive power, scalability, over-smoothing, over-squashing, homophily assumption and catastrophic forgetting. (More details in Section 2.3.6)

Finally, we summarize the limitations of GNNs and the proposed solution to these limitations.

2.3.1 Introduction to GNNs

A GNN is a deep learning model which generates a node embedding by aggregating the node's neighbors embeddings. The GNN's intuition is that a node's state is influenced by its interactions with its neighbors in the graph. Below, we explain GNN's basic architecture and training.

Basic architecture. GNNs can generate node representation vectors by stacking several GNN layers. Let h_i^l represent the node embeddings for node i at layer l. Each GNN layer takes as input the nodes embeddings. The node representations for node i at each layer $l+1$ are updated using the following formula:

$$h_i^{(l+1)} = f(h_i^l, \sum_{j \in N(i)} g(i,j)) \tag{2.13}$$

where f and g are learnable functions and $N(i)$ are the neighbors of node i. h_i^0 are the node i initial features. At each layer, the embedding of the node i is obtained by aggregating the embeddings of the node's neighbors. After passing through L GNN layers, the final representation of node i is h_i^L, which is the aggregation the node's neighbors of L hops away from the node.

GNN training. GNNs can be trained in supervised, semi-supervised and unsupervised frameworks. In supervised and semi-supervised frameworks, different prediction tasks focusing on nodes, edges and graphs can be employed for training the model. Here, we describe the other layers stacked after GNN layers to generate the prediction results.

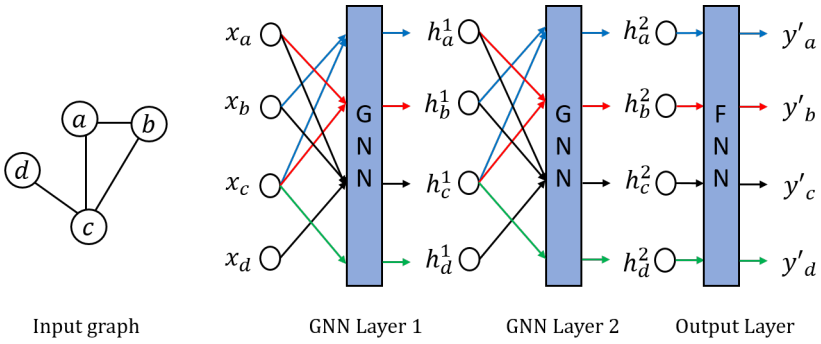


Figure 2.5: A general supervised framework for training Graph neural net layers. Two GNN layers are applied on an input graph to compute the node representation vectors for its nodes. The colors on arrows show neighbors of a target node that are aggregated to generate the target node representation. x_a is the feature vector of node a and h_a^1 and h_a^2 are the representation vectors generated for the node a after applying the first and second GNN layers. The generated embeddings are used in a node classification task. y'_a is the predicted label for the node a.

- Node-focused: For node-level prediction such as node classification, the GNN layers output node representations and then using an MLP or a softmax layer the prediction output is generated.
- Edge-focused: In edge-focused prediction including link prediction, given two nodes' representations, a similarity function or an MLP is used for the prediction task.
- Graph-focused:In graph-focused tasks such as graph classification, a graph representation is often generated by applying a readout layer on node representations. The readout function can be a pooling operation that aggregates representations of a graph's nodes to generate the graph representation vector. A clique pooling operation has also been proposed, which aggregates a graph's cliques for generating the graph embedding [251].

A typical way to train a GNN in a node classification task is by applying the cross entropy loss function as follows:

$$L = \sum_{i \in V_{train}} (y_i \log(\sigma(h_i^T \theta)) + (1 - y_i) \log(1 - \sigma(h_i^T \theta)) \tag{2.14}$$

where h_i is the embedding of node i, which is the output of the last layer of GNN and y_i is the true class label of the node and θ are the classification weights. Figure 2.5 shows a general framework for training a GNN using a node classification task. There are three types of nodes in a node classification in GNN [252]:

- Training nodes: Nodes which their embeddings are computed in the last layer of GNNs and are included in the loss function computation.
- Transductive test nodes: Nodes which their embeddings are computed in the GNN but they are not included in the loss function computation.
- Inductive test nodes: They are not included in the GNN computation and loss function.

Transductive node classification in GNNs is equivalent to semi-supervised node classification. It refers to testing on transductive test nodes that are observed during training but their labels are not used. On the other hand, inductive node classification means that the testing is on inductive test nodes (unseen nodes) where these test nodes and all their adjacent edges are removed during training. The loss function for graph classification and link prediction tasks can be similarly defined using graph representations and pair wise node representations. In an unsupervised framework for GNN training, node similarities obtained from co-occurrence of nodes in the graph random walks can be used for model training. Graph neural nets often compute node representations using a graph-level implementation to avoid redundant computations for neighbors which are shared among nodes. In addition, formulating the message passing operations as matrix multiplications are computationally cheap. As an example for a basic GNN, the node embedding computation formula can be reformulated as:

$$H^{(l+1)} = \sigma(\hat{A}H^{l}W^{l}) \tag{2.15}$$

where H^l contains the embedding of all the nodes in layer l and W_l is the weight matrix at layer l. $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, where $\tilde{A} = A + I_n$, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, I_n is an identity matrix and A, D are the graph's adjacency and degree matrices. The graph-level implementation avoids redundant computations, however, it needs to operate on the whole graph which may lead to memory limitations. A number of methods have been proposed to alleviate the memory complexities of GNNs which will be discussed in Section 2.3.6.

Other important concepts in GNNs. In this section, we define some of the concepts that are frequently used in the GNN based graph representation literature.

- **Receptive Field.** The receptive field of a node in GNNs are the nodes that contribute to the final representation of the node. After passing through each

layer of the GNN the receptive field of a node grows one step towards its distant neighbors.

- **Graph Isomorphism.** Two graphs are isomorphic if they have a similar topology. Some of the early works on GNN such as GCN [106] and GraphSAGE[107] fail to distinguish non-isomorphic graphs in some cases.

- **Weisfeiler & Lehman (WL) test.** The WL test [253] is a classic algorithm for testing graph isomorphism. It has been shown that the representation power of the message passing GNNs is upper bounded by this test [121]. The WL test successfully determines isomorphism between different graphs but there are some corner cases that it fails. Similarly, GNNs fails in those cases. The simple way of thinking about how this test works is that it first counts the number of nodes in two graphs. If two graphs have a different number of nodes, they are different. If two graphs have a similar number of nodes, it checks the number of immediate neighbors of each node. If the number of immediate neighbors of each node is the same, it goes to check the second-hop neighbors of nodes. If two graphs were similar in all these cases, then they are identical or isomorphic.

- **Skip connections.** A skip connection in deep architectures means skipping some layers in the neural network and feeding one layer's output as an input to the next layers, not just the immediate next layer. An skip connection helps in alleviating the vanishing gradient effect and preserving information from previous layers. For instance, skip connections are used in GraphSAGE [107] update step. This method concatenates the node representation at the previous level with the aggregated representation from node neighbors from the previous layer in the update step. This way, it preserves more node-level information in the message passing.

2.3.2 Static Graph Neural Nets

Static GNN based graph embedding methods are suitable for graph representation learning on static graphs, which do not change over time. These methods can be divided into two classes: Recurrent GNNs and Convolutional GNNs that will be explained below.

Recurrent Graph Neural Net (RecGNN). RecGNNs are the early works on GNN that are based on RNNs. The original GNN model proposed by Scarselli et al.

[102] used the assumption that nodes in a graph constantly exchange information until they reach an equilibrium. In this method, the representation of node v at iteration t, h_v^t is defined using the following recurrence equation:

$$h_v^t = \sum_{u \in N(v)} f(x_v, x^e_{(v,u)}, h_u^{(t-1)}, x_u) \tag{2.16}$$

where f is a recurrent function. $N(v)$ is a set of neighborhood nodes of node v. x_v, x_u are feature vectors of nodes v, u and $x^e_{(v,u)}$ is the feature vector of the edge (u, v). This GNN model recursively runs until convergence to a fixed point. Therefore, the final representation h_v^T in this method is a vector that $h_v^T = f(h_v^{T-1})$. In this model, h_v^0 is initialized randomly. The initialization of node representation vectors does not matter in this model because the function f recursively converges to the fixed point using any value as an initialization. For learning the model parameters, the states h_v^t are iteratively computed until the iteration T. An approximate fixed point solution is obtained and used in a loss function to compute the gradients. This model has several limitations. One limitation is that if T is large, the iterative computation of node representation until convergence is time consuming. Furthermore, the node representations obtained from this model are more suitable for graph representation than node representation as the outputs are very smooth. GGNN [103] uses gated recurrent unit (GRU) as the recurrent function in the original RecGNN method proposed by Scarselli et al. [102]. The advantage of using GRU is that the number of recurrence steps is fixed and the aggregation does not need to continue until convergence. The h_v^t formula is as follows:

$$h_v^t = GRU(h_v^{(t-1)}, \sum_{u \in N(v)} h_u^{(t-1)}) \tag{2.17}$$

The h_v^0 are initialized with node features. Implicit Graph Neural Net (IGNN) [104] is another recurrent GNN that generates node representations by iterating until convergence with no limit on the number of neighbor hops. However, it guarantees the existence of the solution for the equilibrium equations by defining the concept of well-posedness for GNNs, which was previously defined for neural nets [254] and enforces it at the training time.

Convolutional Graph Neural Net (ConvGNN). ConvGNNs are a well-known category of graph neural nets. These methods generate node embeddings using the concept of convolution in graphs. The difference between ConvGNNs and RecGNNs

is that ConvGNNs use CNN based layers to extract node embeddings instead of RNN layers in RecGNNs. There are three key characteristics in CNNs that make them attractive in graph representations. 1) Local connections: CNN can extract local information from neighbors for each node in the graph, 2) shared weights: weight sharing in node representation generates node embeddings that consider the information of other nodes in the graph, 3) multiple layers: each layer of convolution can explore a layer of proximities between nodes [255]. ConvGNNs have two categories that can overlap: Spectral based and Spatial based methods. The spectral based methods have roots in graph signal processing and define graph signal filters. The spatial based methods are based on information propagation and message passing concepts from RecGNNs and are more preferred than spectral methods because of efficiency and flexibility. Here, we explain these two categories in more detail.

1) Spectral based. Spectral based graph neural nets utilize mathematical concepts from graph signal processing. **Spectral Network** [105] is one of the early works that defines convolution operation on graphs. Here, we define some of the main concepts shared among spectral based GNNs.

Definition 17 *(Graph signal). In graph signal processing, a graph signal $x \in \mathbb{R}^n$ is an array of n real or complex values for n nodes in the graph.*

Definition 18 *(Eigenvectors and eigenvalues (spectrum)). Let $L = D - A$ be the graph Laplacian of graph G which D, A are the graph's degree matrix and adjacency matrix. The normalized graph Laplacian matrix is $L_N = I_n - D^{-1/2}AD^{-1/2}$ which can be factorized as $L_N = U\Lambda U^T$. U is the matrix of eigenvectors and Λ is the diagonal matrix of ordered eigenvalues. The set of eigenvalues of a matrix are also called the spectrum of the matrix.*

Definition 19 *(Graph Fourier transform). The graph fourier transform is $F = U^T x$ which maps the graph signal x to a space formed by the eigenvectors of L_N.*

Definition 20 *(Spectral graph convolution). The spectral graph convolution of the graph signal x with a filter $g \in \mathbb{R}^n$ is defined as:*

$$x * g = F^{-1}(F(x) \odot F(g)) = U(U^T x \odot U^T g) \tag{2.18}$$

$$\Rightarrow x * g_\theta = U g_\theta U^T x \tag{2.19}$$

$$where\ g_\theta = diag(U^T g) \tag{2.20}$$

Different spectral based ConvGNNs use a different graph convolution filter g_θ. For instance, Spectral CNN [105] defines g_θ as a set of learnable parameters. One of the main limitations of this method is the eigenvalue decomposition computational complexity. This limitation was resolved by applying several approximations and simplification in future works. Graph Convolutional Network (GCN) [106] uses a layerwise propagation rule based on multiplying the first-order approximation of localized spectral convolution filter g_θ with a graph signal x as follows:

$$x * g_\theta = \theta(I_n + D^{-1/2}AD^{-1/2})x \tag{2.21}$$

where D, A are the degree matrix and adjacency matrix of a graph G and I_n is an identity matrix with 1 on the diagonal and 0 elsewhere. θ represents the filter parameters. GCN also modifies the convolution operation into a layer defined as $H = X*g_\Theta = f(\bar{A}X\Theta)$, where f is an activation function and $\bar{A} = I_n + D^{-1/2}AD^{-1/2}$. Using a renormalization trick $\bar{A}$ is replaced with $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, where $\tilde{A} = A + I_n$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. Therefore, the formulation of h_v^{l+1} for node v becomes:

$$h_v^{l+1} = f(\Theta^l(\sum_{u \in \{N(v) \cup v\}} \hat{A}_{v,u} x_u)) \tag{2.22}$$

where $\hat{A}$ is a constant and is approximately computed in a preprocessing step. $N(v)$ is the set of neighbors of a node v. Therefore, h_v^{l+1} value can be roughly approximated as:

$$h_v^{l+1} \approx f(\Theta^l.Mean(h_v^l \cup \{h_u^l, \forall u \in N(v)\})) \tag{2.23}$$

In a neural net setting, f is an activation function such as ReLU and Θ^l is the matrix of parameters at layer l. GCN can be viewed as a spatial-based GNN because it updates the node embeddings by aggregating information from neighbors of nodes. In [183], a spectral based model is proposed which jointly learns relations between nodes and relations between attributes of nodes. The node embeddings in this model are the output of a 2D spectral graph convolution defined as $Z = GXF$. In this formula, X is a node feature matrix and G and F are an object graph convolutional filter and an attribute graph convolutional filter. The object graph convolutional filter is defined by designing a filter on the adjacency matrix of the graph. For defining the attribute graph convolutional filter, an attribute affinity graph is constructed on the original graph by applying either positive point-wise mutual information or word embedding based KNN on the attributes of the nodes. Directional Graph Networks (DGN) [108] defines directions for information propagation in the graph using vector

fields to improve the message passing in a specific direction in the current GNNs. In this method, the contribution of a neighbor node depends on its alignment with the receiving node's vector field. The vector fields denoted by B are defined using the k lowest frequency eigenvectors of the Laplacian matrix of the graph as they preserve the global structure of graphs [256]. The node representations are obtained by multiplication of the matrix B and the adjacency matrix of the input graph. In [257], it has been shown that most common GNNs perform l_2-based graph smoothing on the graph signal in the message passing, which leads to global smoothness. Motivated by the trend filtering idea [258], Elastic GNN [109] accounts for different smoothness levels for different regions of the graph using l_1-based graph smoothing. In [259], a framelet graph convolution is proposed. This method is based on graph framelets and their transforms [260]. Framelet convolution can lower the feature and structure noises in graph representation. This method decomposes the graph into low-pass and high-pass matrices and generates framelet coefficients. Then, the coefficients are compressed by shrinkage activation, which improves the network denoising properties. Simple Graph Convolution (SGC) [110] is a graph convolution network which simplifies the GCN model by removing the non-linear activation functions at consecutive layers. This study theoretically proves that this model corresponds to a fixed low-pass filtering in spectral domain in which similar nodes have similar embeddings. Many other studies introduce different variants of spectral-based GNNs [261, 262, 263, 264, 177, 178, 179, 164].

2) Spatial based. Spatial-based ConvGNNs define the graph convolution similar to applying CNN on images. Images can be viewed as a graph such that the nodes are the pixels and the edges are the proximity of pixels. When a convolution filter applies to an image, the weighted average of the pixel values of the central node and its neighbor nodes are computed. Similarly, the spatial-based graph convolutional filters generate a node representation by aggregating the node representations of neighbors of a node. One of the advantages of spatial-based ConvGNNs is that the learned parameters of models are based on close neighbors of nodes and therefore, can be applied on different graphs with some constraints. In contrast, spectral-based models learn filters that depend on eigenvalues of a graph Laplacian and are not directly applicable on graphs with different structures. GraphSAGE [107] is one of the early spatial based ConGNNs. This method generates node embeddings iteratively. The node embeddings are first initialized with node attributes. Then, a node embedding at iteration k is computed by concatenating the aggregation of

the node's neighbor and the node embedding at iteration $k-1$. For example, for a node v,

$$h^k_{N(v)} = Aggregate_k(h^{k-1}_u, \forall u \in N(v))$$
$$h^k_v = \sigma(W^k.Concat(h^{k-1}_v, h^k_{N(v)}))$$

where h^k and W^k are the node embedding and weight matrix at iteration k. $N(v)$ is the set of neighbors of v. GraphSAGE leverages mean, LSTM and pooling aggregators as follows:

- Mean aggregator. The mean aggregation is similar to GCN [106] which takes mean over neighbors of a node. The difference is that GCN includes the node representation h^{k-1}_v in the mean but GraphSAGE concatenates the node representation with the mean aggregation of neighbor nodes. This way, GraphSAGE avoids node information loss.

- LSTM aggregator. An LSTM aggregator aggregates neighbor nodes representations using an LSTM structure. It is important to note that LSTM preserves the order between nodes, however, there is no order among neighbor nodes. Therefore, GraphSAGE inputs a random permutation of nodes to alleviate this problem.

- Pooling aggregator. In this aggregation, each neighbor node is fed through a fully-connected neural net and then an elementwise max operation is applied on the transformed nodes as follows:

$$Aggregate^{pool} = \max\left(\{\sigma(W_{pool}h_u + b), \forall u \in N(v)\}\right) \tag{2.24}$$

  The above equation uses the max operator for pooling however mean operator can be used as well. The pooling aggregator is symmetric and learnable. The pooling aggregation intuition is that it captures different aspects of the neighborhood set of a node.

The aggregation continues until K iterations. The model is trained using a loss function that generates similar node embeddings for nearby nodes in an unsupervised setting. The unsupervised loss can be replaced with task specific objective functions. Graph Attention Network (GAT) [111] utilizes the self-attention mechanism [265] to generate node representations. Unlike GCN that assigns a fixed weight to neighbor

nodes, GAT learns a weight for a neighbor depending on the importance of the neighbor node. The state of node v at layer k is formulated as follows:

$$h_v^k = \sigma(\sum_{u \in N(v)} \alpha_{vu}^k W^k h_u^{k-1}) \tag{2.25}$$

$$\alpha_{vu} = \frac{exp(\text{LeakyReLU}(a^T[Wh_u||Wh_v]))}{\sum_{k \in N_v} exp(\text{LeakyReLU}(a^T[Wh_u||Wh_k]))} \tag{2.26}$$

$$h_v^0 = x_v \tag{2.27}$$

where α_{vu} is the attention coefficient of node v to its neighbor u defined using a softmax function. N_v is the neighbor set of node v. W is the weight matrix and a is a weight vector. $||$ is the concatenation symbol. In addition to self-attention, GAT's results benefit from using multi-head attention. Similar to GraphSAGE, GAT is trained in an end-to-end fashion and outputs node representations. Multi-hop Attention Graph Neural Network (MAGNA) [112] generalizes the attention mechanism in GAT [111] by increasing the receptive fields of nodes in every layer. Stacking multiple layers of GAT has the same effect, however that causes the oversmoothing problem. MAGNA first computes the 1-hop attention matrix for every node and then uses the sum of powers of the attention matrix to account for multi-hop neighbors in every layer. To lower the computation cost, an approximated value for the multi-hop neighbor attention is computed. MAGNA model aggregates the node features with attention values and passes the values through a feed forward neural network to generate the node embeddings. Message Passing Neural Net (MPNN) [113] proposes a general framework for ConvGNNs. In MPNN framework, each node sends messages based on its states and updates its states based on messages received from its immediate neighbors. The forward pass of MPNN has two parts: A message passing and a readout phase. In the message passing phase, a message function is utilized for information propagation and the node state is updated as follows:

$$h_v^t = U_t(h_v^{t-1}, \sum_{u \in N(v)} M_t(h_v^{t-1}, h_u^{t-1}, e_{vu})) \tag{2.28}$$

$$h_v^0 = x_v \tag{2.29}$$

where M_t is the message function and U_t updates the node representation. U_t, M_t are learnable functions. e_{vu} is the information of an edge (v, u). In the readout phase, the readout layer generates the graph embeddings using the updated node representations, $h_G = R(h_v^t | v \in G)$. Different ConvGNN methods can be formulated using this framework using different functions for U_t, M_t, R. GN block [114] proposes another general framework for Graph neural nets that some of the GNN

methods could fit in its description. A GN block learns nodes, edges and the graph representations denoted as h_i^l, e_{ij}^l, u^l respectively. Each GN block contains three update functions, ϕ and three aggregation functions, ρ:

$$e_{ij}^{l+1} = \phi^e(e_{ij}^l, h_i^l, h_j^l, u^l) \qquad m_i^{l+1} = \rho^{e\to v}(\{e_{ij}^{l+1}, \forall j \in N(i)\}) \tag{2.30}$$

$$h_i^{l+1} = \phi^v(m_i^{l+1}, h_i^l, u^l) \qquad m_V^{l+1} = \rho^{v\to u}(\{h_i^{l+1}, \forall i \in V\}) \tag{2.31}$$

$$u^{(l+1)} = \phi^u(m_E^{l+1}, m_V^{l+1}, h_i^l, u^l) \qquad m_E^{l+1} = \rho^{e\to u}(\{e_{ij}^{l+1}, \forall (i,j) \in E\}) \tag{2.32}$$

The GN assumption is that computation on a graph starts from an edge to a node and then to the entire graph. This phenomenon is formulated with update and aggregation functions: 1) ϕ^e updates the edge representations for each edge. 2) $\rho^{e\to v}$ aggregates the updated edge representations for the edges connected to each center node. 3) ϕ^v updates the node representations. 4) $\rho^{v\to u}$ aggregates node representation updates for all nodes. 5) $\rho^{e\to u}$ aggregates edge representation updates for all edges. 6) finally, the entire graph representation is updated by ϕ^u. GNN-FiLM [115] generates node embedding using the feature-wise linear modulation (FiLM) idea that was introduced in the visual question answering area [266]. Many common GNNs such as GCN [106] and GraphSAGE [107] propagate information along edges using information from the source node of the edges. In GNN-FiLM, the target node representation transformation is computed and applied to incoming messages to generate the feature-wise modulation of the incoming messages. Graph Random Neural Features (GRNF) [116] generates graph embeddings by preserving the metric structure of the graphs in the embedding space and therefore distinguishing between any pair of non-isomorphic graphs. This method is based on a family of graph neural feature maps. The graph neural feature maps are graph neural networks that can separate graphs. The outputs of these GNNs, which are scalar features, are then concatenated to generate the graph embedding. E(n) Equivariant Graph Neural Network (EGNN) [117] is a rotation, translation and permutation equivariant GNN. These properties are fundamental in representing structures that show rotation and translation symmetric characteristics, such as molecular structures [267]. EGNN takes as inputs a feature vector and an n-dimensional coordinates vector for each graph node along with the edge information and outputs the node embeddings. The main difference between this method and common GNNs is that the relative squared distance between a node's coordinates and neighbors has been considered in the GNN message passing operation. Bilinear Graph Neural Network (BGNN) [118] argues that the neighbors of a node can have interactions that may affect the node representations. Therefore, it augments the aggregation of neighbors of a node by pairwise interactions of neighbor nodes. Motivated by factorization machines [268]

, it models the neighbors' interaction using a bilinear aggregator denoted by BA, which computes the average of pairwise multiplication of neighbor nodes of a node. Then, the convolution operator is defined as follows:

$$H^{(k)} = (1-\alpha).AGG(H^{(k-1)}, A) + \alpha.BA(H^{(k-1)}, A) \tag{2.33}$$

where $H^{(k)}$ is the node representation at k-th layer and α is a tradeoff parameter between two components. In [269], it is theoretically shown that all attention-based GNNs fail in distinguishing between certain structures due to ignoring the cardinality information in aggregation. Therefore, this paper introduces two cardinality-preserved attention (CPA) models named Additive and Scaled. The formulation of the Additive model is as follows:

$$h_i^l = f^l(\sum_{j \in N(i)} \alpha_{ij}^{l-1} h_i^{l-1} + w^l \odot \sum_{j \in N(i)} h_i^{l-1}) \tag{2.34}$$

which the first term is the original attention formula and the second term captures the cardinality information. The Scaled model formula is:

$$h_i^l = f^l(\psi^l(|N(i)|) \odot \sum_{j \in N(i)} \alpha_{ij}^{l-1} h_i^{l-1}) \tag{2.35}$$

where $\psi(|N(i)|)$ is a function that maps the cardinality value to a non-zero vector. Both these models improve the distinguishing power of the original attention model. Multi-Channel graph neural network (MuchGNN) [119] generates graph representations using graph pooling operation. However, instead of shrinking the graph layer by layer using graph pooling which may result in loss of information, it shrinks the graph hierarchically. This method generates a series of graph channels at each layer and applies graph pooling on them to generate the graph representation at each layer. The final graph representation is the concatenation of graph representations at each layer. Policy-GNN [270] captures information for each node using different iterations of aggregations to capture the graph's structural information better. To that end, it uses meta-policy [271] trained by deep reinforcement learning to choose the number of aggregations per node. TinyGNN [120] proposes a small GNN with a short inference time. In order to capture the local structure of the graph, this method generates node representations by aggregating peer-aware representations of the node's neighbors. Peer-aware representations consider the interactions between peer nodes, which are neighbor nodes with the same distance from the center node. In addition, inspired by knowledge distillation [272], it proposes a neighbor distillation strategy (NDS) in a teacher-student network. The teacher network is a

regular GNN and has access to the entire neighborhood and the student network is a small GNN that imitates the teacher network. Other spatial based convolution GNNs include [273, 274, 275, 276, 277, 278, 279, 167, 280, 281, 182, 180, 175, 174].

2.3.3 Spatial-Temporal Graph Neural Net (STGNN)

Spatial-temporal GNNs are a category of GNN that capture both the spatial and temporal properties of a graph. They model the dynamics of graphs considering the dependency between connected nodes. There are wide applications for STGNNs such as traffic flow forecasting [227, 186, 225, 226, 197, 203], epidemic forecasting [224] and sleep stage classification [222]. For instance, in traffic prediction, the future traffic in a road is predicted considering the traffic congestion of its connected roads in previous times. Most of the STGNN methods fall into CNN based and RNN based categories which integrate the graph convolution in CNNs and RNNs, respectively.

RNN based. Graph Convolutional Recurrent Network (GCRN) [184] is an example of RNN based STGNN. In this method, an LSTM network is combined with the convolution operation. GCRN has two variants. In the first variant, a CNN layer is stacked with an LSTM layer. The CNN layer extracts the features at time t and the LSTM captures the temporal behavior of nodes over time. In the second variant, GCRN replaces the matrix multiplication operation in the LSTM with the graph convolution operation. In [227], the spatial and temporal correlations between nodes are modeled using three components including a spatial graph neural network layer, a GRU layer and a transformer layer. The input to the model is a sequence of graphs over time. The spatial graph neural network layer captures the spatial relations between nodes in each graph. Then, a GRU and transformer layers are applied on the sequence of graphs which are output from previous layer and capture the temporal relations between graphs over time. Other RNN-based spatial-temporal GNNs include methods in [282, 226, 223, 224, 187, 189, 193, 194, 196, 199, 200, 204, 206, 208, 209, 212, 217, 218, 219, 220, 223].

CNN based. Graph WaveNet [185] is a CNN based spatio-temporal GNN. This method takes as input a graph and feature matrices of nodes for m previous time steps and the goal is to predict the next n feature matrices. For example, in a traffic prediction application, a feature matrix is a $N \times d$ matrix that each row contains traffic features of a node. Each node is a sensor or a road. N is the number of nodes and d is the feature vector dimension. The framework of Graph WaveNet consists of two building blocks: a graph convolution layer and a temporal

convolution layer. In the graph convolution layer, it combines a diffusion convolution layer [223] with a self-adaptive adjacency matrix. The central part of the self-adaptive adjacency matrix is multiplying source and target node embeddings that are initialized randomly and are learned during the model training. The temporal convolution layer adopts a gated version of a dilated causal convolution network [283]. In Spatial-Temporal Fusion Graph Neural Networks (SFTGNN)[186], instead of modeling the spatial and temporal correlations of nodes separately, a spatial-temporal fusion graph is constructed using three $N \times N$ matrices to capture three kinds of correlation for each node. 1) A spatial graph for spatial neighbors, 2) A temporal graph for nodes with similar temporal sequences and 3) A temporal connectivity graph for connection of a node at nearby time points. The spatial-temporal fusion graph is then input to spatial-temporal fusion graph neural module (STFGN module) that generates node representations. In [225, 197, 203, 222, 188, 190, 191, 192, 195, 197, 198, 201, 202, 203, 205, 207, 210, 211, 213, 214, 215, 216, 221], many other CNN-based spatial-temporal GNNs are proposed.

2.3.4 Dynamic Graph Neural Net (DGNN)

Dynamic Graph Neural Nets (DGNN) are Graph neural nets that model a broad range of dynamic behaviors of a graph, including adding or deleting nodes and edges over time. EvolveGCN [229] is a dynamic GNN method. The idea behind it is to use an RNN model to update weights of the GCN at each time point and capture dynamics of the graph. At each time step t, a GCN layer is used to represent the graph at time t. In order to integrate historical information of the nodes, the initial weights for the GCN at time t are the hidden states/output of RNN based models which take as input the weights of previous GCN models. Each RNN based model is assigned to a separate layer of GCN models. The EvolveGCN model is trained end to end for link prediction and edge/node classification tasks. DyGNN [228] proposes a dynamic GNN method that consists of two components: Update and propagation components. These two components work in parallel to update and propagate the information of a new interaction in the graph. Let (v_s, v_g, t) represent a new directed edge that emerges between a source node v_s and a target node v_t at time t. The update component updates the two nodes v_s, v_t representations and the propagate component propagates the interaction information to *influenced nodes* which are defined as 1-hop neighbors of the two interacting nodes.

Dynamic Self-Attention Network (DySAT) [231] consists of two components, a structural block followed by a temporal block to capture the structural and temporal properties of a graph. This method defines dynamic graphs as a series of graphs

over time. In order to capture the structure of the graph at each time point, the structural block, which is a variant of GAT is applied to the graph at each time point. Then, to capture dynamic patterns of nodes, DySAT applies a temporal self attention layer in the temporal block. The inputs to the temporal block are node representations overtime for every node v such that the node representation x_v^t attends over the historical representation of the node ($< t$) to generate the final embedding of each node. Embedding via Historical Neighborhoods Aggregation (EHNA) [232] aggregates the historical neighbors of a node to capture the evolution of the node in the graph. In order to capture the historical neighbors of a node, EHNA first generates k temporal random walks for each node. The transition probability of each edge in a temporal random walk depends on the weight and time of the edge. The generated random walks for each node x are first aggregated using an LSTM model to generate the walk encoding. The sequence of k walk encodings are aggregated to generate a representation for the node x. Temporal Graph Network (TGN) [233] generates node embeddings for dynamic graphs that are modeled as a stream of edges. This model consists of several modules.

- *Memory*: At each time t, TGN saves a vector $s_i(t)$ for each node i in the memory to represent the node history in a compressed format. The vector $s_i(t)$ is initialized as a zero vector and updated as more edges emerge.
- *Message Function*: Every time an edge occurs between two nodes, a message is sent to the nodes participating in the edge.
- *Message Aggregator*: A node can be involved in multiple interactions. TGN keeps the last message in the order of time and the mean of other messages from other interactions for each node.
- *Memory Updater*: Every time an event occurs, the memory of the participating nodes is updated using a learnable memory update function such as LSTM.
- *Embedding*: The embedding module generates the temporal embeddings for each node at any time t using a learnable function h that updates the representation of each node even if the node was not involved in any interaction until that point.

In [284], a streaming GNN is modeled as $G^1, G^2, ..., G^T$ where $G^t = G^{t-1} + \delta G^t$ and δG^t is the changes of a graph between times $t-1$ and t. The loss of the network at time t is formulated as $L_{new} + L_{existing}$. L_{new} is the loss of parts of the graph influenced by the new changes at time t. $L_{existing}$ preserves information from previous time points. In L_{new}, the influenced nodes by new changes are replayed in

the GNN model. In $L_{existing}$, the important nodes from the history are replayed. In addition, the model parameters are approximated such that they do not deviate drastically from the model parameters at the previous time. Temporal Graph Attention Network (TGAT) [230] is a dynamic version of GAT [111]. GAT does not consider time ordering between neighbors of a node and is used for static settings. However, TGAT assumes ordering between neighbors of a node based on the time they arrive. The assumption is that a neighbor that occurs more recently is likely to have more influence on a node. In order to add time to the attention mechanism, a time vector is concatenated to the node feature vector. Time features that are used in TGAT are obtained based on the concepts from Bochner's Theorem and expressed by mapping the time to $\mathbb{R}^d$ is as follows:

$$\phi_d(t) = \sqrt{\frac{1}{d}}[cos(w_1 t), sin(w_1 t), ..., cos(w_{d/2} t), sin(w_{d/2} t)] \tag{2.36}$$

which parameters $w_1, ..., w_{d/2}$ are learnable parameter. Causal Anonymous Walks Neural Net (CAW-N) [10] generates temporal link embeddings by capturing the motif evolution in dynamic networks using a variant of anonymous walks [42]. This method predicts the probability of a link in the future and assumes that if motif structures of two nodes u, v interact over time, the probability of a link occurrence between u, v is higher. Therefore, this model defines set-based anonymization on the temporal random walks to capture the interaction between the motifs of the two nodes over time. Then, in order to obtain a representation of a link (u, v), all the anonymized walks for nodes u and v are encoded and aggregated using mean or attention mechanism and passed through an MLP to obtain the link probability. In Motif-preserving Temporal Shift Network (MTSN) [9], the dynamic network is considered as a series of graph snapshots over time and two components for generating node embeddings at each time point are introduced. The first component is a Motif Preserving Encoder (MPE) and the second is a Temporal shIft based on Motif preserving Encoder (TIME). MPE component preserves the high-order similarity between nodes at each snapshot. First, it generates node embeddings by running a simplified GCN on the adjacency matrix and a combined motif matrix of the graph separately and adding their outputs. The combined motif matrix of a graph is obtained by weighted averaging of motif matrices of the input graph that are computed using the Parametrized Graphlet Decomposition (PGD) technique [285]. TIME component considers the effect of time and is inspired by the Temporal Shift Mechanism in computer vision [286]. It shifts the node embeddings at each snapshot to capture the temporal evolution. In [234, 235, 236, 237], several other dynamic/temporal GNNs are presented.

Table 2.3: Some of the real-world applications of graph neural nets **deployed in production**

The applied Algorithm	Application
Standard GNN with Meta-Gradients [287]	Estimated time of travel (ETA) prediction in Google Maps
HetMatch [288]	Keyword matching for bid keyword recommendation in sponsered search platform of Alibaba Group
Category-aware GNN [289]	Review helpfulness prediction in Taobao
GNN based tag ranking (GraphTR) [290]	Video recommendation in WeChat Top Stories
DecGCN[291]	Online recommendation system in JD.com
DHGAT[292]	Search matching in shop search in Taobao
Dynamic Heterogeneous GNN[293]	Real-time event prediction in DiDi platform
Spatio-temporal graph neural net (ConSTGAT)[294]	Travel time estimation in Baidu Maps
Heterogeneous Graph Attention Maching Network (HGAMN)[295]	Retrieving point of interests in different languages in Baidu Maps
PinSage[296]	Recommendation system at Pinterest
Gemini[297]	Online recommendation at DidiChuXing
M2GRL[298]	Recommender system at Taobao

2.3.5 GNN-based method's real-world applications

Table 2.3 presents some of the real-world applications of graph neural nets which are deployed in production in several companies.

2.3.6 The limitation of GNNs and the proposed solutions

GNNs have several known limitations such as expressive power, oversmoothing and scalability. In this section, we summarize the major articles that address these issues.

Expressive power. The expressive power of a model refers to the model's ability to distinguish between different graphs. In other words, the model can map different graphs to different embeddings and similar graphs to similar embeddings. The expressive power of common GNNs such as GCN [106] and GraphSAGE [107] is bounded by the WL test and they fail to distinguish certain non-isomorphic substructures. A more powerful GNN-based embedding method called Graph Isomorphism Network (GIN) is proposed in [121]. GIN employs the sum operator instead of

the mean or max operators to aggregate the neighbors of a node using the following formula:

$$h_v^{(k)} = \text{MLP}^{(k)}((1 + \epsilon^{(k)})h_v^{(k-1)} + \sum_{u \in N(v)} h_u^{(k-1)}) \tag{2.37}$$

where ϵ can be a fixed or learnable parameter. GIN uses the sum aggregator as its expressive power is higher than the mean or max operators. For instance, assume we have two nodes v_1 and v_2. If v_1 has two equal neighbors and v_2 has three equal neighbors. The mean aggregator generates the same embedding for two nodes. The same applies to the max aggregator. However, the sum operator distinguishes different graph structures and generates different embeddings. It is also theoretically proven that GIN's expressive power is equal to the WL test. Identity-aware Graph Neural Networks (ID-GNN) [134] proposes a coloring mechanism as a solution for increasing the expressive power of GNNs. This model has two variants. The first variant has two components: inductive identity coloring and heterogeneous message passing. For computing the embedding for each node v, a k-hop ego networks of node v is extracted and the center node of the ego network is colored. Then, for each node in the ego network of node v, the embedding is computed using a different message passing component for each node based on color. This paper proposes a fast variant of ID-GNN that augments the node features instead of coloring nodes by injecting identity information such as cycle counts for cycles that start and end in the node v. Nested Graph Neural Net (NGNN) [133] suggests to encode a rooted subgraph instead of rooted subtree in common GNNs to generate node representations. It argues that rooted subtrees have limited expressiveness to represent non-tree graphs. In [127, 176], distance-based features are added to each node to increase the expressive power of GNNs, where distance vectors for each node are computed with respect to each center node. Table 2.4 summarizes the major studies related to increasing the expressive power of GNNs.

Oversmoothing. A critical known issue with GNNs is their depth limitation [302, 303]. GNN methods aggregate information from one-hop neighbors of a node in the first layer. The second layer reaches the two-hop neighbors of a node and stacking additional layers goes forward in the neighbors of a node similarly. After passing through multiple layers, the generated nodes vectors will be *oversmoothed* because the local information for each node is lost. Graph Random Neural Network (GRAND) [141] proposes a new framework to address the oversmoothing problem. This method augments the feature matrix of the input graph using DropNode, which randomly drops features of some of the nodes, similar to the DropEdge mechanism [139]. This augmentation helps in making nodes less sensitive to their neighbor

Table 2.4: A summary of major solutions proposed to increase the expressive power of GNNs

Authors	Algorithm	Brief summary of the solution
Xu et al. [121]	GIN	Aggregating neighbors using the sum operator
Murphy et al [122]	RP-GNN	Adding a unique node label
Morris et al. [123]	k-GNN	Performing message passing between subgraph structures instead of the node level
Maron et al. [124]	PPGN	Considering higher-order message passing
Chen et al. [125]	Ring-GNN	Using a ring of matrices in addition and multiplication
Azizian et al. [126]	FGNN	Augmenting the model with matrix multiplication
Li et al. [127]	DEGNN	Adding an extra node feature based on distance encoding
Balcilar et al. [128]	GNNML	Designing the convolution in spectral domain and masking it with a large receptive field
Barcelo et al. [299]	-	Adding local graph parameter to GNNs
Sato et al. [129]	rGIN	Adding random features to GIN [121]
Papp et al. [130]	DropGNN	Dropping some of the nodes randomly
Wang et al. [131]	PEG	Using separate channels to update the node and positional features
Wijesinghe et al. [132]	GraphSNN	Injecting structural characteristics in the message passing
Zhang et al. [133]	NGNN	Encoding a rooted subgraph for each node instead of a rooted subtree
You et al. [134]	ID-GNN	Inductively injecting node identities either using coloring mechanism or an augmented node feature
Dasoulas et al. [135]	CLIP	Using colors to distinguish similar node attributes
Huang et al. [300]	PG-GNN	Capturing correlation between neighboring nodes using a permutation-aware aggregation
Wijesinghe et al. [301]	GraphSNN	Designing a local isomorphism hierarchy for nodes neighborhood subgraphs

nodes. Then, it aggregates neighbors of a node up to K-hop away using mixed-order propagation, which lowers the risk of oversmoothing. The mixed-order propagation formula is $X = \sum_{k=0}^{K} \frac{1}{K+1} \hat{A}^k \bar{X}$, where $\bar{X}$ is the augmented feature matrix. The model is trained using consistency regularization [304] also to reduce the overfitting issue in the semi-supervised setting in the case of scare labels. In [142], it has been theoretically proved that adding two simple techniques to GCN at each layer can overcome oversmoothing. First, constructing a connection to input features to ensure at least a fraction of node features reach the final node representation. Second, adding the identity matrix to the weight matrix to enforce at least the same performance as a shallow GCN. The formulation of this deep GCN is as follows:

$$H^{(l+1)} = \sigma(((1-\alpha_l)\hat{A}H^{(l)} + \alpha_l H^{(0)})((1-\beta_l)I_n + \beta_l W^{(l)})) \tag{2.38}$$

where α_l and β_l are the hyperparameters. $H^{(0)}$ are the initial node representations which are the node features. I_n is the identity matrix. Table 2.5 shows the formulation of some of the major methods that are introduced for alleviating the oversmoothing problem in GNNs.

Scalability. Another bottleneck of GNNs is scalability. In GNNs, the representations of a node's neighbors are aggregated to generate the node embeddings. Specifically, for a GNN with L layers, the neighbor aggregation computed by the matrix multiplication AH^l has the time complexity $O(LmF)$, where m is the number of neighbors and F is the hidden dimension of model. The number of neighbors can be large in very large graphs, which lowers GNNs training speed and increases the memory consumption. In [107, 150, 154], this problem is alleviated by sampling a subset of node neighbors. In [147], neighbors of a node are sampled at each layer independently. Cluster-GCN [146] reduces the memory problem of GCN by sampling a subgraph for each batch using clustering techniques and applying a graph convolution filter on the nodes in the subgraph. Some methods such as SGC [110] remove the non-linear activation function to reduce the training time. RevGNN [152] is based on the reversible connections [308] and lowers the memory consumption of GNNs with respect to the number of layers. In this model, the feature matrix is divided into several groups, which are then input into the model to generate a group of outputs. The advantage of this model is that only the output of the last input is saved in memory for backpropagation. In [155], a unified GNN sparsification (UGS) framework is proposed that jointly simplifies the graph and the model to lower the GNN's inference time. The loss function of UGS is:

$$L_{UGS} = L(\{m_g \odot A, X\}, m_\theta \odot \Theta) + \gamma_1 |m_g| + \gamma_2 |m_{\theta|} \tag{2.39}$$

Table 2.5: A summary of major solutions proposed to alleviate oversmoothing of GNNs. Z is the node prediction label, H is the node representation, K_l is the convolution of the l-th layer, s is a projection vector, G_o is the discrete gradient operator on the graph, $\hat{A}_{drop}$ is the symmetric normalized adjacency matrix with certain number of edges dropped, INFLATION(.,e) = Normal(Power(Softmax(.),e)), f_l is the number of features at layer l, S^l is the clustering assignment matrix, C is number of groups, μ_i, σ_i are the mean and standard deviation of group i. $\gamma_i, \beta_i, \lambda$ are hyper parameters. TPSD is a total pairwise squared distance measure, X_{drop} is the perturbed feature matrix, p is an step size, G_v is an extracted subgraph for a node v. Φ_l is a learnable parameter, $\tilde{L}$ is symmetric normalized Laplacian matrix. *psi* is a learnable function, X is the feature matrix. T is time step.

Algorithm	Formula
GCN [106]	$H^{(l+1)} = \sigma(\hat{A}H^lW^l)$
APPNP[136]	$H = \sigma(\hat{A}H^lW^l)$, $Z^{l+1} = (1-\alpha)\hat{A}Z^l + \alpha H$
JKNet-Concat [137]	$h_v = \text{concat}(h_v^1, ..., h_v^L)W$
GCN-PN [138]	$H^{(l+1)} = \text{TPSD}(\sigma(\hat{A}H^lW^l))$
DropEdge [139]	$H^{(l+1)} = \sigma(\hat{A}_{drop}H^lW^l)$
SGC [110]	$H = \hat{A}^LXW$
DGN-GNN[140]	$H^{(l+1)} = S^{(l+1)}H^{(l+1)} + \lambda\sum_{i=1}^{C}\gamma_i((H_i^{(l+1)} - \mu_i)/\sigma_i) + \beta_i$
DAGNN[305]	$H = \text{concat}(mlp(X), \hat{A}^1mlp(X), ..., \hat{A}^kmlp(X))), \quad Z = \text{softmax}(\sigma(Hs)H)$
GRAND [141]	$Z = mlp(1/(L+1)\sum_{i=0}^{L}\hat{A}^iX_{drop})$
GCNII [142]	$H^{(l+1)} = \sigma(((1-\alpha_l)\hat{A}H^{(l)} + \alpha_lH^{(0)})((1-\beta_l)I_n + \beta_lW^{(l)}))$
GDC [143]	$H^{(l+1)} = \sigma(\sum_{i=1}^{f_l}\hat{A}_{drop}[:,i]H^l[:,i]W^l[i,:])$
PDE-GCN [144]	$h^{(l+1)} = h^l - pG^TK_l^T\sigma(K_lG_oh^l)$
GRAND-PDE [306]	$H = \psi(X(0) + \int_0^T \frac{\partial X(t)}{\partial t}dt)$
SHADOW-SAGE [145]	$h_v^{(l+1)} = \sigma(W^l.\text{concat}(h_v^l, \text{aggregate}(h_u^l, \forall u \in G_v))$
GCN+inflation [307]	$H^{(l+1)} = \text{INFLATION}^{(l+1)}(\sigma(\hat{A}H^lW^l), e)$
AdaGNN [181]	$H^{(l+1)} = H^{(l)} - \tilde{L}H^{(l)}\Phi_l$

Table 2.6: Time and memory complexities of GCN and some of the proposed scalable GNN models. L is the number of layers, F is the hidden dimension of the model, n is the number of nodes, m is the number of neighbors, b is the batch size, d is the average degree of the graph, s_n is the number of sampled nodes per node,s_l is the number of sampled nodes per layer, δ is the ratio of blocked nodes, $\tilde{s}_n = s_n \times (1-\delta)$, m_g is the number of remaining edges, m_θ is the number of remaining connections in the model, k is number of codewords .

Method	Solution	Time complexity	Memory complexity
GCN [106]	-	$O(LmF + LnF^2)$	$O(LnF)$
GraphSAGE [107]	Neighbor sampling	$O(ns_n^L F + ns_n^{L-1}F^2)$	$O(s_n^L bF)$
SGC [110]	Linear model	$O(nF^2)$	$O(bF)$
ClusterGCN [146]	Graph Sampling	$O(LmF + LnF^2)$	$O(LbF)$
FastGCN [147]	Layer sampling	$O(Lns_l F + LnF^2)$	$O(Ls_l bF)$
LADIES [148]	Layer sampling	$O(Lns_l F + LnF^2)$	$O(Ls_l bF)$
GraphSAINT [149]	Graph sampling	$O(LbdF + LnF^2)$	$O(LbF)$
VR-GCN [150]	Graph sampling	$O(LmF + LnF^2 + s_n^L nF^2)$	$O(LnF)$
GBP [151]	Linear model	$O(LnF^2)$	$O(bF)$
RevGNN [152]	Reversible connections	$O(LmF + LnF^2)$	$O(nF)$
VQ-GNN [153]	Vector quantization	$O(LbdF + LnF^2 + LnkF)$	$O(LbF + LkF)$
BNS [154]	Neighbor sampling	$O(\tilde{s}_n^{L-1}.(s_n bF + (\delta/(1-\delta) + 1).bF^2))$	$O(\tilde{s}_n^{L-1} s_n bF)$
GLT [155]	Graph Sampling and model pruning	$O(Lm_g F + Lm_\theta nF^2)$	$O(LnF)$

where m_g, m_θ are masks for the unimportant connections in the graph and weights in GNNs. L is the cross-entropy loss and γ_1, γ_2 are hyperparameters to control the regularization of m_g, m_θ. After the training, these two masks prune the adjacency matrix and model parameters. Several other papers that studied the scalability of GNNs are [153, 151, 309, 310, 311, 312, 313, 314, 315]. Table 2.6 provides the complexity of GCN and some of the proposed approaches for lowering its complexity.

Capturing long-range dependencies in graphs. GNNs struggle to capture long-range dependencies between nodes in the graph. The reason is that broadening the GNNs receptive field by increasing their depth encounters the oversmoothing problem in node representations. To solve this problem, in [316] a transformer module is combined with the standard GNN to capture the long-range relationships. Efficient infinite-depth graph neural net (EIGNN) [317] has implicit infinite layers

which can capture very long dependencies. The output of EIGNN before the softmax layer is a limit of an infinite sequence of convolutions:

$$f(X, F, B) = B(\lim_{H \to \infty} Z^{(H)}) \tag{2.40}$$

where $Z^{(H)} = \gamma g(F) Z^{(H-1)} S + X$ is the output of the H-th layer. F, B are learnable weight parameters, S is the normalized adjacency matrix of the input graph and $\gamma \in (0, 1]$. The $g(F)$ function is defined as $\frac{1}{|F^T F| + \epsilon_F} F^T F$ which is constrained to be less than one. As a result, the infinite sequence of convolutions is convergent. A closed-form solution is derived for EIGNN instead of using iterative solvers. In [318], the depth-wise and breadth-wise propagations are considered to capture long-range dependencies in graphs. The breadth-wise propagation is between the representation of a node with its neighbors' representations from the previous layer, which is a form of horizontal skip-connections. This model leverages a residual message function to consider edge features and alleviate the breadth-wise gradient diminishing in the model's backpropagation.

Catastrophic Forgetting. Catastrophic forgetting means that the neural network model may forget previously learned knowledge when trained on a new task. In [319], a topology-aware weight preserving module (TWP) is proposed to alleviate this issue in GNNs. This module measures the importance of the GNN's parameters after learning each task. Then, it enforces the model to remember the old parameters when learning a new task by penalizing changes to the important parameters with respect to the old tasks. In [320], an experience replay based model (ER-GNN) selects some nodes from previous tasks and replays them when learning new tasks. The replayed nodes are those whose features are the closest to the mean of features in each class and have the maximum coverage and influence in model training.

Homophily assumption. GNNs are based on the homophily [321] assumption which means that nodes that are connected are similar and have the same class labels. However, this is not true in networks that nodes with opposite characteristics connect to each other. In [156], three design principles from previous methods are combined to make the new model suitable for both homophily and heterophily settings.

1. Separating node embeddings and neighbor embeddings because mixing the node and neighbor information results in similar embeddings among a neighborhood.

2. Considering higher order Neighborhoods to capture more relevant information from more neighbors.

3. Combining the intermediate representations to increase the range of neighbors and information considered in node representations.

Similarly, it has been shown in [158] that the disassortativity of many real-world graphs can lead to the low performance of GNN models on these graphs. Therefore, this paper proposes to generate a computation graph from the original graph and then run the GNN on the computation graph. The computation graph is a multi-relational graph with different types of edges between two nodes based on different levels of neighborhood similarities such as nodes degrees and neighboring nodes degrees. In [160], a class compatibility matrix is added into GNNs to improve the performance of GNNs in graphs that heterophily exist. This framework first estimates a prior belief of every node's class label based on the node features. Then, using a compatibility matrix H, the prior beliefs of nodes are propagated in their neighborhood. Each element H_{ij} in the matrix H is the empirical probability that nodes with class label i connects to nodes with class label j in the dataset. The compatibility matrix which can be learned in this model enables it to go beyond the homophily assumption. Table 2.7 summarizes some of the major papers that studied the homophily assumption in GNNs.

Neglecting substructures. It has been shown in [171] that the expressive power of message passing GNNs in detecting subgraphs of three or more nodes is limited. Therefore, a Local relation pooling (LPR) model is proposed based on egonets. An egonet centered at a node is a subgraph consisting of nodes within a certain distance from the node in the graph. The LPR's idea is that a pattern in a graph can be found in the egonet of some node. Therefore, it generates a node representation by aggregating transformed egonets centered at the node. Graph Structural Kernel Network (GSKN) [172] accounts for graph substructures such as cliques or motifs by leveraging anonymous walk-based graph kernels (AWGK). Graph kernels are similarity measures for pairwise graph comparison. This method defines an anonymous walk kernel and a random walk kernel to capture structural information in the graph. These kernels are defined based on the definition of l-walk kernels, which compute the similarity between two graphs by comparing all length l walks between every node in two graphs. Then, the GSKN formulation is derived using the kernel mapping of these two kernels. In message passing simplicial networks (MPSN) [169], the message passing is performed on simplicial complexes [323] which are a form of subgraphs consisting of several simplices. For instance, 0-simplex is a node,

Table 2.7: A summary of major solutions proposed to make GNNs suitable for both hemophily and heterophily

Author	Algorithm	Brief summary of the solution
Zhu et al. [156]	H2GCN	Combining three designs principles: 1) separation of ego and neighbor sampling, 2) higher order neighbors, 3) combining intermediate representations.
Chien et al. [157]	GPR-GNN	Propagating node hidden features using generalized pagerank methods
Suresh et al. [158]	WRGNN	Generating a computation graph based on nodes' structural equivalences
Yang et al. [159]	DMP	Setting an attribute propagation weight for each edge
Zhu et al. [160]	CPGNN	Adding a compatibility matrix
Jin et al. [161]	U-GCN	Extracting three types of node embeddings from 1-hop, 2-hop and k-nearest neighbors
Liu et al. [162]	NLGNN	Employing an attention-guided sorting of neighbor nodes
Yang et al.[163]	GPNN	Leveraging a pointer network to rank neighbor nodes
Wang et al. [164]	HOG-GCN	Incorporating a learnable homophily degree matrix into a GCN
Fang et al. [165]	Polar-GNN	Using dissimilarities between nodes in the aggregation by introducing attitude polarization
Du et al. [166]	GBK-GNN	Utilizing two kernels to capture the homophily and heterophily and selecting the appropriate one for each node pair
Li et al. [322]	GloGNN	Finding global homophily for nodes showing heterophily by learning a coefficient matrix

Table 2.8: The formulas for the major solutions proposed to help GNNs capture substructures. i is a neighborhood , r is a relationship, $\tau(z_v, z_u)$ is a function that defines a relationship from node v to node u in a latent space, x_v^V are the combined structural features of node v, $e_{u,v}$ are the edge (u, v) features, h_p is the embedding of a simplex p, $m_B(p), m_C(p), m_{\downarrow}(p), m_{\uparrow}(p)$ are the aggregation of messages from the boundary, co-boundary, lower and upper adjacent simplices of the simplex p, R_v is a row in a node-topic matrix representing the probabilities of a node v in a graph belonging to the graph's structural topics, $S_{n_{v,t}}^{k-BFS}$ is the set of permutation of subset of nodes in egonet of node v of depth t compatible with k-truncated breadth first search, α_π is a learnable normalization factor for π, f can be an MLP layer, $B_{v,t}^{[ego]}$ is the tensor representation of the egonet of node v, ϕ is an anonymous walk,$R(\phi)$ is the concatenation of the attributes of ϕ, $Z = [R(\phi_i)]_i$, Φ_l is the set of anonymous walks of length l, $h_{x,c}$ is the representation of a subgraph c, γ is a learnable similarity measure, A_x is a subgraph at channel x, a_x is the representation of A_x.

Algorithm	Formula
Geom-GCN [167]	$h_v^{l+1} = \sigma(W_l.\text{aggregate}_{i,r}((e_{(i,r)}^{v,l+1}, (i,r)))), e_{(i,r)}^{v,l+1} = \text{aggregate}(\{h_u^l \vert u \in N_i(v), \tau(z_v, z_u) = r\})$
GSN-v [168]	$h_v^{l+1} = \sigma(h_v^l, m_v^{l+1}), m_v^{l+1} = \text{aggregate}(\{h_v^l, h_u^l, x_v^V, x_u^V, e_{u,v}\})$
MPSN [169]	$h_p^{l+1} = \text{aggregate}(h_p^l, m_B^l(p), m_C^l(p), m_{\downarrow}^l(p), m_{\uparrow}^l(p))$
GraphSTONE [170]	$h_v^{l+1} = \text{aggregate}(\{\frac{R_v^T R_u}{\sum_u R_v^T R_u} h_u^l, u \in N(v)\})$
DeepLPR [171]	$H_v^{l+1} = \frac{1}{\vert S_{n_{v,t}}^{k-BFS} \vert} \sum_{\pi \in S_{n_{v,t}}^{k-BFS}} \alpha_\pi^l \odot f^l(\pi * B_{v,t}^{[ego]}(H^l))$
GSKN [172]	$h_v = \sum_{\phi \in \Phi^l(G,v)} \sigma(Z^T Z)^{-1/2} \sigma(Z^T R(\phi))$
SUBGNN [173]	$h_{x,c}^{l+1} = \sigma(W.[g_{x,c}^l; h_{x,c}^l]), g_{x,c}^l = \text{aggregate}(\{\gamma(c, A_x).a_x, \forall A_x\})$

1-simplex is an edge and 2-simplex is a triangle. The representation of each simplex is computed by aggregating four types of messages received from its adjacent simplices which are present in the graph, including boundary, co-boundary, lower and upper adjacencies. For example, an edge's boundary simplices are its vertices and the co-boundary simplices of a node are its connected edges. Finally, the global embedding for a simplicial complex is computed using a readout function on the representation of its simplices. In [167, 168, 170, 173, 324], several other GNNs with the focus on the subgraphs are presented. Table 2.8 demonstrates the formulas for some of these methods.

Over-squashing. One of the bottlenecks of GNNs is over-squashing when capturing long-range dependencies. The over-squashing is different from the oversmoothing because the oversmoothing occurs when the graph learning task only needs short-range dependencies and stacking more layers in GNN makes the node embeddings

indistinguishable. However, when long-range dependencies are required and more layers are added to GNNs, the information from distant neighbors is compressed in a fixed-length vector, resulting in over-squashing and low performance of GNNs [325]. A simple solution for this problem is presented in [325] where a fully-adjacent layer is added as the last layer to a GNN model. In this layer, every two nodes are connected which helps consider nodes beyond the nodes' local neighbors in their representation. In [326], it is proved that negatively curved edges are the cause of over-squashing. A negative curvature occurs when an edge becomes a bridge between two sides of the graph where removing the edge disconnects them. Therefore, a curvature based graph rewiring model is proposed to solve the over-squashing issue. This rewiring approach works by adding extra edges to support the most negatively curved edges and removing the most positively curved edges. Furthermore, the graph edit distance between the original graph and the modified one is bounded to ensure a local graph modification.

Design space for GNNs. Designing effective architectures for GNNs in different tasks and datasets requires manual labor and domain knowledge. Therefore, several methods were proposed to automatically design suitable architecture for the given datasets and downstream tasks. Graph neural architecture search method (GraphNAS) [327] allows the automated architecture design using reinforcement learning. In GraphNAS, an RNN generates a GNN architecture that will be trained and validated on the training and validation sets. The validation result is the reward of the RNN. The RNN samples the design of each layer from a search space of different operators such as neighbor sampling, message computation, message aggregation and readout operators. For example, a sample GNN layer is:

$$[\text{first-order, gat, sum, 4, 8, relu}] \tag{2.41}$$

where each element in the list corresponds to neighbor sampling, message computation, message aggregation operators, number of heads, number of hidden layers and the activation function. In [328], a general design space, task space and evaluation method for GNNs are proposed to identify the best GNN architecture for the given task quickly. The design space consists of intra-layer design, inter-layer design and learning configuration. The intra-layer design in each layer of GNN consists of batch normalization, dropout, activation function and aggregation function. The inter-layer design between GNN layers has four dimensions: layer connectivity, pre-process layers, message passing layers and post-processing layers. The training configuration concerns the batch size, learning rate, optimizer and training epochs.

Then, a task similarity metric is introduced that can identify similar tasks. Finally, it develops a controlled random search evaluation to quickly find the best GNN design for the given task among many model-task combinations. In [329, 330], other methods related to the architecture design of GNNs are introduced.

2.4 Interpretability of node embeddings

Little work has been done towards the interpretation of node embeddings. Dalmia et al. [331] evaluate each node embedding method based on the accuracy by which it can preserve different properties of graphs. To do that, first, a set of network elementary property prediction tasks are defined and used to compare the performances of the node representation methods on each of these tasks. Then, node embeddings are applied in downstream tasks. Finally, the paper correlates the performance of the model on downstream tasks with the node properties that the model can best encode.

Another study in [332] provides an interpretability score for each embedding dimension across node groups and for each node group across embedding dimension adapted from the work by Senel et al. [12] on word embeddings. Then, it evaluates the effects of changing hyperparameters of node embeddings methods (number of embedding dimensions) on the performance of the embedding method in downstream tasks. However, the results were only provided on one embedding method (i.e. Node2vec) and the paper does not provide a comprehensive framework for scoring and comparing interpretability of various node embedding methods.

Several works have studied the interpretability of word embeddings including algorithms based on the word intrusion test [24, 25, 13]. Luo et al. [13] first selected top words for each dimension according to the descending order of its values. Then, a word is randomly selected from the lower values of the dimension which has high values in other dimensions. Then, the embedding dimension interpretability is evaluated based on how well human can identify the intruder word (i.e., the word with the low value) in that dimension given the list of selected words. This is an effective but expensive method due to human involvement. In [333], the matrix rotation algorithm was applied to word embedding vectors to improve the interpretability. Then, interpretability of a dimension is computed according to the distance between the intruder and the non-intruder words.

For interpreting the sentence representations, Adi et al. [334] proposed using the accuracy of a prediction task to measure how well the sentence representation encodes

the structural properties of a sentence. A classifier is trained using representation vectors to predict a specific sentence aspect (e.g. the length of a sentence). In another work [12], an interpretable embedding space was introduced according to distribution statistics of category words in word embedding vectors. Then, the interpretability score was computed based on the number of common words between top words and words belong to a category. For each dimension i of word embeddings, the maximum of interpretability scores for all categories is considered as the interpretability score of the dimension i. The overall interpretability score is defined as the average of interpretability scores for all dimensions.

Chapter 3

Dynnode2vec: Scalable Dynamic Network Embedding

3.1 Introduction

In the last few decades, graph embedding methods have achieved remarkable success in the analysis of large networks. The basic aim is to represent nodes of a large, high-dimensional graph in low-dimensional vectors that preserve the neighbourhood information of the graph. Several static graph embedding algorithms [34, 38, 30, 32, 40, 335, 238, 41] have been developed for a variety of machine learning tasks such as visualization [336], node classification [337], link prediction [247], and recommendation [338]. Random walk and edge sampling graph embedding approaches [38, 30, 32] have a remarkable performance in large networks which contain more than thousand nodes and edges.

Large real-world networks evolve naturally over time, i.e., nodes and edges appear or disappear, or edges change. For example, in co-authorship networks new edges may emerge (new collaborations may be formed) or new nodes can be added (new authors) and in social networks, users may delete friends (delete edges) or some users may leave the network (delete nodes). The static graph embedding methods are not capable of dealing with the critical challenge involved in dynamic networks. The disadvantage of using static embedding methods at each timestamp independently are as follows. First, embedding vectors for each timestamp are in different spaces. Second, learning embedding vectors separately is a time-consuming process.

In this work, we propose *dynnode2vec*, a scalable dynamic network embedding for large evolving networks. In order to handle dynamic networks, *dynnode2vec* modifies the well-known static embedding method, *node2vec* by employing the previous

learned embedding vectors as initials weights for the skip-gram model. This is motivated by dynamic language models, especially dynamic Skip-gram models [14, 15]. In addition, we utilizes evolving random walks for updating the trained skip-gram from previous timestamp. The evloving random walks are only generated for nodes that have changed in consecutive times. As random walk generation is the time consuming part of graph embeddings, we are able to significantly reduce the running time. Our main contributions in this study are:

- We develop a dynamic embedding method *dynnode2vec* that captures evolving patterns in large dynamic networks
- *dynnode2vec* is a fast and accurate method for dynamic graph embedding
- We evaluate the performance of our method in variety of tasks including link prediction, node classification and anomaly detection on large real-world graphs

3.2 *Dynnode2vec*: Scalable dynamic network embedding

We represent a dynamic network G as a sequence of graphs $G_1, G_2, \ldots, G_T$ from timestamps 1 to T. Each graph at time t is represented by $G_t = (V_t, E_t)$ where V_t and E_t are the vertices and edges of the graph respectively. Our goal is to modify the static *node2vec* embedding method for learning representation of a dynamic network. We begin with a brief description of the static *node2vec* embedding method. Then, we explain the steps of *dynnode2vec* which are summarized in Algorithm 4. *node2vec* is an extension of *deepwalk* algorithm [32] and has two main subcomponents; *node2vec* random walk and Skip-gram model. *node2vec* random walk is a flexible random walk method which samples neighbourhoods of a source node by Breadth-first Search (BFS) and Depth-first Search (DFS). *node2vec* first generates a corpus by sampling a number of random walks γ of length t starting at each vertex. Then, the Skip-gram uses these random walks to learn the representation vector for each node.

3.2.1 Description of *dynnode2vec* steps

The main challenge of modification of the static *node2vec* is how to learn embedding at time t by updating embedding vectors in time step $t-1$. For a dynamic network

$G = G_1, G_2, \ldots, G_T$, we run the static *node2vec* for the first graph G_1 separately, extract the embedding vectors and keep the structure of trained Skip-gram for the next timestamp. For all other subsequent timestamps $2, \ldots, T$, the following steps are performed between two consecutive timestamps t and $t+1$.

Evolving Walk generation. In the static *node2vec*, random walks are generated independently for each timestamp for all nodes which is very time-consuming process. In *dynnode2vec*, we generate an effective set of random walks for only evolving nodes instead of generating random walks for all nodes in the current timestamp. Therefore, new random walks from changed regions in the graph G_t can efficiently update the embedding vectors according to temporal evolution of networks over time. Assume the structure evolution of the graph G_t from G_{t-1} includes sets of new edges/nodes that are added (E_{add}/V_{add}), deleted (E_{del}/V_{del}) and their weights have changed (E_{change}). The evolving nodes in the timestamp t are defined as follows:

$$\Delta V_t = V_{add} \cup \{v_i \in V_t | \exists e_i = (v_i, v_j) \in (E_{add} \cup E_{del})\} \tag{3.1}$$

Generating evolving random walks is fast and time-efficient since dynamic networks are evolved gradually and neighbourhoods of most nodes are kept unchanged. However, in the worst case, evolving nodes include all nodes as all nodes have changed in the timestamp t.

Dynamic Skip-gram model. In natural language processing domain, several dynamic word embedding [339, 340, 14, 15] have been proposed to track word evolution such as new words are created (internet), and some words 'die out' throughout time. In [15], for learning the representation vectors in the timestamp t they train the Skip-gram by initializing word vectors obtained from previous timestamp $t-1$. In this dynamic Skip-gram model, the vocabulary set is updated and the Skip-gram is retrained by new documents in timestamp t. In network embedding domains, DynGEM [341] incrementally learns embedding vectors for the timestamp t by using embedding vectors from the timestamp $t-1$ as initial weights for SDNE atuoencoder in a dynamic network.

dynnode2vec also takes advantage of dynamic Skip-gram model for obtaining embedding vectors at time t and uses the pre-trained Skip-gram model (Skip-gram$_{t-1}$) as initial weight for (Skip-gram$_t$). In order to do that, first the vocabulary set of Skip-gram$_t$ is updated according to new evolving walks. Then, Skip-gram$_t$ is trained by new evolving walks generated on the evolving nodes.

Algorithm 1 :**Algorithm: Dynnode2vec**

Input: Graphs $G = G_1, G_2, \ldots, G_T$
Output: Embedding vectors $Z_1, Z_2, \ldots, Z_T$
Run static *node2vec* for the Graph G_1
for $t = 2$ to N **do**
    Find a set of evolving nodes,ΔV_t,
    Sample new random walks ($Walk_n$) for ΔV_t
    Train Skip-Gram $Skip_t$ with $Walk_n$ and obtain Z_t
end for

3.3 Experiments

3.3.1 Datasets

The performance of *dynnode2vec* is evaluated on following datasets.

- Hep-th [342]: This dataset is the coauthorship network of researchers in High energy physics theory conference. Hep-th has $34k$ nodes, $421k$ edges divided into 60 graphs.

- Autonomous Systems (AS) [342]: AS is built from logs of the BGP (Border Gateway Protocol) which shows the communication between users. Number of nodes, edges and time steps are $6k$, $13k$ and 100 respectively.

- Enron [343]: Enron is the email communication network between Enron company employees. It has $87k$ nodes and $1.1M$ edges over 175 months.

- StackOverflow (St-Ov) [344]: This dataset is derived from question and answers in Math Overflow website. Each edge shows users interactions. This dataset contains $14k$ nodes and $195k$ edges over 2350 days. We divided the datastream into 58 graphs.

- dblp [345]: This is the coauthorship network dataset among researchers of different fields. It consists of $90k$ nodes and $749k$ edges over 18 years. Each node in dblp has one of the two class labels, database and data mining (VLDB, SIGMOD, PODS, ICDE, EDBT, SIGKDD, ICDM, DASFAA, SSDBM, CIKM, PAKDD, PKDD, SDM and DEXA) and computer vision and pattern recognition (CVPR, ICCV, ICIP, ICPR, ECCV, ICME and ACM-MM).

- Darpa [346]: Darpa is a dataset consisting of communications between source IPs and destination IPs. This dataset contains different attacks between IPs. We used a subset of darpa consisting of $12k$ nodes and $22k$ edges over 100 hours.

3.3.2 Baselines

We compared the performance of our method against the following existing methods:

- DeepWalk[32]: This is the first node embedding method based on random walks. DeepWalk uses Skip-gram model and uniform random walks to learn the neighborhood structure of the graph.
- *node2vec*[30]: This method learns node representation in networks by preserving network neighborhood of nodes. It explores the neighborhood of a node by generating DFS and BFS random walks for that node.
- DynGEM [341]: This approach is a stable dynamic node embedding method that works for growing dynamic graphs. It incrementally builds the embedding vectors at each time using the embedding vectors from previous time. In dynamic networks with a large number of nodes like Enron and DBLP, the memory requirement of Dyngem signicantly increases. Therefore, we are only able to run it on three datasets; AS, Hep-th, and St-Ov.

3.3.3 Link Prediction

Link prediction is an important application of graph embeddings. In this task, future edges are predicted given the previous edges. We consider the link prediction as a classification task similar to [30]. For example, if we have a sequence of graphs $G_0, G_1, ..., G_t$, we predict edges at time t using edges from time 0 to $t-1$. For instance, edges of G_1 are predicted using the positive and negative edges of G_0. For G_2, we use edges from G_0 and G_1. We do the same for all the graphs at future time points.

Edge embedding. The embedding of an edge (u, v) can be computed using embedding vectors of nodes u and v. In the literature, different operators are applied on node embedding vectors to compute edge embeddings including Weighted-L1, Weighted-L2, Hadamard and average as defined in [30].

We report the link prediction results for all the four aforementioned operators in Table 6.1. The results are the average AUC (Area Under Curve) with a grid search over $p, q \in \{0.5, 1, 2, 4\}$. We evaluated all methods on four datasets: AS, Hep-th, Enron and St-Ov. The results show that *dynnode2vec* outperforms baselines in almost all datasets. Among different operators, *dynnode2vec* has the best performance with Hadamard operator and the worst performance with average operator.

Table 3.1: Link prediction results using four operators a) Weighted-L1 b) Weighted-L2 c) Hadamard d) Average

Op	**Algorithm**	**Dataset**			
		AS	Hep-th	Enron	St-Ov
A	DeepWalk	0.957	0.9887	0.8489	0.5595
	node2vec	0.9554	0.9893	0.8533	0.5617
	DynGEM	0.7855	0.6246	-	0.6242
	dynnode2vec	**0.9625**	**0.996**	**0.8922**	**0.66**
B	DeepWalk	0.9577	0.9882	0.805	0.5561
	node2vec	0.9561	0.9894	0.8649	0.5608
	DynGEM	0.7701	0.6161	-	0.6246
	dynnode2vec	**0.9635**	**0.9977**	**0.8981**	**0.6698**
C	DeepWalk	0.888	0.9749	0.8167	0.7456
	node2vec	0.8935	0.9762	0.8236	0.7478
	DynGEM	0.8005	0.5882	-	0.6387
	dynnode2vec	**0.9512**	**0.9975**	**0.9012**	**0.8886**
D	DeepWalk	0.6197	0.5571	0.5271	0.5182
	node2vec	0.6258	0.5577	0.5083	0.5149
	DynGEM	**0.7949**	0.555	-	0.6175
	dynnode2vec	0.7718	**0.6269**	**0.6147**	**0.6824**
node2vec settings (p,q)		(0.5,1)	(0.5,1)	(0.5,1)	(1,2)

3.3.4 Node Classification

Another application of node embeddings is in node classification. In supervised classification, nodes have class labels in the dataset. We used logistic regression as the classification method. Similar to link prediction, node embeddings of previous time points are used in predicting class labels of future time points. We compared the results of our method with baselines on dblp dataset. The Micro-F_1 and Macro-F_1 results are reported in Table 4.2. Our result are better than other baselines in terms of Micro-F_1 scores.

3.3.5 Anomaly Detection

Anomalies are any deviations from normal behavior. There are different categories of anomaly detection methods in dynamic settings including detecting anomalous nodes, edges and change detection. Inspired by DynGEM, we used *dynnode2vec* to detect changes in the dynamic network. We computed the norm of the differences between embedding vectors of common nodes at consecutive time points. High values for the norm signals a significant change in the structure of the graph. We applied *dynnode2vec* and node2vec on a subset of Darpa dataset[346]. Figure 4.4

Table 3.2: Node classification results

Metric	**Algorithm**	**Dataset**
		dblp
Micro-F1	DeepWalk	0.5337
	node2vec	0.5272
	dynnode2vec	**0.5415**
Macro-F1	DeepWalk	**0.4141**
	node2vec	0.3847
	dynnode2vec	0.4056
node2vec settings (p,q)		(1,2)

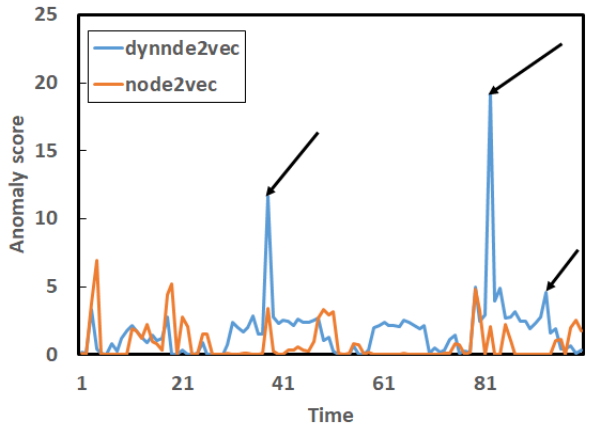


Figure 3.1: Anomaly detection results

shows that there are three major peaks in the dynamic node2vec curve. These spikes correspond to time points that three important attacks occurred in the dataset.

3.3.6 Effects of evolving walk generation

In this work, we only generate walks for the nodes in the graph that have changed compared to previous time point. This leads to a significant speedup in *dynnod2vec* running time. In order to show the time efficiency of *dynnod2vec*, we compared the running time of *dynnod2vec* with two methods: node2vec and *dynnod2vec* version denoted by *dynnod2vec-all* that samples walks for all the nodes. The comparison results on three datasets: AS (the first fifty time steps), Hep-th and Enron (the first forty time steps) are shown in Table 4.3. *dynnod2vec* was faster than other methods. All experiments are performed on a windows X-64 with 7 cores, 64 GB RAM and a clock speed of 3.6 GHz.

Additionally in terms of accuracy, we compared the AUC scores of *dynnod2vec* with

Table 3.3: Running time comparison

	AS	Hep-th
node2vec	23.22 min	14.42 min
dynnode2vec-all	23.10 min	13.52 min
dynnode2vec	4.49 min	1.25 min

Table 3.4: Comparison of dynnode2vec vs dynnode2vec-all

OP	**Algorithm**	**Dataset**			
		AS	Hep-th	Enron	St-Ov
a	dynnode2vec-all	0.9799	0.9973	0.9114	0.6674
	dynnode2vec	0.9625	0.996	0.8922	0.66
b	dynnode2vec-all	0.9803	0.9976	0.9042	0.6734
	dynnode2vec	0.9635	0.9977	0.8981	0.6698
c	dynnode2vec-all	0.9099	0.9966	0.9022	0.8825
	dynnode2vec	0.9512	0.9975	0.9012	0.8886
d	dynnode2vec-all	0.6488	0.5982	0.6364	0.6798
	dynnode2vec	0.7718	0.6269	0.6147	0.6824
node2vec setting (p,q)		(0.5,1)	(0.5,1)	(0.5,1)	(1,2)

dynnod2vec-all. Table 6.6 indicates the AUC scores in link prediction task. In most cases the AUC for these two methods are not significantly different.

3.4 Summary

In this work, we propose *dynnode2vec*, a scalable dynamic network embedding that learns representation vectors for dynamic networks. *dynnode2vec* employs the dynamic Skip-gram model and evolving random walks to discover information changes in temporal networks. In the dynamic Skip-gram model, the previous learned embedding vectors are transferred to the next timestamp as initial weights. This results in smooth embedding vectors for graphs over times. Furthermore, the evolving random walks are generated to efficiently reflect the changes in dynamic graph structure. By only considering subset of random walks, *dynnode2vec* can obtain embedding vectors in notably less time without sacrificing accuracy. Our experiments demonstrate the superiority of *dynnode2vec* as compared with the state-of-the-art embedding methods in various tasks. Future work will investigate using other dynamic Skip-gram models [339, 14] for dynamic graph embedding.

Chapter 4

Dynamic Graph Embedding via LSTM History Tracking

4.1 Introduction

Graphs are powerful tools to represent many of the real world data such as social networks, protein-protein networks, traffic data and scientific collaborations. For instance, in social networks, nodes of the graph are people and connections between them are the edges of the graph. Graphs can be either static or dynamic. In static graphs, the structure of graph is fixed but in dynamic graphs nodes and edges get added and deleted over time.

Many graph mining algorithms deal with static graphs. With the growing number of dynamic networks in the real world, there is a tremendous need for developing efficient algorithms that work properly in dynamic settings. Furthermore, the size of a network increases rapidly over time and makes it challenging to have a proper representation of the entire data. This creates the need for effective algorithms that map networks into a low dimensional space so that they can be utilized in downstream machine learning applications.

A widely used tool to represent graphs are adjacency matrices. The problem with adjacency matrices is that they are memory intensive for very large graphs. Various algorithms were developed to lower the dimensions of these matrices while preserving the necessary information. These algorithms include matrix factorization methods [33, 347, 34, 348] and linear or non-linear dimensionality reduction techniques such as Principal Component Analysis (PCA) [349]. In general, these methods are computationally expensive for large graphs and fail to perform well in many graph mining tasks.

A recent alternative approach to learning node embeddings is deep learning techniques designed for static graphs. The state-of-the-art static methods are DeepWalk [32], LINE[38], node2vec [30] and Variational graph autoencoder [41]. DeepWalk[32] and Node2vec [30] take advantage of random walk for each node to preserve the neighborhood structure of the graph. Compared to other embedding methods that use adjacency matrices as input, random walk based methods capture more than immediate neighborhood of nodes. As a result, they can produce more accurate representative vectors for nodes. Furthermore, training the neural networks with random walks are significantly faster than adjacency matrices based methods.

However, it is still a challenging problem to apply deep node embedding methods for dynamic graphs. Dynamic graphs can change with different rates over time and we need mechanisms to reflect temporal changes in the node embeddings. There have been some recent efforts in this direction that resulted in several algorithms [341, 5, 6, 350, 41, 6]. DynGEM [341] initializes its model with embeddings from previous time points to stabilize the overall embeddings in consecutive time points. The difficulty with this method is that it works with adjacency matrices which makes it an memory intensive method. Dynnode2vec [6] initializes the node2vec model at the current timestamp with the previous timestamp's embeddings and trains the model using random walks, which capture the neighborhood structure in the graph better than adjacency matrices. However, this method only generates random walks for changed nodes between snapshots which may partially reflect the changes in the graph structure in some cases. The method in [5] generates dynamic embeddings for each node by defining temporal walks for the nodes and utilizes them in training a skipgram model [242], which does not consider the order of the elements in the temporal walks. However, it is essential to avoid losing the order of time dependencies among sequence elements. On the other hand, LSTM proved effective in word and sentence representations in natural language processing. It is specifically useful in preserving long term dependencies between elements in a sequence [16, 17, 18].

In this work, we propose LSTM-Node2vec for computing node embeddings in dynamic graphs. Our key contribution is using an autoencoder LSTM for keeping the history of nodes and training it with a special kind of temporal random walks that capture the evolving patterns in the structure of the graphs. In our method, the embeddings obtained from history are used as initial weights for a node2vec model. Afterwards, node2vec considers the local information from the current graph and produces an embedding that is the combination of both temporal and static information for the nodes of the graph. In addition, for aligning the node embeddings

over time, the weights of the model at previous time points is passed along to the model for the next graphs.

In our experiments, we evaluate the performance of LSTM-Node2vec in 11 anomaly detection, link prediction and node classification on datasets from various domains. Our approach outperforms existing methods in majority of the cases. Overall, our contributions can be summarized as follows:

- We propose LSTM-node2vec, a novel dynamic embedding method that captures temporal changes with LSTM and then the learned parameters are transferred into node2vec to incorporate the local structure of each graph.
- We train an autoencoder LSTM model with temporal walks to capture the history of nodes over time which is the first study to consider temporal walks as the input to an LSTM model.
- We evaluate our method on three main data mining tasks including anomaly detection, link prediction and node classification.

The chapter is organized as follows. We first present the technical details of our method and show the experiments results. Then, in the conclusion section we summarize the findings in this work.

4.2 Problem Statement

A graph stream is represented by $G_1, G_2, ..., G_t$ where $G_i = (V_i, E_i)$ is the graph at time i and V_i and E_i are the nodes and edges of the graph. Given a graph stream, our goal is to compute the dynamic representation vectors for the nodes of graphs at each time point. The embedding of a node v into a d-dimensional vector can be formulated as a mapping function $f : v \rightarrow R^d$.

In order to generate an embedding vector for each node in a graph, we propose LSTM-Node2vec that combines both dynamic and static states of a node to generate a more accurate representation of the node.

4.3 Dynamic Network Embedding Method

4.3.1 Overview of LSTM-Node2vec

LSTM-Node2vec generates the node embeddings for graph G_i in the graph stream using three main steps: (1) generating temporal random walks over nodes from a

sequence of graphs before G_i, (2) training an autoencoder LSTM with the temporal neighbor walks to learn node embedding for G_i, and (3) passing the embeddings as initial weights to a node2vec model. Algorithm 4 presents the steps of LSTM-Node2vec. Below, we will describe the three steps in detail.

Algorithm 2 :**LSTM-Node2vec**

Input: Graphs $G_{t-(L-1)}, G_{t-(L-2)}, ..., G_t$, where L is the temporal window size and t is the current time point.
Output: Z_t: embedding vectors for all the nodes in G_t
for $i = 1$ to $|V_{G_t}|$ **do**
    Generate k temporal random walks of maximum length L for node v_i of G_t, and add them to set R
end for
Initialize the input layer weights of an LSTM encoder with Z_{t-1}
Train the LSTM autoencoder with node sequences in set R
Initialize node2vec with the input layer weights W_i of the trained LSTM encoder
Train a skipgram model with random walks on G_t
Return the weights in the input layer of the skipgram model as Z_t

4.3.2 Temporal neighbor walk generation.

Inspired by [5] and [85], we define a temporal random walk as follows. Given a series of graphs $G_1, G_2, ..., G_t$ a temporal neighbor walk for each node v in graph G_i is defined as a sequence of neighbors of a node v at L previous time points represented by $u_{i-L}, u_{i-L+1}, ..., u_i$ where u_x is the neighbor of node v at time x.

We sample a neighbor for node v at each of L previous time points based on the alias sampling method used in node2vec [30] for random walk generation. We utilized the same approach (described in Section E below) to sample a neighbor node for node v in each time point. The temporal walk is the concatenation of neighbors of node v in the increasing order of time. By defining the temporal random walk this way, we track the changes of neighbors of a node at each time point and as a result the random walk reflects the changes in the structure of the graph over time. For instance, in the coauthorship temporal network, the temporal walk shows the changes in the collaboration of a particular author over time. Figure 6.2 illustrate the temporal walk generation in a temporal network for window size $L = 3$.

The window size that we consider for generating temporal walk is L. The length of the temporal walk is allowed to have a value between $[2, L]$. 2 is the minimum required length to define a valid sequence and L is the maximum window size. As mentioned in [5], it is possible that a node does not occur in several time points in

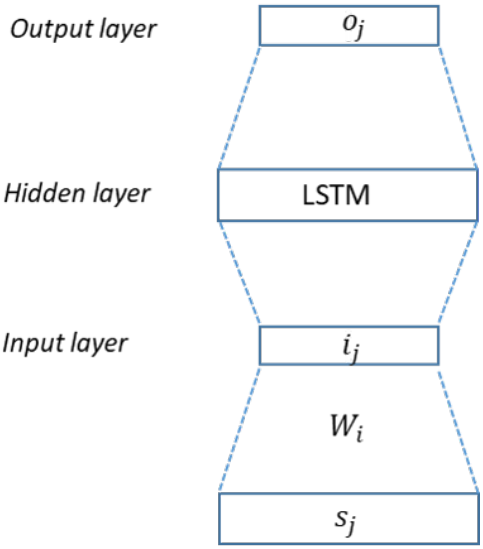


Figure 4.1: LSTM training for word representation. s_j is the jth word in a sentence which is represented by a one-hot encoding vector. W_i is the weights of input layer in LSTM.

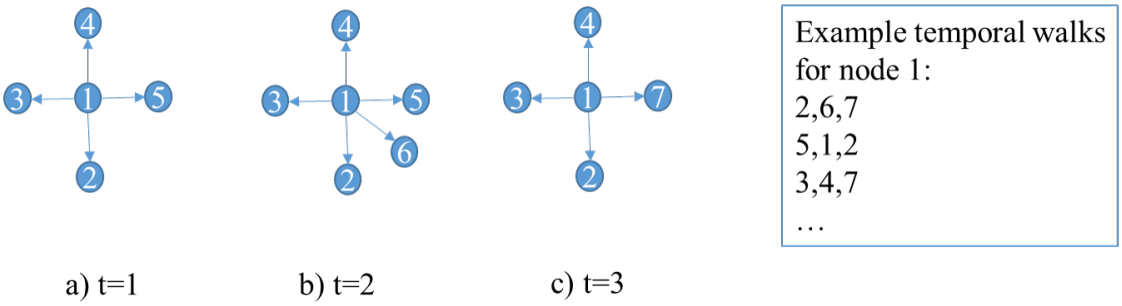


Figure 4.2: The process of temporal walk generation. a, b, c are three time points of a temporal network. Given $L = 3$ the temporal walks for node 1 is generated by sampling from its neighborhood nodes.

the L previous time points. As a result, depending of how frequent a node is in the window, the length of its temporal walk ranges between 2 and L. In addition, we generate k temporal walks per node in each graph. Therefore, the total number of generated temporal walks is a multiple of the total number of nodes in the graph.

4.3.3 LSTM Autoencoder

An LSTM is a recurrent neural network (RNN) used for capturing long term order dependencies between elements in sequences. LSTMs are widely used in many areas of machine learning, specially natural language processing (NLP), for problems such as text translation and question answering [351, 352, 353, 354, 355, 356, 357]. A

common usage of LSTM in NLP is to generate word representations [16, 17, 18]. Given a set of sentences, an LSTM autoencoder is trained to take a sentence as input and outputs the same sentence. Given a sentence $S = \{s_1, s_2, ..., s_n\}$ of length n where s_i is the ith word in the sentence, an LSTM autoencoder takes the words one by one using the one-hot word representation and is trained to produce the same word in the output layer. After the training, the weights in the input layers are the word representation for s_i is obtained from learned weights W_i in LSTM. Figure 4.1 depicts the LSTM training for word representations.

Inspired by the LSTM effectiveness in NLP, we use LSTM to learn node representations in graphs. Here, a temporal walk over nodes at different time points is considered as a sentence. The order in the temporal walk reflects the way that the neighborhood of a node evolves over time. By training an LSTM over a set of temporal random walks, LSTM can capture the temporal dependencies between the neighbors of nodes in the graph evolution. Let R denote a set of all temporal walks obtained for nodes of graph G_t, $R = \{r_1, r_2, ..., r_m\}$. $r_i = u_1, u_2, ..., u_L$ is a temporal walk generated for each node in the graph. $m = N \times k$ is the total number of walks when there is N nodes and k temporal walks are generated for each node. We used each r_i in R for training LSTM autoencoder such that r_i is the input and output of the model. The formula of LSTM in the encoder part of the model is as follows:

$$
\begin{aligned}
i_t &= \sigma(W_i[h_{t-1}, u_t] + b_i) \\
f_t &= \sigma(W_f[h_{t-1}, u_t] + b_f) \\
o_t &= \sigma(W_o[h_{t-1}, u_t] + b_o) \\
\tilde{C}_t &= tanh(W_c[h_{t-1}, u_t] + b_c) \\
C_t &= f_t \times C_{t-1} + i_t \times \tilde{C}_t \\
h_t &= o_t \times tanh(C_t)
\end{aligned}
$$

In these equations, f_t, i_t, o_t are forget gate, input gate and output gate. h_{t-1} is the output of LSTM at previous time point. C_t is the cell state vector which is updated in two parts. First forget gate f_t decides which part of cell state in previous time C_{t-1} will be discarded and then the new values will be stored in cell state. The output of LSTM in the current time point h_t will be a filtered version of cell state. b_x, W_x are biases and weights for respective gates. u_t is each node in the temporal walk and W_i are the node representations.

We run LSTM-Node2vec for every graph in the stream to get the node embeddings of the graph. It is crucial that the generated embeddings for consecutive graphs be in the same vector space. In order to do that, the LSTM autoenceder for the first

graph G_0 is randomly initialized. Then, we initialize the LSTM autoencoder of G_i with the weights of the previous graph G_{i-1} model.

4.3.4 Node2vec

Node2vec [30] is the state-of-the-art node representation method for static graphs. The node2vec algorithm is built on the concepts that were first introduced in DeepWalk algorithm. Both of these methods are categorized as random walk based low dimensional node representation algorithms. Node2vec algorithm consists of two parts: random walk generation and skipgram [242]. The main idea of node2vec is to generate second order random walks for all nodes in the graph such that it effectively explore neighborhood of the node by interpolating between DFS and BFS search strategies. In order to create that effect, node2vec defines a biased random walk for each node using two parameters p and q that control the random walk exploration procedure. Specifically, p is a return parameter which can change the probability that the walk returns to an already visited node. q is the inward and outward exploration parameter. Large values of q guide the walk toward exploration of nodes that are close to the target which is similar to how BFS works and small values of q encourage outward exploration similar to DFS. Formally the unnormalized transition probability between node v and node u is defined as follows:

$$\alpha_{pq}(v,u) = \begin{cases} 1/p, & \text{if } d_{vu} = 0 \\ 1, & \text{if } d_{vu} = 1 \\ 1/q, & \text{if } d_{vu} = 2 \end{cases} \tag{4.1}$$

where d_{vu} is the shortest path distance between nodes v and u.

In the next step of node2vec, the random walks generated in previous step are used to train a skipgram architecture. Skipgram model learns the continuous representations of each node. The objective function of node2vec is a maximum likelihood optimization problem that maximizes the probability of preserving neighborhood of a node in a d-dimensional space. Here is the objective function formula:

$$max \sum_{u \in V} logPr(N_s(u)|f(u)) \tag{4.2}$$

where u is the target node in the graph $G = (V, E)$. $N_s(u)$ and $f(u)$ are the sample neighborhood of node u generated using random walks and the embedding vector of u, respectively. In the context of Natural Language Processing, neighborhood of a word in a sentence is defined as a window of words close to the target word in the

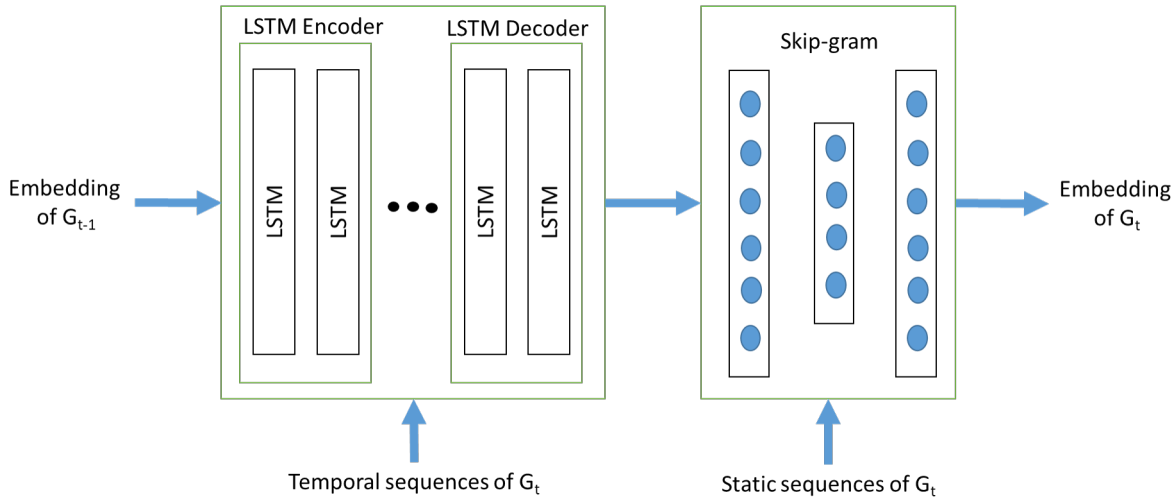


Figure 4.3: Framework for LSTM-Node2vec for creating embedding of G_t using both temporal and static sequences

sentence. This concept is transferred to graphs using random walks that depending on the definition sample different neighbors of a node.

We initialize the embedding layer of the node2vec skipgram model with the learned weights in the input layer of LSTM autoencoder for each graph at time t. This way, the final node representations in node2vec for graph G_t, which are the output of LSTM-Node2vec, are the combination of both static and dynamic states of the graph.

4.4 Experiments

We applied the generated node representation vectors in different data mining tasks in different datasets. The results of the experiments are given below.

4.4.1 Baselines

The evaluation of LSTM-Node2vec was performed against following the state-of-art methods for static and dynamic node representations.

- **DeepWalk [32]:** DeepWalk is a static network embedding method based on random walks that utilize uniform random walks.
- **Node2vec [30]:** This method is a static network representation algorithm consisting of BFS/DFS like random walks and skipgram.

- **dyngraph2vecAE [4]:** It is a dynamic network embedding method based on dyngraph2vec. It utilizes deep learning models with multiple fully connected layers to model the interconnection of nodes.

- **dyngraph2vecAERNN [4]:** This is another variant of the dyngraph2vec method which is a dynamic representation learning method. It feeds previously learned representations to LSTMs to generates embedding vectors.

4.4.2 Experiment settings

In all the experiments the embedding vector size is 128. The parameter L in LSTM-Node2vec is selected depending on the size of each dataset. For Radoslaw, Ubuntu, Contact, St-Ov, AS, Dblp and Acm, the length of L equals 10, 10, 10, 20, 20, 5 and 5, respectively. The LSTM-Node2vec model is trained with the Adam optimizer. We run Node2vec with $(p, q) = (0.25, 1)$ and DeepWalk with $(p, q) = (1, 1)$.

4.4.3 Anomaly Detection

Anomaly detection is an important data mining and graph mining application. Anomalies are any deviation from normal behavior. Identifying these irregular patterns in the data is the task of anomaly detection methods [358]. Anomaly detection is widely applied in static and dynamic network mining tasks such as network intrusion detection, bank frauds and social networks [359, 360, 361, 362]. Comprehensive description of anomaly detection methods in graphs can be found in two surveys [358, 363]. We evaluated the performance of LSTM-Node2vec in anomaly detection task and compared the results with those of other static and dynamic embedding methods.

Datasets: We run our experiments on the following three datasets:

- Radoslaw[364]: This is a dataset of email communications between employees of a company with 167 nodes, 89K edges over 39 time points.

- Ubuntu[364]: Ubuntu is a network of interactions on the Ask Ubuntu website. Interaction between users include answering/commenting on other users questions/comments. This temporal dataset consists of 137K nodes, 280K edges and has 79 time steps.

- Contact[364]: This dataset is a network of human contacts with 274 nodes and 28.2K edges. We divided the dataset into 39 graphs.

Table 4.1: AUC results of anomaly detection

Algorithm	Radoslaw	Ubuntu	Contact
DeepWalk	0.72095	0.53984	0.67082
node2vec	0.715568	0.674564	0.64213
dyngraph2vecAE	0.431883	0.5787	0.732996
dyngraph2vecAERNN	0.477085	0.5965	**0.76226**
LSTM-Node2vec	**0.895336**	**0.703182**	0.682581

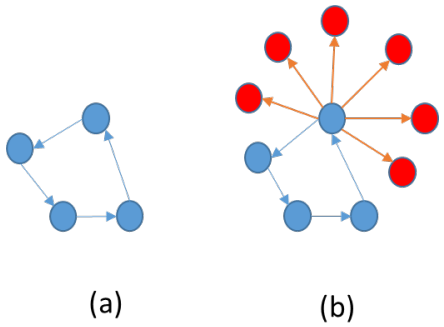


Figure 4.4: A temporal network at two time steps. (a) the graph at time $t-1$. (b) the graph at time t with an star shaped anomaly, one previously normal node starts attacking multiple nodes shown in red.

Injecting anomalies: Because of the challenges in finding datasets that have ground truth labels in anomaly detection, we directly injected anomalies into normal datasets [363, 99]. In network intrusion detection, one of the important type of anomalies are bursts in activity. For instance, bursts can happen when a particular node in the network starts attacking other nodes. This creates a star shaped dense subgraph in the graph structure.

In this work, we inject star shaped anomalies into three datasets. For creating star anomalies, we add a large number of new edges between a target node and n other existing nodes in the dataset that did not have an edge with the target node previously. This shows a sudden burst of activity from the target node toward other nodes. We create multiple star shaped anomalies and then inject them into dynamic graphs in each dataset. In order to do that, we inject one anomaly in k consecutive graphs. For instance, for $k = 3$ in graph streams $G_1, G_2, ..., G_{10}$ we inject anomalies in G_3, G_4, G_5. Then we jump m graphs and start injecting the second anomaly and continue similarly on the entire graphs in the dataset.

Classification: We use the anomalous datasets generated in previous step for anomaly detection tasks. Here, we formulate the anomaly detection as a classification task. Consider a graph stream of $G_1, G_2, ..., G_t$. First, the node representation vector of all the nodes in each graph in the stream is computed. In our

Table 4.2: Anomaly detection AUC scores for three graphs to analyze the effect of previously seen anomalies

	G_1	G_2	G_3
Radoslaw	0.88	0.90	0.91
Ubuntu	0.64	0.89	0.97
Contact	0.76	0.95	0.99

datasets, edges have anomalous or normal class labels because each edge represents an attack from a source node to a destination node. As a result, we create edge embedding for all the edges in the stream. There are multiple known ways to create edge embeddings from node embedding vectors including Hadamard, l_1, l_2 and average [30]. In our experiments, we used l_1 operator meaning that for an edge (u, v), $f(u, v) = f(u) - f(v)$, where $f(x)$ is the embedding of x.

After creating edge embeddings for each graph, we perform classification on graph G_i using edge embeddings of graphs in $G_0, G_1, ..., G_{i-1}$ for i from L to t. The reported value for each dataset is the average AUC score of classification task over the entire time points using the Random Forest classifier. The results in Table 4.1 show that LSTM-Node2vec outperforms other embedding methods in Radoslaw and Ubuntu datasets and the average AUC score in Contact is less than two other methods. This can be due to the fact that the changes in the structure of graphs in Radoslaw and Ubuntu are rather smoother than Contacts and the history is still relevant in the current time. Therefore, we can say that LSTM-Node2vec works well when the changes of graphs are not significant over time.

Detecting previously seen anomalies: It is expected that the anomaly detection accuracy increases if the anomalies have previously occurred in the dataset. We tested this on three datasets. As explained in previous sections, one anomaly is injected into $k = 3$ consecutive time points and the AUC score of anomaly detection in each of anomalous graphs is computed. The results are shown in Table 4.2. We first introduced the anomaly in graph G_1. As a result, it has the lowest anomaly detection score. For G_2 and G_3, however, as the anomaly has existed in the dataset, the accuracy starts to increase.

4.4.4 Node Classification

Node classification is to classify a node in a graph into a predefined category. Similar to any classification task, part of the dataset is considered as a training set and the class labels for the test set are predicted. An approach for node classification is

Table 4.3: Macro-f1 and Micro-f1 scores for node classification

Metric	Algorithm	Dblp	Acm
Macro-f1	DeepWalk	0.34	0.3458
	node2vec	0.3333	0.3696
	dyngraph2vecAE	0.3299	0.334
	dyngraph2vecAERNN	0.3682	0.3969
	LSTM-Node2vec	**0.4658**	**0.4605**
Micro-f1	DeepWalk	0.4632	0.4908
	node2vec	0.4624	**0.5291**
	dyngraph2vecAE	0.4523	0.4656
	dyngraph2vecAERNN	0.4645	0.4962
	LSTM-Node2vec	**0.501**	0.4995

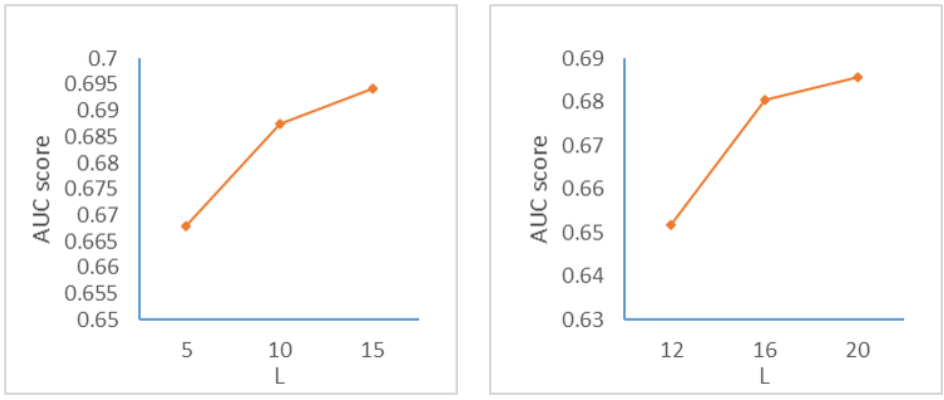
presented in [341] in the dynamic graph setting. Based on this approach, we classified each graph at time t using the previous graph at $t-1$ by applying the logistic regression method. We used two measures Macro-f1 and Micro-f1 for evaluating our method. The results are reported in Table 4.3. The datasets used are as follows:

- Dblp[365, 345]: Dblp is the network of coauthorship between researchers from 2000 to 2017 with 90k nodes and 749k edges. Nodes in this dataset are the researchers that belong to either database/data mining class (VLDB, SIGMOD, PODS, ICDE, EDBT, SIGKDD, ICDM, DASFAA, SSDBM, CIKM, PAKDD, PKDD, SDM and DEXA) or computer vision/pattern recognition class (CVPR, ICCV, ICIP, ICPR, ECCV, ICME and ACM-MM).
- Acm[365, 345]: The Acm dataset is similar to Dblp and spans from 2000 to 2015.

As it is evident in Table 4.3, LSTM-Node2vec outperforms other methods in the Dblp dataset in terms of both Macro-f1 and Micro-f1. In Acm, our performance gain is above other methods in terms of Marcro-f1 and it is comparable with other results in terms of Micro-f1. LSTM-Node2vec is remarkable in the node classification task. This is reasonable as a class label in Acm and Dblp datasets is an area of research of authors which does not drastically change over time. Thus, including history in the embedding computation helps in giving a better picture of the author in the current time.

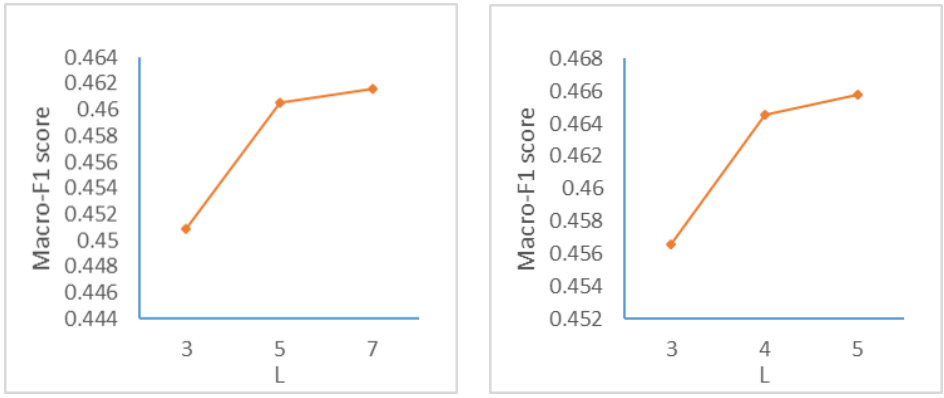
4.4.5 Link Prediction

One of the main graph mining tasks is link prediction. Link prediction can be formulated as a classification task such that each edge either has a positive or negative



(a) Radoslaw (b) St-Ov

Figure 4.5: Analysis of effect of parameter L on link prediction for Radoslaw and St-Ov datasets



(a) Acm (b) Dblp

Figure 4.6: Analysis of effect of parameter L on node classification for Acm and Dblp datasets

class label. Then the edges in the test set are classified using a model trained with the training set. For this purpose, we predict edges in graph G_t using previous graphs from 0 to $t-1$ in the graph stream, based on a method proposed in [341]. Our classifier is logistic regression. Table 6.6 summarizes the results in terms of the average AUC score for three datasets. This evaluation is performed on the following three dataset.

- St-Ov[342]: This is the user interaction network in the Math Overflow website. St-Ov contains 14k nodes and 195k edges with 58 time steps.
- Radoslaw[364]: Explained in the anomaly detection section.

Table 4.4: AUC scores for link prediction

Algorithm	St-Ov	Radoslaw	AS
DeepWalk	0.5976	0.6516	0.8858
node2vec	0.6038	0.6738	0.8577
dyngraph2vecAE	0.5098	0.5233	0.7193
dyngraph2vecAERNN	0.5372	0.5612	0.7085
LSTM-Node2vec	**0.6857**	**0.6875**	**0.8878**

- AS[342]: AS is the communication network of who-talk-to-whom gathered from logs of Border Gateway Protocol. The numbers of nodes, edges and time steps are 6k, 13k and 100 respectively.

Table 6.6 shows that LSTM-Node2vec outperforms other methods in terms of AUC scores on all the datasets. Node2vec also has a good performance compared to other methods. LSTM-Node2vec improves that even more by including the temporal information of nodes.

4.4.6 Effect of length of history parameter L

We analyze the influence of length of L in the link prediction and node classification tasks. We computed the scores using different lengths of L depending on the datasets. For instance, as Dblp consists of 18 graphs, we compute the results for $L = 3, 4, 5$. The same is true for the Acm dataset. We use longer L values for Radoslaw and specially St-Ov as they are bigger datasets. The results of this analysis are given in Figures 4.5 and 4.6. They show that increasing the length of history has a positive effects in the results. However, based on our observations, there is a limit to this increase which is due to the fact that by including very far history in the current state of a graph, we may consider information that are not relevant to present time.

4.4.7 Effects of changes in model structure

Evaluating LSTM-Node2vec for all three tasks was performed using one LSTM layer in the encoder and one LSTM layer in the decoder. In order to analyze the effect of adding complexity to the model, we computed the results with an additional LSTM layer in the encoder and reported the difference in performance in Table 6.7. The results show that making the model more complex does not significantly improve the performance of the model and in some cases even negatively effects the results. In general, this means that the LSTM-Node2vec model with one layer for both

Table 4.5: Difference in performance by adding one more layer to the encoder

	AUC	Mac-f1	Mic-f1
Radoslaw	-0.0085	-	-
St-Ov	-0.0014	-	-
AS	-0.0096	-	-
Dblp	-	-0.0045	0.0086
Acm	-	0.0036	0.0059

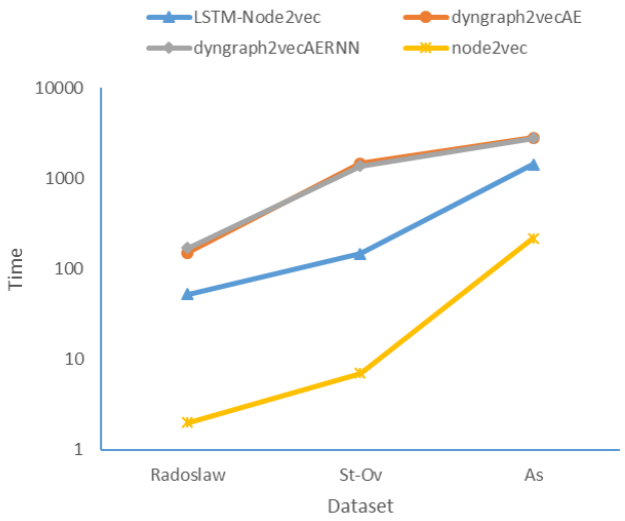


Figure 4.7: Time complexity Analysis for 4 time steps in 3 datasets

the encoder and decoder can return results that are comparable to more complex models.

4.4.8 Time Analysis

We compare the running time of LSTM-Node2vec to the static embedding model Node2vec and two dynamic models dyngraph2vecAE and dyngraph2vecAERNN on three datasets, AS, Radoslaw and St-Ov. We ran the experiments on a Ubuntu server with 512 GB RAM, 7 GPUs with 2 x Intel Xeon E5-2687W v4 3.0 GHz each. The results in Figure 4.7 show that LSTM-Node2vec is slower than Node2vec. This is because Node2vec only focuses on one graph for the computation. However, LSTM-Node2vec uses historical information from previous graphs in the current graph embedding. Compared to the two dynamic embedding models, the running time of LSTM-Node2vec is better as LSTM-Node2vec uses random walks instead of adjacency matrices of the graphs. We are working on implementing a distributed version of LSTM-Node2vec to lower the running time further.

4.5 Summary

We have presented LSTM-Node2vec, a dynamic network embedding method that combines an LSTM autoencoder and the node2vec model to learn node representations in dynamic graphs. We defined temporal neighbor walks that capture evolving patterns in the history of nodes of the graphs. In order to generate dynamic network embeddings, we first trained an LSTM autoencder with temporal neighbor walks to extract temporal information of nodes over time. The weights of the LSTM model is then used to initialize a node2vec model to learn node representations based on the structure of the current graph. This way we incorporated temporal information into static local states of the nodes, which leads to producing better node representations. We evaluated the effectiveness of our model in three applications including anomaly detection, link prediction and node classification tasks. Our method outperformed other state-of-the-art static and dynamic embedding methods in most of the cases. For future work, we are interested in adding attention mechanism to the LSTM-Node2vec model. This approach can be specially useful in the anomaly detection application that we want the model to focus more on the anomalous parts of the dataset. Furthermore, in the current version of the method, the length of L is fixed. One interesting future work is to automate the process of choosing the length of L. This way the model can adjust L based on how much of the history it wants to include in the computation. Therefore, if the history of a graph is not relevant to the current state of the graph, the effects of it in the current embedding will be limited. Similarly, if a graph does not change significantly over time, the history will be given better consideration.

Chapter 5

Temporal Graph Representation Learning via Maximal Cliques

5.1 Introduction

Many of the real-world data can be modeled using graphs. Graphs represent the entities using nodes and relationships between entities using edges. Common examples of graphs are social networks and E-commerce networks. In social networks, nodes represent people and edges represent interactions between them. In E-commerce platforms, nodes represent users and items, and edges can represent purchasing relationships between users and items. A key issue in analyzing and learning from graphs is how to represent the graphs effectively. The goal of graph representation learning is to preserve the graph's structure in a way that is useful for downstream applications such as node classification, link prediction and anomaly detection. Depending on their techniques, graph representation learning methods can be categorized into GNN-based and non-GNN-based approaches.

Non-GNN-based methods utilize various techniques such as random walks [30, 6], temporal point processes [85] and neural networks [7, 341]. These methods can be applied on static and dynamic graphs. Examples of static non-GNN-based methods are Node2vec [30] and Deepwalk [32]. Dynnode2vec [6], LSTM-Node2vec [7] and DynGEM [3] are examples of dynamic non-GNN-based methods. The main limitations of non-GNN-based methods include that they cannot generate embeddings for unseen nodes and that they do not use node and edge features. The GNN-based approaches leverage deep learning modules and aggregate the information of node neighbors to generate node embeddings [366]. These methods have demonstrated great performance in different applications. One of the main advantages

of these methods is that they are inductive and can be applied to unseen nodes. In addition, they can consider features of nodes and edges in generating node embeddings. GNN-based methods are mainly divided into static, spatio-temporal and dynamic/temporal categories. Static methods such as GCN [106], GraphSAGE [107] and GAT [367] are suitable for static graphs. These methods generate node embeddings by averaging embeddings of node neighbors using different operations including mean, LSTM, pooling and attention. While these methods proved to be strong in learning representation of static graphs, most of the real-world graphs are dynamic and change over time.

As a result, spatial-temporal and dynamic/temporal methods have been proposed to consider evolution of graphs. The spatial-temporal methods integrate graphs' spatial and temporal features and have been widely applied to traffic prediction [368, 369] and human trajectory forecasting [370]. Dynamic/temporal GNNs consider a broader range of changes in the graph over time. Examples of these methods are TGAT [230] and TGN [233]. In TGAT, each node is represented by a concatenation of the node and time features. This method obtains node representations by applying an attention mechanism over neighbors of a node. TGN saves the node embeddings in memory units to capture the history of nodes and updates them when new interactions occur in the graph. However, most of these methods do not consider important substructures such as triads, cliques and motifs in the networks [9].

There are a few dynamic/temporal GNN-based methods which leverage network substructures. MTSN [9] captures motifs in dynamic networks. To do so, it first divides the graph into multiple snapshots. Then, it generates the node representations at each snapshot by applying a graph convolutional network and a temporal shift operation on adjacency and motif matrices of the snapshot. However, this method may lose some fine-grained dynamic information between graph snapshots. CAW-N [10] considers dynamic graphs as a stream of edges and uses set-based anonymous walks to capture the temporal motif structures. In CAW-N, for each link, an edge embedding is obtained by aggregating anonymous walks generated for the two nodes connected by the edge. Although this method is shown to outperform previous methods in the link prediction task, it does not provide node embeddings for prediction tasks on nodes such as node classification. Moreover, it does not scale well on large graphs as we will explain in Section 5.5.7.

In this work, we propose a novel temporal graph representation learning method based on maximal cliques, called **TGR-Clique** short for **T**emporal **G**raph **R**epresentation learning via maximal **Clique**s. Maximal cliques are important graph substructures

that represent groups of densely connected nodes. These subgraphs have been successfully used in community detection in social networks [19, 20], clustering in wireless networks [21] and protein-protein interaction detection in biological networks [22, 23]. TGR-Clique generates a node's embedding in dynamic graphs by aggregating the node's temporal random walks on the maximal cliques containing the node. Temporal random walks respect the time order between nodes and therefore, capture the temporal dependency between nodes. The embedding of a node is obtained by aggregation of the temporal random walks starting from the node. We will show that the embeddings generated in TGR-Clique can lead to better predictive performance in downstream tasks and that our method is faster than CAW-N and some other state-of-the-art methods. In addition, our method can be used in both link prediction and node classification tasks since we produce both node and link embeddings.

In brief, the main contributions of our work are as follows:

- We present a novel temporal graph representation learning method based on maximal cliques for both inductive and transductive learning tasks.
- Our method generates temporal random walks on maximal cliques of the graph to capture the evolution of the graph over time. By leveraging maximal cliques in random walk generation, we identify the most connected neighbors of a node to consider in its representation.
- We evaluate the performance of our method in link prediction and node classification tasks on seven real world datasets against five state-of-the-art models to show that our method achieves better or comparable predictive performance while being faster than other methods compared. In addition, we perform ablation studies to show the importance of each feature of the proposed method.

In the rest of the chapter, we first provide some preliminaries on maximal cliques. Then, we describe our model and present the results of the evaluation of the method. Finally, we conclude the study.

5.2 Preliminaries

In this section, we provide the definitions of cliques and maximal cliques, and briefly describe the algorithm that we utilize for the maximal clique generation.

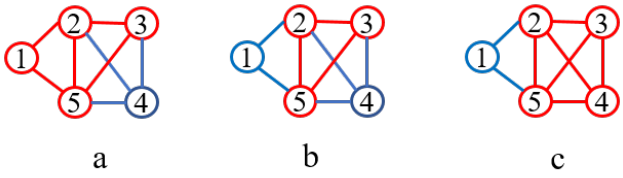


Figure 5.1: A graph with five nodes {1, 2, 3, 4, 5} with a subgraph depicted in red: a) the red subgraph is not a clique as there is no edge between nodes 1 and 3. b) the red subgraph is a clique but not a maximal one because node 4 can be added to the clique. c) the red subgraph is a maximal clique as there is an edge between every pair of nodes and it does not have a supergraph that is also a clique.

Definition 21 *(Clique [371]). A clique in a graph G is a subgraph of G, in which there is an edge between every pair of nodes.*

Definition 22 *(Maximal clique [371]). A maximal clique of a graph G is a clique that is not contained by another clique of G.*

Maximal cliques represent maximal sets of densely connected nodes. Figure 5.1 shows examples of cliques and maximal cliques.

Finding maximal cliques in a graph is a fundamental problem in the graph theory. Solving this problem is time-consuming as the number of cliques in graphs can be exponential. However, many of the real-world graphs are sparse [372]. Therefore, several algorithms were proposed to solve this problem in much less time considering the sparsity property of real-world graphs [373, 374, 375]. We employed the *Eppstein et al. algorithm* [373, 374] which is one of the fastest algorithms for maximal clique generation. This algorithm is based on Bron et al. [1] and Tomita et al. [375] algorithms. The Bron et al. algorithm is a simple depth-first search (DFS) method. It starts from every node in the graph and recursively traverses the neighbors of the node to find the maximal cliques containing the node. For instance, Table 5.1 shows the potential steps in this DFS search to find the maximal clique $\{5, 4, 2, 3\}$ in the graph in Figure 5.1. In step 1, it starts from node 5. In step 2, among the neighbors of 5 it selects node 4 and adds it to the partial clique created until this step. In each future step, a node from the candidate neighbors set which is the intersection of neighbors of all the nodes in the partial clique is selected and added to the partial clique in the next step. Finally, in step 4, when no node is left in the candidate neighbors set, the maximal clique is returned. The Tomita et al. algorithm proposes an improved version of this algorithm that avoids generating duplicate maximal cliques when traversing all the nodes in the graph. However, the worst-case time complexity of this algorithm is $O(3^{n/3})$ where n is the number

Table 5.1: An example showing potential steps for generating maximal clique $\{5, 4, 2, 3\}$ in the graph in Figure 5.1 using the original recursive method in [1].

Step	Node	Partial clique	Candidate neighbors
1	5	{5}	{1,2,3,4}
2	4	{5,4}	{2,3}
3	2	{5,4,2}	{3}
4	3	{5,4,2,3}*	{}

of nodes in the graph. The Eppstein et al. algorithm significantly speeds up the Tomita et al. algorithm by leveraging the concept of degeneracy which is low in sparse graphs.

Definition 23 *(Degeneracy [373, 374, 376]). The degeneracy of a graph $G = (V, E)$ is the smallest number d such that every subgraph of G contains a node v where degree$(v) \leq d$.*

A graph of degeneracy d has a *degeneracy ordering* of nodes, in which at most d of a node's neighbors appear after it [373, 374]. For instance, in the graph in Figure 5.1, the degeneracy of the graph is 3 and one possible degeneracy ordering of the graph is $1, 5, 4, 3, 2$. As an example, node 5 has 3 of its neighbors that occur after it in the ordering. The Eppstein et al. algorithm utilizes the degeneracy ordering to limit the number of nodes that are considered to be added to the maximal clique to at most d in each recursive call. In order to do that, it first sorts the nodes in the graph based on the degeneracy order of the graph. Then, it calls the Tomita et al. algorithm on each node to generate its maximal cliques using its neighbors that appear after that node in the degeneracy ordering of the graph. The time complexity of this algorithm is $O(dn3^{d/3})$ which is much faster than the previous methods as the degeneracy d is small in sparse graphs. For details of the algorithm, please see [373, 374].

5.3 Problem definition

Let $G = (V, E, T)$ be a temporal graph where V is the set of nodes, E is the set of edges and T is the set of time stamps on the edges. An edge in the graph is denoted as $(v_i, v_j, t_{i,j})$ where v_i and v_j are the nodes connected by the edge and $t_{i,j}$ is the time stamp the edge occurs. Our goal is to generate temporal embeddings for nodes and edges in the graph so that they can be used in downstream tasks such as node classification and link prediction. We generate temporal node embeddings using maximal cliques to capture the most connected neighbors of nodes in the

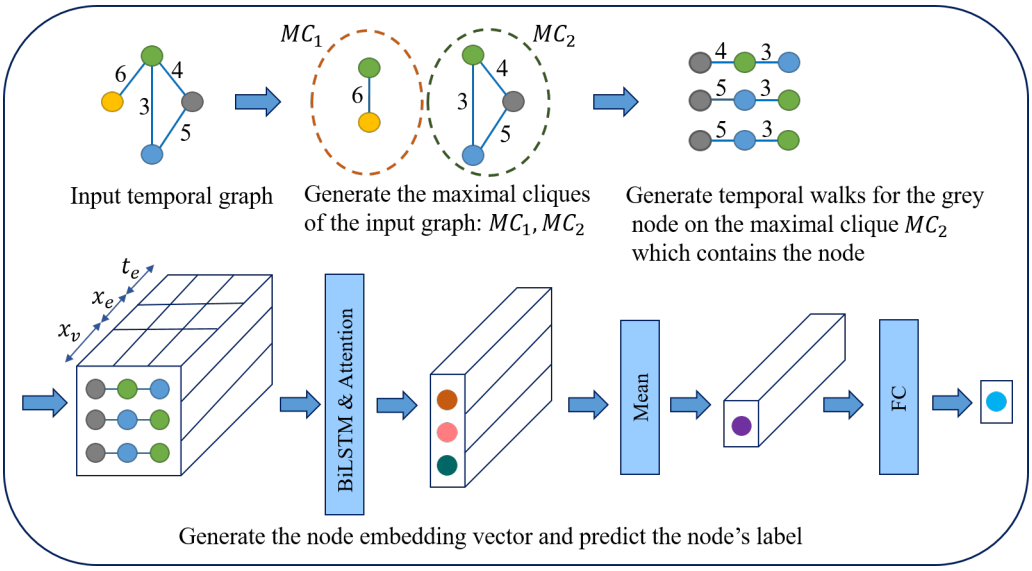


Figure 5.2: The general framework of TGR-Clique for the node classification task. The input is the temporal graph. We generate the embedding for the grey node. First, a set of temporal walks are generated for the node from the maximal cliques containing the node. For the illustration, the number and length of walks are 3 and 2. Then, these walks are input to the BiLSTM, attention and mean layers to generate the node embedding (block with the purple dot). x_v, x_e, t_e are the node, edge and time features associated with each node in the walks. The framework for link prediction is similar except that embeddings from two nodes are concatenated to input to the FC layer for link prediction.

embeddings. We obtain the temporal node and link embeddings by taking temporal random walks on maximal cliques.

5.4 Proposed Method

5.4.1 Method overview

The core of our method generates temporal node representations in two main steps: 1) for each node v at a time point, a set of temporal random walks is generated on maximal cliques that contain the node v; 2) the set of generated random walks are encoded and aggregated by our model TGR-Clique using a BiLSTM and attention mechanisms to generate the final embedding for the node. Figure 5.2 illustrates our model.

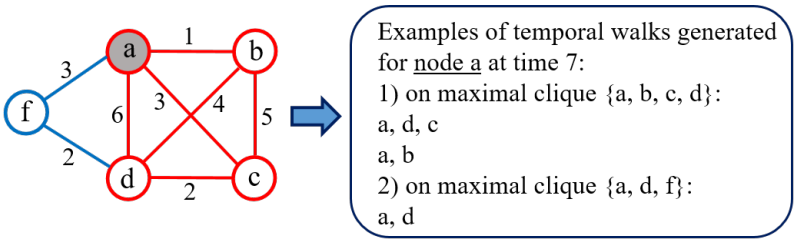


Figure 5.3: Examples of temporal random walks for node a, at time 7 on the graph's maximal cliques. For instance, a, d, c is a temporal random walk of length 2 for node a. In this walk, node d is a neighbor of node a and node c is a neighbor of node d. The nodes in the walk belong to the maximal clique $\{a, b, c, d\}$ which contains node a. The edges in the walk are sampled in an anti-chronological order such that $7 > 6 > 2$. No other edge can be added to the walk after node c.

5.4.2 Temporal walk generation

At the beginning of training, maximal cliques of the training graph are generated using Eppstein et al. algorithm [373, 374] as described in Section 5.2. A temporal walk for node v_i at time t, denoted as $v_i, v_{i+1}, ..., v_{i+l}$, is generated on a maximal clique that contains the node v_i. Therefore, all the nodes of a walk belong to the same maximal clique. In order to consider the temporal behavior of the nodes in the graph, edges in the walk are sampled in the anti-chronological order, meaning that the time that the edge (v_k, v_{k+1}) occurs is more recent than the time that the edge (v_{k+1}, v_{k+2}) occurs and all the edges have occurred before time t, where $k = i, ..., i + l - 2$ and l is the length of the walk. Formally, a temporal walk for node v_i at time t is expressed as:

$$\begin{aligned} w_i^t = &\{v_i, v_{i+1}, ..., v_{i+l} \mid t > t_{i,i+1} > ... > t_{i+l-1,i+l}, \\ &(v_k, v_{k+1}) \in MC_m \text{ for } \{k = i, i+1, ..., i+l-1\}\} \end{aligned} \tag{5.1}$$

where MC_m is the mth maximal clique containing node v_i. Figure 5.3 shows examples of valid walks that can be generated on a toy graph. For each node, we generate M temporal random walks from M maximal cliques that are randomly sampled with replacement from the entire set of maximal cliques that contain the node. By doing random sampling, the walks for a node are generated from a variety of maximal cliques containing the node, so that a variety of its neighbors can be included in the random walks.

5.4.3 Model

After generating a set of temporal walks for a node v_i at time t represented as W_i^t, the TGR-Clique model encodes and aggregates the walks of a node to generate the node embedding. In a walk $w_i^t \in W_i^t$, each node is represented by the concatenation of the node features, features of the edge connecting to the node in the walk and features of the timestamp when the edge occurs.

$$\begin{aligned} w_i^t =& \{v_i, v_{i+1}, ..., v_{i+l}\} = \{(x_k \oplus x_{k-1,k}^e \oplus f(t_{k-1,k})), \\ & \text{for } \{k = i, i+1, ..., i+l\}\} \end{aligned} \tag{5.2}$$

where x_k are the features of the node v_k, $x_{k-1,k}^e$ are the features of the edge (v_{k-1}, v_k), f computes time features of the timestamp the edge occurs, and $\oplus$ is the concatenation operator. As the walk starts from node v_i, $x_{i-1,i}^e$ is replaced with a zero vector and $t_{i-1,i} = t$. The function f encodes the time into a vector of length d_t using the harmonic encoder [230, 10]:

$$f(t) = [\cos(w_1 t), \sin(w_1 t), ..., \cos(w_{d_t/2} t), \sin(w_{d_t/2} t)] \tag{5.3}$$

where parameters $w_1, ..., w_{d_t/2}$ will be learned during training. This time encoding method has shown to do better time modeling than using the actual time values [230]. Our model encodes the walk which is a sequence of node representation vectors into a single vector using a BiLSTM model as follows:

$$Enc(w_i^t) = \text{BiLSTM}(w_i^t) \tag{5.4}$$

The matrix containing all the encoded walks of a node is denoted as $Enc(W_i^t) \in \mathbb{R}^{M \times d_c}$, where d_c is the length of concatenation of node features, edge features and time features for a node. Inspired by [10], the set of encoded walks of a node are then projected using a scaled dot-product multi-head attention mechanism [265] to capture the relations between walks. In the multi-head attention, each attention head output h is defined as:

$$h = \text{Attention}(Enc(W_i^t)) = \text{Softmax}(\frac{QK^T}{\sqrt{d_c}})V \tag{5.5}$$

where $Q = Enc(W_i^t)P_Q$, $K = Enc(W_i^t)P_K$ and $V = Enc(W_i^t)P_V$ are queries, keys and values and P_Q, P_K, P_V are learnable weight matrices. Assuming there are N heads, we have N attention head outputs denoted as $\{h_k = \text{Attention}_k(Enc(W_i^t)), \text{for } \{k =$

$1, ..., N\}\}$. The multi-head attention output is the concatenation of the output of N heads as follows:

$$\text{Multi-head attention}(Enc(W_i^t)) = (h_1 \oplus ... \oplus h_N)P^O \tag{5.6}$$

where P^O is a learnable weight matrix. The projected encoded walks are then aggregated using a mean operation:

$$z_i = Mean(\text{Multi-head attention}(Enc(W_i^t))) \tag{5.7}$$

where z_i is the representation vector for node v_i at time t.

5.4.4 Training

TGR-Clique is trained for the link prediction task and can be applied to both link prediction and node classification. For this task, the embedding of a link (v_i, v_j) denoted by $z_{i,j}$ is obtained by concatenation of z_i and z_j vectors where z_i and z_j are the embedding of nodes v_i and v_j, respectively. The $z_{i,j}$ is then forwarded to a two-layer MLP which generates a probability score for the link being positive. The link prediction loss is the binary cross entropy.

$$\text{Loss} = \sum_{(v_i,v_j)\in E^+ \cup E^-} y_{i,j}\log(\sigma(z_{i,j}^T\theta)) + (1 - y_{i,j})\log(1 - \sigma(z_{i,j}^T\theta)) \tag{5.8}$$

where E^+ and E^- are positive and negative samples. Positive samples are the links in the temporal graph and negative samples are generated between nodes that do not have any interaction in the dataset. Following [230, 10, 233], an equal number of negative samples are used in the dataset. $y_{i,j}$ is the true label of the sample (v_i, v_j), where $y_{i,j} = 1$ if there is a link between v_i and v_j and 0 otherwise. σ and θ are the sigmoid function and parameters of the MLP layers.

The training framework is shown in detail in Algorithm 1. The input graph is a set of time-stamped edges, which is divided into training and validation sets where all the edges in the validation set occur after the edges in the training set. The algorithm first generates the maximal cliques based on the training set. Then, the training data is split into B batches along the time dimension. In each epoch, we iterate over training batches. For each edge $(v_i, v_j, t_{i,j})$ in a batch, M temporal random walks are generated for both v_i and v_j with respect to time $t_{i,j}$. The walks are used to generate the embeddings (z_i and z_j) of v_i and v_j by aggregating the encoded walks using the BiLSTM, Multi-head attention and Mean layers. The two node embeddings are then concatenated into an edge embedding z_{ij}. For each batch, we

also randomly generate a set of "negative" edges which do not occur in the training set. The embedding of each negative edge is computed in the same way as for the positive edges. At the end of each batch, the loss function is computed and all the model parameters are updated using backpropagation. We use an early stopping mechanism in training. After each training epoch, the model is evaluated on the validation set for link prediction. If the improvement in the average precision of the model over the best model in the previous epochs does not exceed a threshold after a consecutive round of epochs, the training stops and the best model so far is returned. The ***average precision*** is computed as $\sum_i (r_i - r_{i-1})p_i$, with p_i and r_i denoting precision and recall at ith probability threshold that determines whether an edge is positive or not based on the predicted probability score.

5.4.5 Inference

For inference on the validation set, we perform link prediction on each edge in the validation set and a set of random negative edges generated between nodes that do not have any direct connections in the training and validation sets. In order to generate a prediction for an edge, we generate a set of temporal random walks for each node of that edge. A temporal random walk from a node of an edge is generated by starting at the edge and walking randomly in the anti-chronological order along the edges in both validation and training data. The temporal random walks for each node are then input to the trained model and the model returns the link prediction.

Similarly, the inference on the test set is performed by evaluating the model for edges that exist in the test set and a set of random negative edges generated between nodes that are not directly connected in the entire dataset. The temporal walks for each node of an edge are obtained by taking random walks in the anti-chronological order on the edges that occur before that edge in the dataset. These walks are given to the model to generate node and link embeddings. The temporal walks for the nodes in the validation set and test set are generated on the graph rather than maximal cliques to avoid generating maximal cliques during inference to speed up prediction.

Our model can do both transductive and inductive inference. In the transductive inference, all the nodes in the test/validation set were observed in the training set. In the inductive inference, some nodes in the test/validation set did not exist in the training set. A new node in the test/validation set may or may not have a connection to other nodes in the data. If a new node does not have any connection, the temporal random walks for that node only contains that node and only node features are used

Algorithm 3 : **Training framework**

Input: Graph $G = (V, E, T)$ containing a *training set* of an edge sequence and a *validation set* of an edge sequence, number of random walks M, length of random walks l, batch size B, maximum number of epochs $epoch_{max}$
Output: Trained model
$MC \leftarrow$ Generate all the maximal cliques for the training set
Divide the training data (a sequence of edges ordered by timestamps) into B batches $(E_1^+, E_2^+, ..., E_B^+)$ along the time dimension
for $i = 1, 2, ..., epoch_{max}$ **do**
  for each batch $E_b^+ \in$ training set $(b = 1, ..., B)$ **do**
    for all $(v_i, v_j, t_{i,j}) \in E_b^+$ **do**
      Generate M temporal walks of length l for v_i on MC containing v_i using edges that occur before $t_{i,j}$
      Generate M temporal walks of length l for v_j on MC containing v_j using edges that occur before $t_{i,j}$
      $z_i, z_j \leftarrow$ Generate embedding for v_i, v_j by aggregating their encoded walks using the BiLSTM, Multi-head attention and Mean layers
      $z_{ij} \leftarrow$ Concatenate z_i, z_j to get the edge embedding
      Compute the output of the fully connected layers
    end for
    $E_b^- \leftarrow$ Generate a random negative sample of size b containing pairs (v_i, v_j) where nodes v_i and v_j exist in E_b^+ but there is no edge between them in the training data.
    for $(v_i, v_j) \in E_b^-$ **do**
      $z_i, z_j \leftarrow$Get the embeddings of v_i, v_j, which were computed in the above for-loop for positive examples
      $z_{ij} \leftarrow$ Concatenate z_i, z_j to get the edge embedding
      Compute the output of the fully connected layers
    end for
    Compute Loss based on Equation (5.8)
    Update all the model parameters based on the Loss via backpropagation
  end for
  Evaluate the model on the validation data
  if Average precision on the validation set does not improve the previous best model by a tolerance value for a consecutive few epochs (e.g., 3 epochs in our experiment) **then**
    return the best model in the previous epochs
  end if
end for
return the trained model

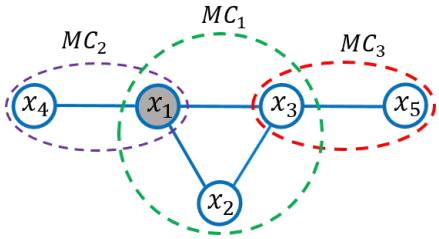


Figure 5.4: An MRF over random variables $X = \{x_1, x_2, x_3, x_4, x_5\}$ with three maximal cliques MC_1, MC_2 and MC_3.

as the input to the trained model to generate the node embedding. In cases where a new node is connected to other nodes either in the training or test/validation set, a random walk longer than one node can still be generated to obtain an embedding of the node considering both the node features and its connection structures.

In addition to link prediction, our model can also be used for node classification by training a node classification model with the node embeddings generated by our model.

5.4.6 Justification of using maximal cliques

We explain why TGR-Clique is designed to walk within maximal cliques when generating the random walks for node embedding. To that end, we take a probabilistic view on graphs using Markov random fields (MRF). An MRF is an undirected graphical model described by a graph in which each node is associated with a random variable. Nodes in an MRF satisfy the Markov property stating that a node is independent of other nodes given its immediate neighbors, which are connected by an edge to the node [377, 378]. Therefore, assuming this property, a node is independent of other nodes given the nodes in all the maximal cliques containing the node because the nodes in the maximal cliques containing a node are all the immediate neighbors of the node. For instance, in Figure 5.4, we can say, $p(x_1|x_2, x_3, x_4, x_5) = p(x_1|x_2, x_3, x_4)$. The goal of TGR-Clique in generating random walks on the maximal cliques is to capture the nodes in maximal cliques and the dependencies between them. Assuming that all the nodes in the maximal cliques containing x_1 are captured in the random walks, x_1 is independent of others nodes. Even though it is possible that TGR-Clique does not cover all the nodes in the maximal cliques by random walks, it excludes the nodes that x_1 is less dependent on, which are the nodes not in any of x_1's maximal cliques.

In addition, the probability distribution $p(X)$ over random variables $X = \{x_1, ..., x_n\}$ in an MRF can be expressed as follows [378, 379]:

$$p(X = \{x_1, ..., x_n\}) = \frac{1}{Z} \prod_{MC_i \in MC} \phi_{MC_i}(X_{MC_i}) \tag{5.9}$$

where MC is the set of all the maximal cliques of the graph and MC_i is the ith maximal clique. $Z > 0$ is a normalization term. ϕ_{MC_i} (or in short ϕ_i) is a non-negative potential function of MC_i, which captures dependencies of variables within the clique. X_{MC_i} is the set of the nodes in the MC_i. Based on Equation (5.9), the following can be derived with our example[1]:

$$\begin{aligned} p(x_1|x_2, x_3, x_4, x_5) &= \frac{p(x_1, x_2, x_3, x_4, x_5)}{p(x_2, x_3, x_4, x_5)} \\ &= \frac{\frac{1}{Z}\phi_1(x_1, x_2, x_3)\phi_2(x_1, x_4)\phi_3(x_3, x_5)}{\int \frac{1}{Z}\phi_1(x_1, x_2, x_3)\phi_2(x_1, x_4)\phi_3(x_3, x_5)dx_1} \\ &= \frac{\phi_1(x_1, x_2, x_3)\phi_2(x_1, x_4)}{\int \phi_1(x_1, x_2, x_3)\phi_2(x_1, x_4)dx_1} \end{aligned} \tag{5.10}$$

Therefore, as $p(x_1|x_2, x_3, x_4, x_5) = p(x_1|x_2, x_3, x_4)$, the probability distribution of x_1 given all the nodes in the maximal cliques containing x_1 correlates with the product of potential functions of all maximal cliques containing x_1. To account for this, TGR-Clique generates separate walks on each maximal clique to capture the dependencies between nodes in each clique and then aggregate the walks to generate the embedding of node x_1. The same applies to other nodes in the graph. We demonstrate that TGR-Clique have better or comparable predictive performance compared to other baselines in our experiments in Section 5.5. We will also show that TGR-Clique is faster and more scalable than other baselines.

5.5 Experiments

We evaluate the performance of TGR-Clique in link prediction and node classification tasks.

5.5.1 Baselines

We compare the results of our model against the following state-of-the-art graph representation learning methods.

[1]The derivation follows the one in [377].

Table 5.2: The datasets statistics. #MC is the number of maximal cliques in the training set, #F is the number of node and edge features.

Dataset	Nodes	Edges	Time	#MC	#F
Contacts	11k	415k	80 days	14k	0
SMS	44k	548k	338 days	35k	0
Wiki	9k	157k	1 month	13k	172
Fb	2k	60k	216 days	9k	0
Reddit	10k	672k	1 month	63k	172
Dblp	61k	493k	9 years	27k	0
Acm	38k	226k	8 years	14k	0

- Jodie [380]: Joint Dynamic User-Item Embedding method consists of update and project operations that update embedings of users and items after each interaction and project the future embedding of users.
- DyRep [87]: DyRep captures the evolution of graphs using two temporal point processes for communication and association events which occur between nodes and their neighbors.
- TGN [233]: Temporal Graph Network is a temporal GNN that uses memory modules to keep history of nodes and updates nodes embeddings as new interactions emerge.
- TGAT [230]: TGAT is a temporal GNN and a temporal version of GAT model. It concatenates time features with node features to consider evolution of the graph.
- CAW-N [10]: CAW-N is a temporal GNN model that captures the evolution of motifs in the graph using set-based anonymized random walks.

5.5.2 Settings

For TGR-Clique, we use the Adam optimizer with a learning rate $1e-4$. The batch size is 64 and the max epoch number is 50 using early stopping with a tolerance $1e-3$ and max-round epochs 3. All the experiments were performed on an Ubuntu 16.04 server with a NVIDIA GeForce GTX 1080Ti 11 GB GPU and an Intel Xeon 3.0 GHz 12-core processer. The results are the average over 10 runs. The datasets are undirected. Following [230, 10], for datasets that have no feature, we used all-zero feature vectors. The dimension of feature vectors is set as 100. We used the public implementation of CAW-N and TGAT methods. The code for Jodie, DyRep and

TGN were all provided in the TGN public repository. All the baselines are run with their default parameters. The only exception is the CAWN method on the Wiki dataset for which the default parameters do not yield good results (achieving only 70% AUC) and we thus set $\alpha = 1e - 5$ as suggested in the README file provided in the code package [381]. The AUC and AP values for the DyRep method on Wiki in Table 5.3 and 5.4 were taken from [382] as its program returned memory errors on our server. The results for Jodie, DyRep, TGN and TGAT on Reddit for both inductive and transductive link prediction tasks were also taken from [382].

5.5.3 Link prediction

The link prediction task was performed on the following five public datasets. Table 5.2 shows the dataset statistics.

- **Fb-messages** (**Fb**) [364]: a social network similar to Facebook for students of UCI. The nodes are the students and the edges are the messages sent between them.
- **Contacts-dublin** (**Contacts**) [364]: a human contact network from an exhibition in Dublin, where nodes are people and an edge means that the two people have been in contact.
- **SMS-A** (**SMS**) [364]: a communication network where a node is a user and an edge represents messages sent between two users.
- **Wikipedia** (**Wiki**) [381]: a network between users and Wikipedia pages. Nodes are users and pages. There is an edge between a user and a page if a user edited that page.
- **Reddit** [381]: a network where users and subreddit pages are nodes. An edge occurs between a user and a subreddit, if the user posts in the subreddit.

Each data set is divided into training, validation and test sets with the 70:15:15 split ratio along the temporal order of the edges. Tables 5.3 and 5.4 present the results for transductive and inductive link prediction tasks in terms of area under ROC curve (AUC) and average precision (AP) scores which are commonly used for link prediction tasks [10, 233, 230]. Both scores are computed using the corresponding functions in the Scikit-Learn library with the predicted probabilities of testing edges being positive and their true labels as inputs. For the inductive task, similar to [230, 233], we remove some of the nodes and their connected edges from the training set so

Table 5.3: The transductive link prediction results.

Method	Score	Contacts	SMS	Wiki	Fb	Reddit
Jodie	AUC	0.965	0.974	0.948	0.587	0.976
	AP	0.959	0.977	0.944	0.541	0.970
DyRep	AUC	0.965	0.972	0.942	0.511	0.980
	AP	0.961	0.977	0.946	0.493	0.980
TGN	AUC	0.998	0.990	0.983	0.879	0.986
	AP	0.997	0.991	0.984	0.883	0.987
TGAT	AUC	0.973	-	0.931	0.719	0.980
	AP	0.977	-	0.938	0.687	0.981
CAW-N	AUC	0.999	0.983	0.988	0.980	**0.999**
	AP	0.999	0.987	0.989	0.986	0.999
TGR-Clique	AUC	**0.999**	**0.997**	**0.992**	**0.982**	0.998
	AP	**0.999**	**0.997**	**0.994**	**0.986**	**0.999**

that the validation and test sets contain some nodes that did not occur in the training set. The length and number of random walks for Contacts, SMS, Fb, Wiki and Reddit are (3,8), (4,7), (2,10), (1,64) and (1,64) respectively, which are determined by a grid search around the average length and number of maximal cliques per node in the training set. Results from other parameter settings are shown in Section 5.5.6. We observe from the results that our method (TGR-Clique) outperforms Jodie, DyRep, TGN and TGAT on all the datasets in transductive and inductive link prediction tasks. TGAT did not finish on SMS and returned errors. The results suggest that considering neighbors of a node that co-exist in maximal cliques with the node provides better node representations than other methods. In comparison to CAW-N, which is a very competitive method, TGR-Clique outperforms CAW-N on SMS and Wikipedia and has a comparable performance on Contacts, Fb and Reddit. In Section 5.5.7, we will show that TGR-Clique runs much faster than CAW-N. Overall, our results indicate that considering substructures of graphs in node representations (as in TGR-Clique or CAW-N) yields better results in comparison to other methods.

5.5.4 Node classification

We ran our method on the following two datasets for the node classification tasks. Statistics of the datasets are shown in Table 5.2.

- **Dblp**[345]: a co-authorship network where nodes represent researchers and edges represent co-authorship. A node belongs to one of the two classes: data science and computer vision. A data science label is assigned to a node if

Table 5.4: The inductive link prediction results.

Method	Score	Contacts	SMS	Wiki	Fb	Reddit
Jodie	AUC	0.922	0.932	0.928	0.538	0.954
	AP	0.920	0.944	0.930	0.518	0.945
DyRep	AUC	0.923	0.931	0.910	0.508	0.957
	AP	0.922	0.943	0.919	0.508	0.957
TGN	AUC	0.991	0.968	0.976	0.800	0.975
	AP	0.987	0.973	0.977	0.821	0.976
TGAT	AUC	0.953	-	0.889	0.640	0.964
	AP	0.953	-	0.900	0.631	0.966
CAW-N	AUC	0.999	0.980	0.969	**0.989**	0.993
	AP	0.999	0.984	0.973	**0.993**	0.994
TGR-Clique	AUC	**0.999**	**0.995**	**0.996**	0.980	**0.999**
	AP	**0.999**	**0.995**	**0.996**	0.985	**0.999**

the majority of its papers appear in VLDB, SIGMOD, PODS, ICDE, EDBT, SIGKDD, ICDM, DASFAA, SSDBM, CIKM, PAKDD, PKDD, SDM or DEXA conferences. Similarly, the computer vision label is assigned to a node with the majority of its publications being in CVPR, ICCV, ICIP, ICPR, ECCV, ICME or ACM-MM conferences.

- **Acm**[345]: a co-authorship networks similar to Dblp in which a node belongs to either data science or computer vision class.

Each data set is divided into training, validation and test sets with the 70:15:15 split ratio along the temporal order of the edges. The length and number of walks for both Acm and Dblp are 2 and 2. Table 5.5 shows the results of node classification in terms of AUC similar to [230, 233]. Based on the results, our method outperforms all the baseline methods on both datasets, which suggests that considering neighbors of a node in maximal cliques in the node representation can better distinguish the node classes in the Acm and Dblp datasets. On Dblp, TGAT raises errors and was not able to finish. CAW-N is not applicable in node classification task as it only generates link embeddings. Our method is suitable for the prediction tasks on both nodes and edges.

5.5.5 Ablation study

We investigate the effects of different components of TGR-Clique on node classification and link prediction with four datasets: Acm, Dblp, Wiki and Fb. Table 5.6 depicts the results of the experiments. A major component of our model is

Table 5.5: AUC scores for the node classification.

Method	Acm	Dblp
Jodie	0.486	0.458
DyRep	0.509	0.547
TGN	0.513	0.532
TGAT	0.500	-
CAW-N	NA	NA
TGR-Clique	**0.533**	**0.551**

Table 5.6: AUC scores for the ablation study. NC and LP are short for node classification and link prediction respectively.

Dataset	Task	Original	No-clique	No-time	LSTM
Acm	NC	0.533	0.515	0.456	0.472
Dblp	NC	0.551	0.483	0.496	0.538
Wiki	LP	0.992	0.969	0.612	0.992
Fb	LP	0.982	0.973	0.981	0.981

the maximal cliques. To validate the effectiveness of maximal cliques, we compare our method with the No-clique variant in which the temporal walks are generated on the graph itself. As shown, taking random walks on maximal cliques performs better than doing these on graphs as it more likely considers similar and influential neighbors in the node embeddings. We also replace time features with zeros in the No-time variant and see that time features are helpful in our model performance. The reason is that the input graph is temporal and time features provide insights into the history of the nodes interactions. In addition, we experimented with replacing BiLSTM with LSTM. Our model performs better using BiLSTM on Acm and Dblp in node classification tasks as it considers both backward and forward dependencies between nodes in the walks. However, using BiLSTM does not have a significant effect on the performance on Wiki and Fb.

5.5.6 Parameter sensitivity

Effects of the length of walks. Part a) of Figure 5.5 depicts the effects of changing the length of walks on our method's performance on the Contacts, Fb and SMS datasets. We changed the walk length from 1 (containing 2 nodes) to 4 (containing 5 nodes). Other parameters are fixed in this experiment (which also applies to the other experiments in this section). As shown in the plot, on Contacts, the method works very well for all walk lengths and longer walks do not have significant effects on the performance of the method. On SMS, longer walks have better performance. On

Fb, the best result is achieved at the length 2 and the performance starts dropping after that. This could be due to the fact that the average length of maximal cliques in this dataset is around 2 and generating longer walks hurts the performance of the method. We recommend that grid search around the average length of maximal cliques in the training be conducted to determine the length of the walks.

Effects of the number of walks. Part b) of Figure 5.5 shows the effects of changing the number of walks in our method on the Contacts, Fb and SMS datasets. As shown, increasing the number of walks improved the performance on the Contacts dataset. The performance on SMS and Fb increases at the beginning when increasing the number of walks, but decreases slightly at the end. Similar to the length of walks, a grid search around the average number of maximal cliques per node in the training set can be used to find the best number of walks.

Effects of the output embedding dimension. We computed the AUC values for link prediction with our method on the Contacts, Fb and SMS datasets with embedding dimension values $\{50, 100, 150, 200\}$. Part c) of Figure 5.5 shows that the embedding dimension of 150 leads to the best results on all the datasets. We used an embedding dimension of 100 on these datasets in our experiments although other values may achieve better performance.

Effects of the number of attention heads. The AUC values for the link prediction task with different number of heads $\{1, 2, 4, 8\}$ on the Contacts, Fb and SMS datasets are shown in part d) of Figure 5.5. Increasing number of heads does not have a significant effect on the performance on Contacts. However, the model has the best performance on Fb and SMS with 8 attention heads. We use 8 for the number of attention heads in other experiments.

5.5.7 Time analysis

Training time analysis. We compare all the methods in terms of training time on the SMS dataset. The batch size is 64 for all the methods. For the two random-walk based methods (ours and CAW-N), the number of walks is set to 10 and the walk length is 2. To see the scalability of each method, we experiment with different subsets of SMS, whose number of edges ranges from {80k,160k,240k,320k}. Figure 5.6 depicts the training time of one epoch for all the methods over different sizes of the graph. As shown on the plot, our method is significantly faster than all the baselines. Interestingly, CAW-N is one of the most time-consuming methods

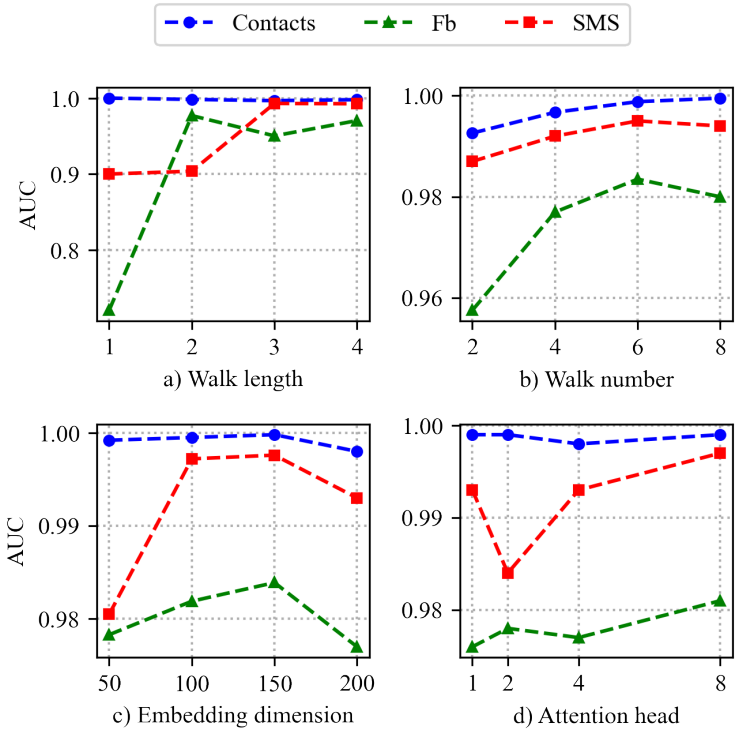


Figure 5.5: Parameter sensitivity analysis on the transductive link prediction task.

among the methods evaluated, although it is also based on random walks. This could be due to the fact that CAW-N does the set-based anonymization of random walks (which takes time) after the walks are generated, while in our method we directly use the random walks to compute node representations. In addition, CAW-N, TGN and TGAT generate a node's embedding by sampling a subgraph from the node's neighbors and then aggregating the neighbors of a node from its subgraph [383]. However, TGR-Clique does not construct a subgraph for each node which is time consuming. Instead, it generates the node's random walks on maximal cliques containing the node, which are computed at the beginning of the training.

DyRep and Jodie are also among the top time-consuming methods on larger graphs with DyRep being the worst performer in terms of training time on one epoch. This could be because DyRep uses two temporal point processes to model graphs' dynamics, which makes the model complex and time-consuming to train specially when the number of edges increases. We also measure the total run time of all the methods using the whole SMS dataset with all the edges and let all the programs run until its completion. The tolerance and the max-round epoch values used in early stopping for all the methods are set as $1e-3$ and 3, respectively. The total run time of DyRep, Jodie, TGN, CAW-N and TGR-Clique are 15307, 15086, 14380,

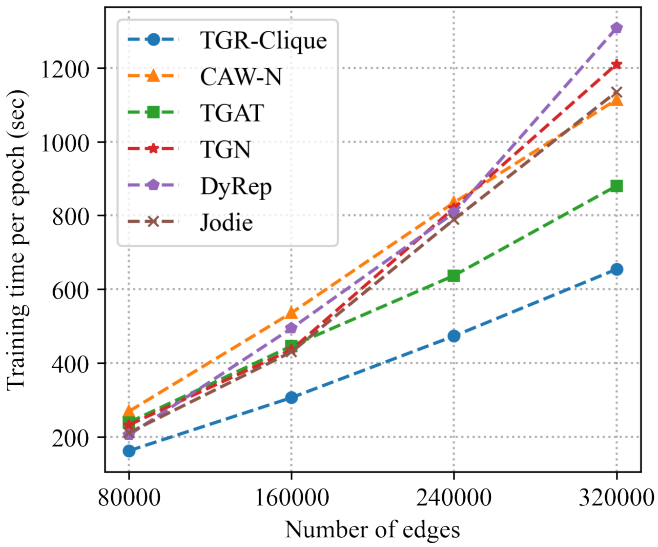


Figure 5.6: Training time on the SMS dataset over one epoch and different numbers of edges.

14443 and 9403 seconds respectively. The results indicate that our method has the lowest total run time among all the baselines.

Run time of maximal cliques generation. The run time of the maximal cliques generation algorithm on the training set of SMS is 0.163 seconds which is very fast. Similaly, on Fb, Contacts, Wiki, Dblp, Acm and Reddit, the time for generating all the maximal cliques is 0.037, 0.073, 0.048, 0.184, 0.107 and 0.198 seconds respectively.

5.6 Comparison with our other methods proposed in Chapters 3 and 4

In chapters 3 and 4, we proposed two dynamic graph embedding models: Dynnode2vec and LSTM-Node2vec. In this section, we compare these methods with TGR-Clique in terms of the models design and performance in the link prediction task.

Model design. The differences in the designs of Dynnode2vec, LSTM-Node2vec and TGR-Clique models are as follows:

1. *Dynnode2vec:* Dynnode2vec belongs to traditional dynamic graph embedding methods and is based on random walks. Compared to using the graph's adjacency

matrix as input in the previous embedding methods, random walks are more memory efficient and better capture the neighbourhood structure of the graph. Dynnode2vec is a discrete-time dynamic graph embedding method, meaning that its inputs are snapshots of the graph over time. The model at the current time point is a node2vec model that is initialized with the model's weights from the previous time point. In addition, the temporal walks are only created for a set of nodes that have changed in the current time. The training of Dynnode2vec is unsupervised in the sense that no node or edge labels are needed during the training.

2. *LSTM-Node2vec:* LSTM-Node2vec is also a traditional and discrete-time dynamic graph embedding method based on random walks. In LSTM-Node2vec, we tried to improve our previous method by integrating node2vec with an LSTM autoencoder. The reason was that an LSTM autoencoder was a powerful seq2seq model in Natural Language Processing for capturing the long-term dependencies between words. Therefore, our goal was to capture dependencies between nodes and their neighbours over time using an LSTM autoencoder and then use the LSTM's weights to initialize the node2vec model at each time point. Similar to Dynnode2vec, LSTM-node2vec is trained in an unsupervised fashion in the sense that no node or edge labels are needed during the training.

3. *TGR-Clique:* Inspired by the recent success of GNNs, we designed TGR-Clique, a GNN-based dynamic graph embedding method. This method generates a node embedding by aggregating the node's neighbours' embedding. In contrast to Dynnode2vec and LSTM-Node2vec, TGR-Clique can use the node, edge and time features. In addition, TGR-Clique is continuous-time which means that it takes as input a stream of edges with a time stamp over each edge. Compared to discrete-time methods, the advantage of continuous-time methods is that they do not lose fine-grained dynamic information between snapshots. Another distinction between TGR-Clique and the two previous methods is that TGR-Clique considers the maximal cliques in the graph, which can show strong relationships between nodes. In addition, similar to most of the other GNN-based methods, TGR-Clique uses the node or link labels as ground-truth labels during training, and thus its training is considered supervised. A summary of the differences in the design of our three methods is shown in Table 5.7.

Link prediction. We compared the performance of these methods in the link prediction task for St-Ov, Radoslaw and AS datasets. The specification of these

Table 5.7: Comparison of our 3 proposed dynamic graph embedding methods. Dyn is short for Dynamic, Cat is Category, Trad is Traditional, Continu is Continuous and Feat is Feature.

Model	Type	Cat	Input	Technique	Substruct	Feat	Training
Dynnode2vec	Dyn	Trad	Discrete	Random walk	-	-	Unsupervised
LSTM-Node2vec	Dyn	Trad	Discrete	Random walk & LSTM	-	-	Unsupervised
TGR-Clique	Dyn	GNN	Continu.	Aggregation	Maximal clique	yes	Supervised

Table 5.8: Link prediction results in terms of AUC.

Model	St-Ov	Radoslaw	AS
Dynnode2vec	0.660	0.629	**0.962**
LSTM-Node2vec	0.685	0.687	0.887
TGR-Clique	**0.714**	**0.730**	0.631

datasets are given in Chapters 3 and 4. Results of this comparison is shown in Table 5.8 in term of AUC. The result of each method is computed using its own setup. As can be seen, TGR-Clique has the best performance compared to other methods in St-Ov and Radoslaw datasets. However, in the AS dataset, dynnode2vec has the best performance. This could be due to the fact that in AS dataset, the graph does not change significantly in each snapshot. Therefore, Dynnode2vec, which initializes the weights of the current model with the weights from the previous model can generate very accurate node embeddings. However, AS dataset lacks the time information in each snapshot and among snapshots. Therefore, to create an edge stream from these snapshots, we had to give a similar time to all the edges in the same snapshot which may have an adverse effect on generating temporal random walks in TGR-Clique. Thus, the final dataset was not ideal for TGR-Clique to perform well.

5.7 Summary

We proposed TGR-Clique, a novel method for generating temporal node embeddings for temporal graphs by leveraging the history of node interactions. We utilize maximal cliques for generating temporal random walks for the nodes to consider the most connected neighbors of a node in the node representation. We compared the performance of our method in link prediction and node classification tasks on seven datasets against five state-of-the-art models. Our method is shown to have better or comparative predictive performance in both tasks while being faster than other methods compared. Furthermore, we performed an ablation study on our method and demonstrated the impact of each component of the proposed method. Finally,

we compared the performance of TGR-Clique with our two other methods proposed in Chapter 3 and 4. As future work, we will employ these temporal embeddings in other graph mining applications such as recommendation and anomaly detection and evaluate their performance.

Chapter 6

Interpretability measures for Graph Embeddings

6.1 Introduction

Graph data are rapidly growing in recent years and powerful graph representation methods are required to represent the graph information. One of the most effective ways to represent large graphs is graph embedding. Graph embedding methods map nodes or edges into a lower dimensional space such that similar nodes or edges are represented by similar vectors in the embedding space. A large number of graph embedding methods have been proposed including Node2vec[30], LINE[38], DeepWalk[32] and SDNE[39]. The use of graph embedding vectors in downstream tasks, such as node classification, link prediction and anomaly detection, has shown superior performances compared to the use of traditional methods of feature engineering for node representation [30, 38, 239].

Although graph embedding methods have been highly successful in different applications, not much attention has been given to interpreting the graph embeddings. In traditional feature engineering methods, each dimension represents a particular piece of information, such as the degree or importance of a node, which is meaningful and thus potentially useful in explaining the decision made in the application. Graph embedding dimensions, however, are usually not explainable. There have been some studies on the quantifying interpretability of the word embeddings whose embedding algorithms are the basis for some of the graph embedding methods. The common approaches in that area are based on defining prediction tasks for different aspects of language structures [334] and word intrusion test [12, 24, 25, 13].

In [334], a framework is proposed for measuring interpretability of the sentence representations. This method inspects different sentence embedding methods with respect to their capacity to encode different language information. To that end, it defines a set of prediction tasks around sentence properties including sentence length, the word content and their order in the sentence. Then, it demonstrates the relative strength of the sentence embedding methods in terms of their accuracy in those prediction tasks. This was one of the first works to design experiments to compare different embedding methods in a task-independent manner.

In [24, 25, 13], interpretability of word embeddings is evaluated using word intrusion test. In the word intrusion detection test, a noisy, unrelated word is added to a set of top 5 words with high values in each word embedding dimension and a human is asked to identify the intruder word. This is based on the idea that if an embedding dimension is coherent and as a result interpretable, a human can detect the intruder word easily. This method for evaluating intepretability of embeddings needs human effort which is costly. Therefore, [12] automates this task by comparing a set of top words in each word embedding dimension with different sets of semantically similar words to measure how well the word embedding matrix captures categories of similar words.

There are few works towards the graph embedding interpretation. In [331], node embedding methods are evaluated based on the accuracy of predicting some elementary features of the graph. [332] computes a score for each embedding dimension across node groups and a score for each node group across the embedding dimensions. These scores are based on the number of common nodes between top nodes in an embedding dimension and a node group. In addition, this paper evaluates effects of changing hyperparamerters of node embedding methods, namely the number of embedding dimensions on their performance in downstream tasks. However, this study only focuses on one node embedding method (Node2vec) and does not provide a comprehensive method for scoring and evaluating interpretability of different node embedding methods.

In this work, we aim to quantify and compare the interpretability of different graph embedding methods effectively by adapting the techniques originally designed for quantifying interpretability of word embeddings [12, 24, 25, 13]. We quantify the interpretability of a node embedding method based on the extent to which its embedding dimensions can represent the centrally-located or important nodes in a category with extreme values. For this purpose, we use centrality properties of nodes which have proven to be very successful in graph clustering and community detection tasks [26, 27, 28, 29]. In addition, we provide a method for comparing

the interpretabilities of different embedding methods. Our method is general and can be applied to any graph embedding method based on different techniques such as random walks, matrix factorization and deep learning. Overall, our contribution can be summarized as follows:

- We propose two interpretability measures for node embedding methods using the betweenness centrality and closeness centrality properties of the nodes in a graph.
- We empirically validate the new interpretability measures by comparing the ranking of embedding methods using our interpretability scores with that using a node classification task whose class labels are defined based on node centrality.
- We illustrate how to use the proposed measures to interpret embedding dimensions.
- We investigate whether there is a relationship between the proposed interpretability scores of the embeddings and the predictive performance of a downstream classification task with the embeddings.

The chapter is organized as follows. First we introduce the problem and our proposed method. Then, in the experiment section, we show the results of our experiments.

6.2 Preliminaries

We use betweenness centrality and closeness centrality properties of nodes in the definition of our interpretability measures. In this section, we briefly review the definition of these properties of graphs.

Definition 24 *(Betweenness centrality [384]). The betweenness centrality of a node measures the number of times that the node occurs in the shortest path between each pair of nodes in the graph. The normalized version of this property is formulated for each vertex v in the graph as follows:*

$$BC(v) = \frac{\sum_{s \neq t \neq v \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}}{(n-1)(n-2)/2} \tag{6.1}$$

where $\sigma(s,t)$ is the number of the shortest paths between vertices s and t, $\sigma(s,t|v)$ is the number of the shortest paths between vertices s and t that passes through v. n

is the total number of nodes in the graph, and $(n-1)(n-2)/2$ *is the normalization factor over all node pairs in the graph excluding node* v*.*

Definition 25 *(Closeness centrality [384]). The closeness centrality of a node is the inverted mean length of the shortest paths between the node and all other nodes of the graph. The normalized version of this property is defined in the following equation:*

$$CC(v) = \frac{n-1}{\sum_{s \neq v \in V} distance(v,s)} \tag{6.2}$$

where $distance(v,s)$ *is the length of the shortest path between nodes* v *and* s *and* n *is the total number of nodes in the graph.*

6.3 Problem definition

Let $G = (V, E)$ denote a graph G, where V and E are nodes and edges of the graph, respectively. A graph embedding method computes an embedding matrix $Z_{|V| \times D}$ which contains one D-dimensional embedding vector for each node of the graph where $D << |V|$. The embedding vectors of nodes in the embedding space are generated in such a way that similar nodes in the graph have similar vectors in the embedding space. In this study, our objective is to quantify and evaluate the interpretability of the graph embeddings.

Inspired by the framework for quantifying the interpretability of word embeddings from [12], we define several interpretability scores for graph embeddings using centrality properties of nodes. Node centrality is an important graph property and has been used in many studies for graph clustering and community detection tasks [26, 27, 28, 29]. We leverage the centrality properties of nodes to interpret the embedding dimensions in terms of how well they capture central or important nodes in a category.

6.4 Method

6.4.1 Approach overview

For quantifying the intepretability of graph embeddings, we first define an intepretability score for each embedding dimension and then the interpretability of the embedding method is obtained by averaging these scores for all dimensions. The interpretabiliy score for the embedding method shows overall how well the

embeddings capture the important nodes. For evaluating the interpretability of each embedding dimension, we select nodes with either high or low values of each embedding dimension and define the Betweenness Centrality Importance (BCI) or Closeness Centrality Importance (CCI) scores of each embedding dimension with respect to each category using these nodes. Then, the interpretability score of the embedding dimension is the maximum score of the dimension over all categories. Algorithm 4 presents the steps of the BCI computation. The computation of CCI follows the same steps.

Algorithm 4 :BCI score computation

Input: Graph $G = (V, E)$, Node embeddings $Z \in \mathbb{R}^{|V| \times D}$, K node categories, $C_k \subset V$ set of nodes that belong to the kth category
Output: BCI score for the node embeddings
for $d = 1...D$ **do**
  for $k = 1...K$ **do**
    $BCI^{+}_{d,k} \leftarrow \text{Average}(\{BCI_{d,k}(v_x), \forall v_x \in N^{+}_{d,k} \cap C_k, BCI_{d,k}(v_x) \neq 0\})$
    $BCI^{-}_{d,k} \leftarrow \text{Average}(\{BCI_{d,k}(v_x), \forall v_x \in N^{-}_{d,k} \cap C_k, BCI_{d,k}(v_x) \neq 0\})$
    $BCI_{d,k} \leftarrow \max(BCI^{+}_{d,k}, BCI^{-}_{d,k})$
  end for
  $BCI_d \leftarrow \max(\{BCI_{d,k}, \forall k \in \{1, ..., K\}\})$
end for
$BCI \leftarrow \text{Average}(\{BCI_d, \forall d \in \{1, ..., D\}\})$

6.4.2 The definition of interpretability scores

In this section, we define the interpretability scores for graph embedding methods. Assume that there are K categories and nodes in the graph can belong to one or more of these K categories. $C_j \subset V$ is the set of nodes that belongs to the jth category. These categories of similar nodes can be defined based on the predefined class labels, communities or partitions of the graph [332].

For measuring the interpretability of an embedding dimension d with respect to a category C_k, two sets of nodes are selected based on the values of the nodes in the embedding dimension. The first set, denoted as $N^{+}_{d,k}$, has the highest values for dimension d, and the second set, denoted as $N^{-}_{d,k}$, has the lowest values for d. The number of nodes in the two sets depends on category C_k. When used for category C_k, the number of top or bottom nodes for dimension d is set to $\lambda * n_k$ where n_k is the total number of nodes in category C_k. The parameter $\lambda \in [1, \frac{|V|}{n_k}]$ determines how many nodes can be selected from each dimension. Assigning a large value for λ relaxes the strictness of the interpretability scores of each category and each

dimension because it allows more of nodes in each dimension to be considered in the computation. A small value for λ has the opposite effect. We use $\lambda = 1$ because it gives the best result (shown in Section 6.5.7).

Definition 26 *(Betweenness centrality importance). For each node $v_x \in (N^+_{d,k} \cup N^-_{d,k}) \cap C_k$, we define the Betweenness Centrality Importance (BCI) of v_x with respect to dimension d and category C_k, based on the number of times v_x is placed on a shortest path between two other nodes of C_k as follows:*

$$BCI_{d,k}(v_x) = \frac{\sum_{v_i \neq v_j \neq v_x \wedge \{v_i, v_j, v_x\} \subseteq C_k} \frac{\sigma(v_i, v_j | v_x)}{\sigma(v_i, v_j)}}{n_b} \tag{6.3}$$

where $\sigma(v_i, v_j)$ is the number of the shortest paths from node v_i to node v_j, $\sigma(v_i, v_j|v_x)$ is the number of those paths that pass through node v_x, and n_b is the number of node pairs in category C_k at least one of whose shortest paths passes through v_x (that is, $\sigma(v_i, v_j|v_x)$ is nonzero).

Note that if $n_b = 0$ (meaning that for all node pairs (v_i, v_j) in a category C_k, $\sigma(v_i, v_j|v_x) = 0$), we assign $BCI_{d,k}(v_x)$ to 0 because node v_x does not appear in the shortest path of any node pairs in category C_k and as a result, it is not a centrally important node in that category.

Definition 27 *(Closeness centrality importance). For each node $v_x \in (N^+_{d,k} \cup N^-_{d,k}) \cap C_k$, we define the Closeness Centrality Importance (CCI) of node v_x with respect to dimension d and category C_k, based on the sum of the shortest distances between top nodes of dimension d and the category C_k as follows:*

$$CCI_{d,k}(v_x) = \frac{n_c}{\sum_{v_x \in C_k \wedge v_j \in C_k} distance(v_x, v_j)} \tag{6.4}$$

where $distance(v_x, v_j)$ is the length of the shortest path between nodes v_x and v_j. Note that n_c is the number of times that $distance(v_x, v_j)$ values are greater than zero.

The betweenness centrality importance of dimension d in both directions with respect to category C_k, denoted as $BCI^+_{d,k}$ and $BCI^-_{d,k}$, is defined by averaging the non-zero BCI scores for all the nodes in $N^+_{d,k} \cap C_k$ and $N^-_{d,k} \cap C_k$, respectively. We average over nodes with non-zero scores because these are the centrally important nodes in the category. The betweenness centrality importance of dimension d with respect to category C_k, denoted by $BCI_{d,k}$ is the maximum of $BCI^+_{d,k}$ and $BCI^-_{d,k}$

because the distinctive nodes of the dimension could be in either direction of embedding dimension [12]. The closeness centrality importance of dimension d with respect to category C_k, denoted by $CCI_{d,k}$ is defined similarly. Intuitively, the $BCI_{d,k}$ and $CCI_{d,k}$ scores measure how well dimension d represents the category C_k in terms of capturing the centrally located nodes of category C_k with its extreme values. That is, the higher the $BCI_{d,k}$ or $CCI_{d,k}$ value, the better dimension d of the embedding matrix is in capturing the centrally important nodes in the category C_k.

The betweenness centrality and closeness centrality interpretability measures of a dimension d are then computed as the maximum of $BCI_{d,k}$ and $CCI_{d,k}$ scores over all categories C_k ($k \in \{1, 2, ..., K\}$) because this way we assign a score to each dimension based on the best category that it represents:

$$BCI_d = \max_{k \in \{1,...,K\}} BCI_{d,k} \tag{6.5}$$

$$CCI_d = \max_{k \in \{1,...,K\}} CCI_{d,k} \tag{6.6}$$

Using the BCI_d and CCI_d measures for all the embedding dimensions $d \in \{1, ..., D\}$, we define the BCI and CCI scores for a node embedding method by averaging over all the D embedding dimensions generated by the node embedding method:

$$BCI = \frac{1}{D} \sum_{d=1}^{D} BCI_d \tag{6.7}$$

$$CCI = \frac{1}{D} \sum_{d=1}^{D} CCI_d \tag{6.8}$$

The higher the BCI or CCI score of a graph embedding method, the better it can represent or distinguish the important nodes.

6.4.3 Example

Figure 6.1 shows the BCI score computation for one dimension with respect to one category in a small example. In this figure, a small graph is shown along with the embedding of nodes with embedding dimension $D = 3$ and 2 node categories C_1, C_2. For computing the BCI score of the embedding dimension 1 with respect to the category 1, we compute $BCI^+_{1,1}$ and $BCI^-_{1,1}$ and the $BCI_{1,1} = max(BCI^+_{1,1}, BCI^-_{1,1})$. For computing $BCI^+_{1,1}$, first a set of nodes with high values in the dimension 1 is

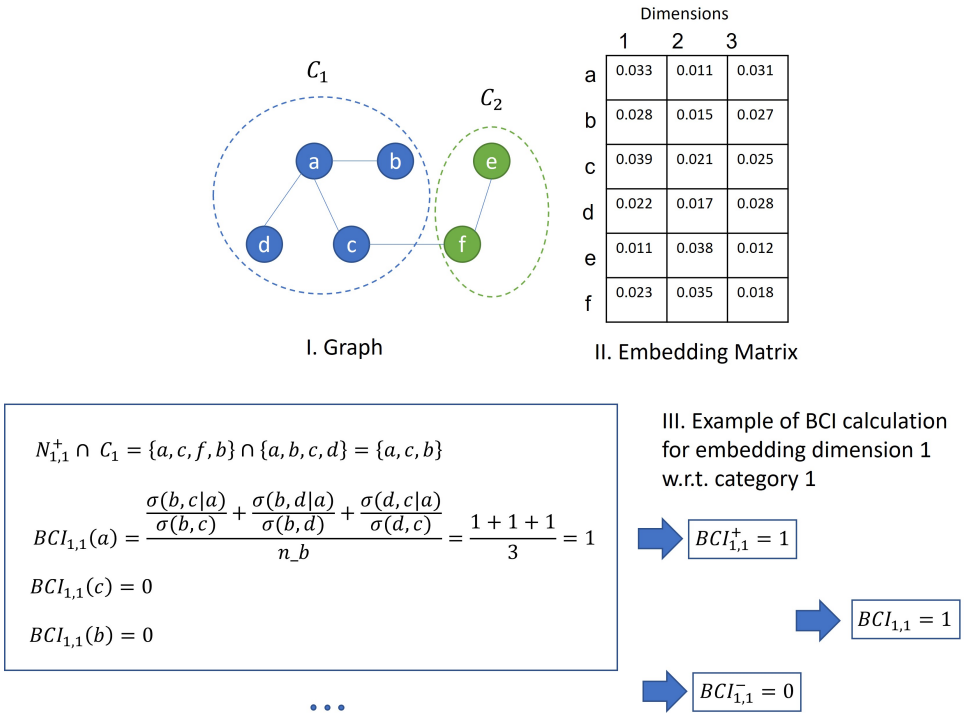


Figure 6.1: An example of BCI calculation for one embedding dimension with respect to one category. I. A sample graph with nodes $\{a, b, c, d, e, f\}$. Nodes $\{a, b, c, d\}$ belong to category 1 (C_1) and $\{e, f\}$ belong to category 2 (C_2). II. Node embedding matrix of the graph where D = 3. III. Computation of BCI score for embedding dimension 1 with respect to node category 1.

selected. $|N^+_{1,1}| = 1 * 4 = 4$ as $n_1 = 4$. Therefore, four nodes with high values in dimension 1 are selected to be compared with nodes in category 1. The set $N^+_{1,1} = \{a, c, f, b\}$.

Then, the common nodes between nodes with high value in dimension 1 and category 1 are selected which is $N^+_{1,1} \cap C_1 = \{a, c, b\}$. For each node v_x in this set, the $BCI_{1,1}(v_x)$ is computed. As an example, for node a, the $BCI_{1,1}(a)$ equals betweenness centrality score of the node a with respect to any node pairs in category 1 except itself, which are $\{(b, c), (b, d), (d, c)\}$. For nodes b and c, $BCI_{1,1}$ equals 0 as they did not occur in any shortest paths of any node pairs of category 1. Therefore, the $BCI^+_{1,1}$ which is the average of non-zero BCI scores of nodes $\{a, c, b\}$ equals 1. Assume that the $BCI^-_{1,1}$ is 0. Then, the overall BCI score for dimension 1 with respect to category 1, is 1.

6.5 Experiments

In this section, we compute the betweenness centrality importance and closeness centrality importance scores for four graph embedding methods on three datasets. We further empirically validate validity of the our interpretability measures. We also demonstrate how to use the proposed measures to interpret some dimensions of node embeddings. At the end, we investigate if there is a correlation between the interpretability measures and performance of the embeddings in downstream tasks.

6.5.1 Datasets

We computed the BCI and CCI scores for the embedding vectors generated for the following 3 datasets:

- Fb-cmu [364]. This dataset is from facebook website for students of CMU. The nodes and edges are the students and the friendship between them. The Fb-cmu dataset consists of 6k nodes, 249k edges and 3 categories of nodes.
- Wisconsin [385]. Wisconsin is a web network collected from the computer science department of Wisconsin university. The webpages are the nodes and the hyperlinks are the edges of this dataset. The categories of this dataset are student, project, course, staff and faculty. The number of nodes and edges are 251 and 499.

Table 6.1: Betweenness centrality importance and Closeness centrality importance scores

Dataset	Score	Node2vec	LINE	SDNE	HOPE
Wisconsin	BCI	0.3824	0.3560	**0.4867**	0.4520
	CCI	0.5006	0.4958	**0.5350**	0.5134
Squirell	BCI	0.0349	0.0309	**0.0424**	0.0327
	CCI	0.3711	0.3583	0.3931	**0.3973**
Fb-cmu	BCI	0.1099	0.0969	0.1069	**0.1237**
	CCI	0.5662	0.5596	**0.5679**	0.5673

- Squirrel [386]. This is a Wikipedia network dataset on the topics related to Squirrels. In this dataset, nodes are the wiki pages and edges are the mutual links between them. Nodes are classified into five classes based on the average of monthly traffic on each webpage. There are 5k nodes and 217k edges in this dataset.

6.5.2 Node embedding methods

The representation vectors of the datasets are generated using the well-known graph embedding methods given below.

- Node2vec[30]: Node2vec is a random walk based node embedding method. It generates random walks to capture the neighborhood structure of a graph and then learns the embedding by training a skipgram model.
- LINE[38]: LINE learns the representation of a graph by optimizing a loss function that preserves the first order and second order proximity in the graph separately. This method is a node embedding method that uses probabilities similar to random walk based methods.
- SDNE[39]: Structural Deep Network Embedding (SDNE) is a network embedding method that leverages an autoencoder model to preserve first-order and second order proximities of the nodes in the network.
- HOPE[35]: High-Order Proximity preserved Embedding method (HOPE) is based on matrix factorization and captures high order proximities in graphs.

6.5.3 Quantitative measuring of interpretability of graph embedding methods

We computed the betweenness centrality importance and closeness centrality importance scores for four graph embedding methods on three datasets. The categories of nodes are the true class labels of nodes in each dataset. The embedding dimension for all datasets is 128. Table 6.1 shows the results of BCI and CCI scores for the graph embedding methods. As shown in Table 6.1, for the Wisconsin dataset, both the BCI and CCI scores are the highest for the SDNE method. In the Squirell dataset, SDNE has the max BCI score and the CCI scores of the SDNE and HOPE are very similar and the highest compared to Node2vec and LINE. For the Fb-cmu dataset, HOPE has the best BCI and again HOPE and SDNE have the highest CCI scores compared to other methods. Overall, BCI and CCI scores perform similarly and identify SDNE and HOPE as the most interpretable embedding methods on all the datasets. HOPE and SDNE both use adjacency matrices of the graph to generate embedding vectors and based on the results, they preserve the important nodes in categories in the graph better than Node2vec and LINE which generate embeddings based on random walks.

6.5.4 Validation

In this section, we investigate the validity of the BCI and CCI scores of node embeddings in terms of whether they capture the centrality of nodes. For evaluating the word embedding interpretability, Adi et al. [334] score the interpretability of the sentence embeddings based on the accuracy of a classifier that is trained on the generated embeddings in predicting different aspects of sentence structures. Similarly, [331] uses this technique for interpreting node embeddings. Here, we investigate if there is a correlation between the ranking of node embedding methods from our BCI and CCI scores and that from a node classification method (defined below to predict node centrality) according to the classification accuracy.

To that end, we train three classifiers on node embeddings to classify nodes based on some graph information. In these classification tasks, each instance in the dataset is the embedding vector of a node and the instance label is a binary label based on some node properties. Here, we create binary labels for each node by discretizing the degree, betweenness centrality and closeness centrality of nodes. For degree label, the label of a node is 1 if the node degree is above the median of node degrees in the graph and 0 otherwise. For betweenness centrality and closeness centrality, the label of a node is 1 if the value of the node property is above the mode of the value in

Table 6.2: Macro-F1 and Micro-F1 scores for node classification task using node degree labels

Dataset	Score	Node2vec	LINE	SDNE	HOPE
Wisconsin	Macro-F1	0.6197	0.4645	**0.8820**	0.3480
	Micro-F1	0.6297	0.4887	**0.8847**	0.5339
Squirell	Macro-F1	0.7800	0.5041	**0.9576**	0.8399
	Micro-F1	0.7842	0.5089	**0.9577**	0.8456
Fb-cmu	Macro-F1	0.7230	0.6334	0.8594	**0.8603**
	Micro-F1	0.7259	0.6340	0.8684	**0.8627**

Table 6.3: Macro-F1 and Micro-F1 scores for node classification task using betweenness centrality node labels

Dataset	Score	Node2vec	LINE	SDNE	HOPE
Wisconsin	Macro-F1	0.5261	0.4100	**0.8149**	0.6388
	Micro-F1	0.5342	0.4139	**0.8177**	0.6759
Squirell	Macro-F1	0.5397	0.5130	**0.8030**	0.6956
	Micro-F1	0.5415	0.5134	**0.8035**	0.7035
Fb-cmu	Macro-F1	0.5527	0.5793	0.7939	**0.8079**
	Micro-F1	0.5729	0.5799	0.8042	**0.8106**

Table 6.4: Macro-F1 and Micro-F1 scores for node classification task using closeness centrality node labels

Dataset	Score	Node2vec	LINE	SDNE	HOPE
Wisconsin	Macro-F1	0.7698	0.5239	0.9881	**0.9920**
	Micro-F1	0.7789	0.5281	0.9881	**0.9920**
Squirell	Macro-F1	0.8560	0.5233	**0.9517**	0.8995
	Micro-F1	0.8569	0.5236	**0.9517**	0.9004
Fb-cmu	Macro-F1	0.7435	0.6581	0.8469	**0.8572**
	Micro-F1	0.7456	0.6586	0.8556	**0.8591**

the dataset and 0 otherwise. As the betweenness centrality and closeness centrality of many nodes in the graph can be zero, the mode in these values is zero.

For these classification tasks, we utilized 10 fold cross validation using the logistic regression classifier. Table 6.2, 6.3 and 6.4 depict the results of the classification using labels defined by node properties. As shown in these tables, the Micro-F1 and Macro-F1 scores of the node classification are highly aligned with BCI scores in Table 6.1 on all three datasets with SDNE having the highest F1 scores followed by HOPE, Node2vc and then LINE. These scores also have the trends similar to those of CCI in Table 6.1, with HOPE and SDNE being close top 2 performers, followed by Node2vec and LINE. These node classification results provide strong validation on the accuracy of our BCI and CCI scores.

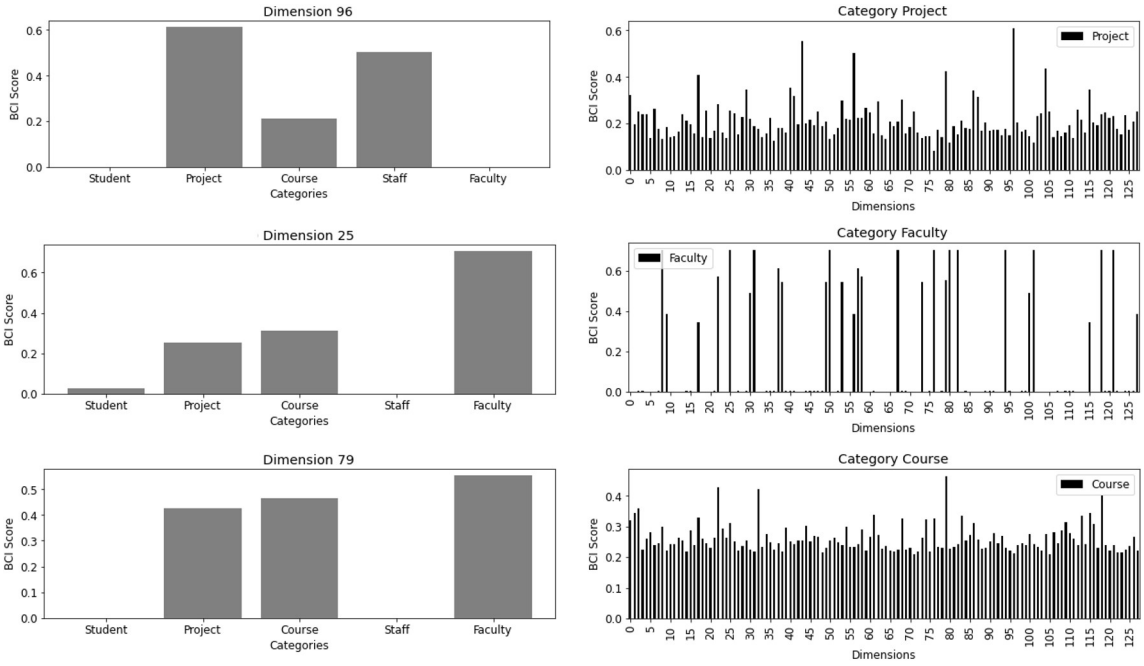


Figure 6.2: Dimension and Category decomposition of BCI scores on the Wisconsin dataset using Node2vec embedding vectors of 128 dimensions: Left side BCI scores of embedding dimensions 96, 25 and 79 over all categories. Right side BCI values over all dimensions for the Project, Faculty and Course category.

6.5.5 Interpreting embedding dimensions with visualization

In Figure 6.2, we use examples to visualize the BCI scores of node embedding dimensions for some categories to provide an interpretation for some dimensions. We also show the distribution of BCI scores for some of the categories. Visualization for the CCI scores can be done in the same way. The sample graph is Wisconsin which consists of 5 categories: Student, Course, Project, Faculty and Staff. The embeddings are generated by Node2vec.

The left side of Figure 6.2 depicts the BCI scores of three sample embedding dimensions (96, 25 and 79) over node categories. Based on the figures, dimension 96 captures the important nodes in three categories Project, Staff and Course with the highest score for Project category. Dimension 25 has the highest score for category Faculty compared to other categories which means that it is an important dimension for representing category Faculty in terms of its central nodes. Dimension 79 has a relatively similar score for three categories.

On the right hand side of Figure 6.2, we show the dimensional decomposition of three sample categories. In the Project category, some embedding dimensions such as 96 and 43 are dominant in representing the central nodes in the category. The important nodes in the Faculty category are much better represented by a small

subset of embedding dimensions, but not by other embedding dimensions. For the Course category, however, the encoding of central nodes in the category is distributed among many embedding dimensions. In general, dimensions with the highest scores for a category are important dimensions in representing central nodes in the category. This type of visualization provides a way to understand the embedding dimensions in terms of whether they capture the important nodes in a category.

6.5.6 Downstream tasks

In this section, we apply the node embedding methods in downstream tasks to see whether there is an agreement between the BCI and CCI scores of each node embedding method and the predictive performance of the node embedding methods in downstream tasks including link prediction and node classification.

Link prediction: One of the main applications of node embeddings is link prediction. In the link prediction task, a classifier is trained on a subset of edges and tested on the ability to predict the existence of edges in a test set. Each edge is represented by an edge embedding vector that is obtained using mean of the embedding vectors of nodes connected by the edge [30]. We applied the obtained node embedding vectors from four methods to the link prediction task. The dataset is split into training and test sets with a 80-20 ratio. In each set, the positive samples are the existing edges and equal number of negative edges are randomly sampled between nodes that are not connected in the dataset, following Node2vec [30] approach. We used Logistic regression as the classification method. Table 6.5 shows the results of the experiment.

As can be seen in the table, for Wisconsin and Squirrel datasets, SDNE has the highest performance among other methods. This is highly aligned with the BCI score of SDNE in those datasets. The Fb-cmu dataset has the best AUC score using HOPE node embeddings which also has the highest BCI scores. We computed the correlation between BCI ranking and AUC score ranking of link prediction using Kendall rank correlation coefficient [387]. On average, the correlation between these two rankings is about 0.8 which shows a strong agreement between link prediction performance of node embedding methods with the BCI scores.

Node classification: We apply the node embedding methods in another downstream task, which is node classification using the true node labels in the datasets. We applied 10-fold cross validation with Logistic Regression as the classification

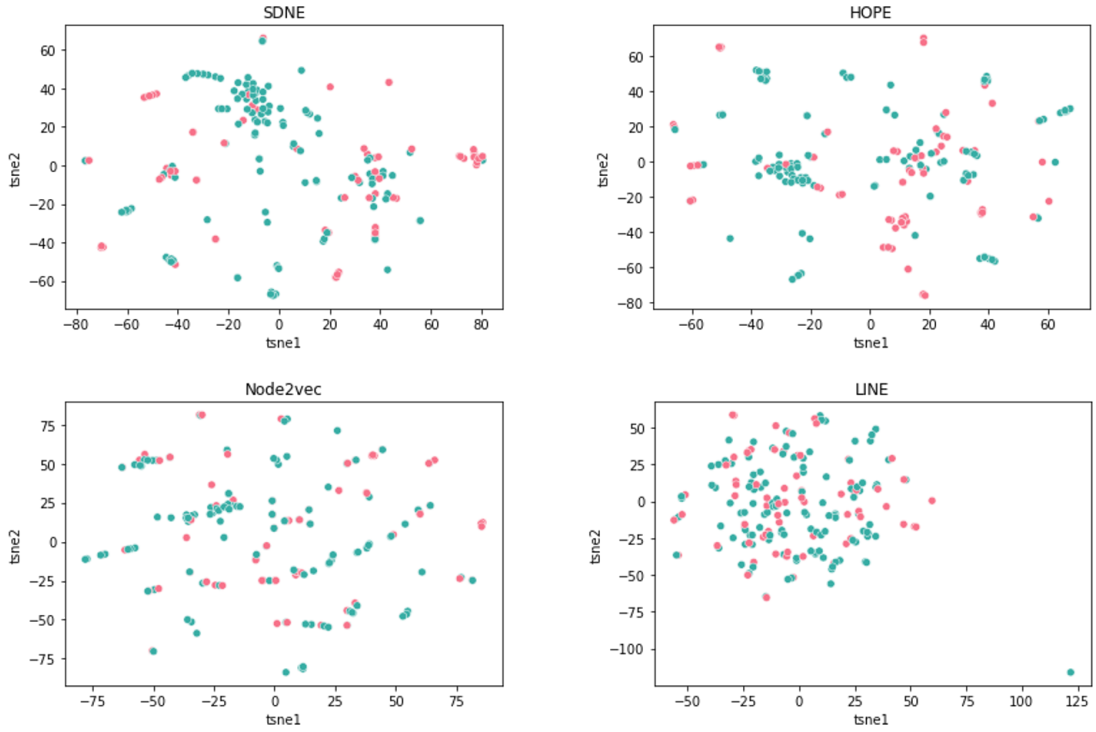


Figure 6.3: Visualization of the embedding of Wisconsin dataset using t-SNE for top two most populated classes.

method for the downstream task. Table 6.6 shows the results of the node classification for each of the node embedding methods on the three datasets.

For the Wisconsin and Squirell datasets, SDNE has the highest node classification scores. The general ranking of the embedding methods in F1-scores is highly aligned with that of these methods using the BCI and CCI scores on these datasets (see Table 6.1). Figure 6.3 is the visualization of node embeddings generated by four embedding methods on Wisconsin dataset in two categories of nodes using t-SNE [336]. In these figures, two classes are represented by green and red colors and it seems that SDNE better captures or distinguishes the two classes compare to other embedding methods in Wisconsin dataset. For the Fb-cmu dataset, Node2vec works best in the node classification task, while SDNE and HOPE have the highest BCI and CCI scores although the BCI and CCI scores of Node2vec are very close to the best ones. We conjecture that the node categories in the Fb-cmu dataset are not as distinguishable as in two other datasets in terms of the node centrality properties[1]. In terms of correlation between BCI scores and node classification scores, the average Kendall rank correlation [387] of BCI and CCI score with node classification scores in Wisconsin and Squirell datasets is around 0.7 which shows a good correlation. However, as expected a weak correlation around 0.4 is found in the Fb-cmu dataset.

[1]The node labels in the Fb-cmu dataset are 1, 2 and 3 and their meanings are not provided.

Table 6.5: Downstream link prediction task auc scores

Dataset	**Node2vec**	**LINE**	**SDNE**	**HOPE**
Wisconsin	0.5888	0.4944	**0.6111**	0.5000
Squirell	0.6766	0.5275	**0.7013**	0.6728
Fb-cmu	0.5934	0.5516	0.6021	**0.6138**

Table 6.6: Downstream node classification Macro-F1 and Micro-F1 scores

Dataset	**Score**	**Node2vec**	**LINE**	**SDNE**	**HOPE**
Wisconsin	Macro-F1	0.2322	0.1249	**0.3190**	0.1279
	Micro-F1	0.4713	0.4546	**0.6134**	0.4703
Squirell	Macro-F1	0.2646	0.1956	**0.4081**	0.3209
	Micro-F1	0.2730	0.1959	**0.4157**	0.3419
Fb-cmu	Macro-F1	**0.4436**	0.3388	0.4283	0.3231
	Micro-F1	**0.6635**	0.5605	0.6549	0.5923

Table 6.7: CCI score using different λ values on the Wisconsin dataset

λ	**Node2vec**	**LINE**	**SDNE**	**HOPE**
1	0.5006	0.4958	0.5350	0.5134
2	0.9606	0.9488	0.9668	0.9653
3	0.9999	0.9999	0.9999	0.9999

6.5.7 Parameter sensitivity

We evaluate the effects of the parameters λ in the interpretability scores. λ is the parameter used in computing the BCI and CCI scores such that $\lambda * n_k$ determines how many top nodes in each embedding dimension are considered in the calculation and can reflect the strictness of the score definition. In order to evaluate the effect of λ, we compute the CCI scores using λ values ranging in $\{1, 2, 3\}$ on the Wisconsin dataset and report the results in Table 6.7. As can be seen in the Table 6.7, increasing λ values results in the CCI scores that are harder to distinguish among embedding methods. For instance, when $\lambda = 2$ the CCI scores are smoother than when $\lambda = 1$. When $\lambda = 3$, it is not possible to distinguish among different embedding methods based on the CCI scores. The reason for this is that as λ increases more nodes from each embedding dimension are included in the interpretability score computation and as a result the interpretability scores are less distinguishable. Therefore, it is important not to choose a big value for parameter λ. We use $\lambda = 1$ in other experiments.

6.6 Summary

We proposed two new interpretability measures, betweenness centrality importance and closeness centrality importance scores, for quantifying the interpretability of node embedding methods. We used the measures to compute the interpretability scores for four graph embedding methods on three graph datasets. These scores are validated by comparing the ranking of embedding methods using the proposed interpretability scores with that using a node classification task whose class labels are defined according to node centrality measures. We also showed that the proposed measures can be used to explain the graph embedding dimensions via visualization. Another interesting finding is that embeddings with high BCI and CCI scores tend to do well in downstream link prediction and node classification tasks.

As future work, we will investigate how these scores can be used in developing better graph embedding methods. In addition, other properties of graphs such as the neighborhood structure of a graph could be utilized in defining new interpretability scores. In this study, we focused on intepretability measures for static graph embedding methods. Dynamic graphs are more challenging because of the dynamic characteristics of the graphs. We would like to investigate intepretability scores for dynamic graph embedding methods in the future. In addition, we will also investigate how to design an interpretability space of graph embedding vectors by using rotation matrix algorithms.

Chapter 7

Conclusions and Future Directions

7.1 Summary of Contributions

Graph embedding has been a very active area of study recently. Different methods were proposed to represent graphs effectively. Most of these methods are developed for static graphs. However, the majority of the graphs in real-world are dynamic and more challenging to represent. In this dissertation, we first surveyed the graph embedding methods in both static and dynamic setting. Then, we introduced three dynamic graph embedding methods to capture evolving characteristics of graphs. Finally, we proposed two interpretability measures for graph embedding methods. A summary of our contributions are given below. In a survey on graph embedding methods, we review both traditional and GNN-based graph embedding methods. The distinction between our surveys and others are as follows:

- We put together the graph embedding methods in both traditional and GNN-based categories for both static and dynamic graphs and include over 300 papers consisting of papers published in reputable venues in data mining, machine learning and artificial intelligence [1] since 2017 until the time of this submission, and also influential papers with high citations published before 2017.

- We summarize a number of limitations of GNN-based methods and the proposed solutions to these limitations until the time of submission. These limitations are expressive power, over-smoothing, scalability, over-squashing, capturing long-range dependencies, design space, neglecting substructures, ho-

[1]The venues include KDD, ICLR, ICML, NeurIPS, AAAI, IJCAI, ICDM, WWW, WSDM, DSAA, SDM, CIKM.

mophily assumptions, and catastrophic forgetting. Such a summary was not provided in previous surveys.

- We provide a list of the real-world applications of GNN-based methods that are deployed in production.
- We suggest a list of future research directions including new ones that are not covered by previous surveys.

Then, to address the challenges and limitations in modeling dynamic graphs, we developed three dynamic graph embedding methods. Our first method Dynnde2vec is a fast and scalable traditional dynamic graph embedding method that adapt the well-known node2vec model to dynamic settings. Our major contributions in this collaboration are as follows:

- We develop a dynamic embedding method *dynnode2vec* that captures evolving patterns in large dynamic networks
- *dynnode2vec* is a fast and accurate method for dynamic graph embedding
- We evaluate the performance of our method in variety of tasks including link prediction, node classification and anomaly detection on large real-world graphs

Secondly, we developed LSTM-Node2vec which combines benefits of LSTM model and node2vec. In this work, following contributions are achieved:

- We propose LSTM-node2vec, a novel dynamic embedding method that captures temporal changes with LSTM and then the learned parameters are transferred into node2vec to incorporate the local structure of each graph.
- We train an autoencoder LSTM model with temporal walks to capture the history of nodes over time which is the first study to consider temporal walks as the input to an LSTM model.
- We evaluate our method on three main data mining tasks including anomaly detection, link prediction and node classification.

Third, TGR-Clique is introduced which is a GNN-based temporal graph embedding methods that considers the effects of neighbors of a node in the maximal cliques in the node embedding. This works has the following major contributions:

- We present a novel temporal graph representation learning method based on maximal cliques for both inductive and transductive learning tasks.
- Our method generates temporal random walks on maximal cliques of the graph to capture the evolution of the graph over time. By leveraging maximal cliques in random walk generation, we identify the most connected neighbors of a node to consider in its representation.
- We evaluate the performance of our method in link prediction and node classification tasks on seven real world datasets against five state-of-the-art models to show that our method achieves better or comparable predictive performance while being faster than other methods compared. In addition, we perform ablation studies to show the importance of each feature of the proposed method.

Finally, motivated by the lack of proper metrics to quantify the interpretability of the graph embeddings, we proposed two metrics for this purpose with the following contributions:

- We propose two interpretability measures for node embedding methods using the betweenness centrality and closeness centrality properties of the nodes in a graph.
- We empirically validate the new interpretability measures by comparing the ranking of embedding methods using our interpretability scores with that using a node classification task whose class labels are defined based on node centrality.
- We illustrate how to use the proposed measures to interpret embedding dimensions.
- We investigate whether there is a relationship between the proposed interpretability scores of the embeddings and the predictive performance of a downstream classification task with the embeddings.

7.2 Future Directions

Although numerous techniques were developed for graph representation learning, there are still significant issues and challenges that need to be addressed. We will discuss some of them here.

Proposing solutions for limitations of GNNs. GNNs have several limitations such as expressive power, scalability and over-squashing. Different solutions have been proposed for these issues. However, some of them, such as over-squashing, were newly discovered and solutions to them are still preliminary, which have potential for further research. In addition, the proposed solutions for the limitations of GNNs were all for static GNNs and not much work has been done on solving the problems of dynamic and spatial-temporal GNNs. Furthermore, there might be new issues in dynamic settings that are not discovered yet and are worth studying. Finally, by applying GNNs to different real-world problems and datasets, new drawbacks of GNNs may be discovered, which will be interesting to study and solve.

Studying the theory side of GNNs. GNNs have been very successful in different applications, and researchers have tried to understand the theoretical aspects of GNNs success. However, the theoretical analyses of GNNs models in terms of optimization properties and generalization across graph sizes are less understood. In [388], the first steps in understanding the global convergence of gradient descent in GNNs are studied. In addition, in [389], it is shown that GNNs that are trained on some graph distributions cannot generalize to larger unseen graphs. Therefore, future work is needed to understand these topics fully.

Defining new GNNs by employing differential equations. Partial differential equations (PDE) are used to model physical phenomena [390]. Recently, PDEs have been used to model the information propagation in graphs and design new GNN models [306, 144]. One of the advantages of these new GNNs is that they address the over-smoothing limitations of GNNs. Therefore, it is an interesting future work to investigate what other GNN models can be proposed using differential equations and if these models are more powerful than other GNNs.

Including domain-knowledge into GNNs. The domain knowledge is the information about a specific problem that may not be available for a machine learning model. This knowledge can be included in the models by changing the input, loss function, and model architecture [391]. For instance, we can enrich the input data by considering additional relationships or constraints on existing relationships among the entities in the data. A few works incorporate domain knowledge into GNNs, such as [392, 393], but this area is still new and has the potential for further research.

Limited training data labels. Training neural networks with limited training data has always been challenging, which is also true for training the graph neural

nets. Solutions such as self-supervised learning, data augmentation, and contrastive learning, have been proposed to solve the label scarcity problem and are employed in graph neural nets. However, there are still opportunities for further studying these types of learning in GNNs in both static and dynamic settings.

Transferring advances in deep learning models to GNNs. Graph neural nets use deep learning techniques. Any advances and new models that are proposed in deep learning can be adapted in graph neural nets. Specially, computer vision and natural language processing are very active areas and the new models that are developed for images, videos and texts can be studied in graphs as well. For instance, ideas from video representation learning might be useful in dynamic graph learning because the concept of objects moving in a video's frames is similar to nodes changing over time.

More applications. GNNs achieved great success in application in many domains such as social networks, financial networks, and protein structures. Some of the newer applications of GNNs are in electrical power grid monitoring [394], drug over-prescription prediction [395], and paper publication prediction [396]. Therefore, it is interesting to investigate the effectiveness of GNNs in other applications.

Bibliography

[1] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[2] S. Khoshraftar and A. An, "A survey on graph representation learning methods," *arXiv preprint arXiv:2204.01855*, 2022.

[3] P. Goyal, N. Kamra, X. He, and Y. Liu, "Dyngem: Deep embedding method for dynamic graphs," *3rd International Workshop on Representation Learning for Graphs (ReLiG), IJCAI 2017*, 2017.

[4] P. Goyal, S. Rokka Chhetri, N. Mehrabi, E. Ferrara, and A. Canedo, "Dynamicgem: A library for dynamic graph embedding methods," *arXiv preprint arXiv:1811.10734*, 2018.

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

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

[7] S. Khoshraftar, S. Mahdavi, A. An, Y. Hu, and J. Liu, "Dynamic graph embedding via lstm history tracking," in *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 119–127, IEEE, 2019.

[8] S. Khoshraftar, A. An, and N. Babanejad, "Temporal graph representation learning via maximal cliques," in *2022 IEEE International Conference on Big Data*, IEEE, 2022.

[9] Z. Liu, C. Huang, Y. Yu, and J. Dong, "Motif-preserving dynamic attributed network embedding," in *Proceedings of the Web Conference 2021*, pp. 1629–1638, 2021.

[10] Y. Wang, Y.-Y. Chang, Y. Liu, J. Leskovec, and P. Li, "Inductive representation learning in temporal networks via causal anonymous walks," in *International Conference on Learning Representations*, 2020.

[11] S. Khoshraftar, S. Mahdavi, and A. An, "Centrality-based interpretability measures for graph embeddings," in *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–10, IEEE, 2021.

[12] L. K. Şenel, I. Utlu, V. Yücesoy, A. Koc, and T. Cukur, "Semantic structure and interpretability of word embeddings," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, no. 10, pp. 1769–1779, 2018.

[13] H. Luo, Z. Liu, H. Luan, and M. Sun, "Online learning of interpretable word embeddings," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1687–1692, 2015.

[14] R. Bamler and S. Mandt, "Dynamic word embeddings," in *International conference on Machine learning*, pp. 380–389, PMLR, 2017.

[15] Y. Kim, Y.-I. Chiu, K. Hanaki, D. Hegde, and S. Petrov, "Temporal analysis of language through neural language models," *arXiv preprint arXiv:1405.3515*, 2014.

[16] P. Wang, Y. Qian, F. K. Soong, L. He, and H. Zhao, "Learning distributed word representations for bidirectional lstm recurrent neural network," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 527–533, 2016.

[17] J. Li, M.-T. Luong, and D. Jurafsky, "A hierarchical neural autoencoder for paragraphs and documents," *arXiv preprint arXiv:1506.01057*, 2015.

[18] Y. Zhang, Q. Liu, and L. Song, "Sentence-state lstm for text representation," *arXiv preprint arXiv:1805.02474*, 2018.

[19] Z. Lu, J. Wahlström, and A. Nehorai, "Community detection in complex networks via clique conductance," *Scientific reports*, vol. 8, no. 1, pp. 1–16, 2018.

[20] M. Adraoui, A. Retbi, M. K. Idrissi, and S. Bennani, "Maximal cliques based method for detecting and evaluating learning communities in social networks," *Future Generation Computer Systems*, vol. 126, pp. 1–14, 2022.

[21] K. Biswas, V. Muthukkumarasamy, and E. Sithirasenan, "Maximal clique based clustering scheme for wireless sensor networks," in *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pp. 237–241, IEEE, 2013.

[22] B. Zhang, B.-H. Park, T. Karpinets, and N. F. Samatova, "From pull-down data to protein interaction networks and complexes with biological relevance," *Bioinformatics*, vol. 24, no. 7, pp. 979–986, 2008.

[23] O. Rokhlenko, Y. Wexler, and Z. Yakhini, "Similarities and differences of gene expression in yeast stress conditions," *Bioinformatics*, vol. 23, no. 2, pp. e184–e190, 2007.

[24] J. Chang, S. Gerrish, C. Wang, J. L. Boyd-Graber, and D. M. Blei, "Reading tea leaves: How humans interpret topic models," in *Advances in neural information processing systems*, pp. 288–296, 2009.

[25] B. Murphy, P. Talukdar, and T. Mitchell, "Learning effective and interpretable semantic models using non-negative sparse embedding," in *Proceedings of COLING 2012*, pp. 1933–1950, 2012.

[26] Q. Wu, X. Qi, E. Fuller, and C.-Q. Zhang, "follow the leader: A centrality guided clustering and its application to social network analysis," *The Scientific World Journal*, vol. 2013, 2013.

[27] C. Tong, J. Niu, B. Dai, and Z. Xie, "A novel complex networks clustering algorithm based on the core influence of nodes," *The Scientific World Journal*, vol. 2014, 2014.

[28] S. Ahajjam, M. El Haddad, and H. Badir, "A new scalable leader-community detection approach for community detection in social networks," *Social Networks*, vol. 54, pp. 41–49, 2018.

[29] J. J. Whang, D. F. Gleich, and I. S. Dhillon, "Overlapping community detection using neighborhood-inflated seed expansion," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 5, pp. 1272–1284, 2016.

[30] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD*, pp. 855–864, ACM, 2016.

[31] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathe-*

matical statistics and probability, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.

[32] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD*, pp. 701–710, ACM, 2014.

[33] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proceedings of the 22nd international conference on World Wide Web*, pp. 37–48, 2013.

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

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

[36] Y. Yin and Z. Wei, "Scalable graph embeddings via sparse transpose proximities," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1429–1437, 2019.

[37] H. Chen, B. Perozzi, Y. Hu, and S. Skiena, "Harp: Hierarchical representation learning for networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[38] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th WWW*, pp. 1067–1077, International World Wide Web Conferences Steering Committee, 2015.

[39] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD*, pp. 1225–1234, ACM, 2016.

[40] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

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

[42] S. Ivanov and E. Burnaev, "Anonymous walk embeddings," in *International conference on machine learning*, pp. 2186–2195, PMLR, 2018.

[43] Y.-A. Lai, C.-C. Hsu, W. H. Chen, M.-Y. Yeh, and S.-D. Lin, "Prune: Preserving proximity and global ranking for network embedding," *Advances in neural information processing systems*, vol. 30, 2017.

[44] S. Abu-El-Haija, B. Perozzi, R. Al-Rfou, and A. A. Alemi, "Watch your step: Learning node embeddings via graph attention," *Advances in neural information processing systems*, vol. 31, 2018.

[45] F. Nie, W. Zhu, and X. Li, "Unsupervised large graph embedding," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[46] C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao, "Scalable graph embedding for asymmetric proximity," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.

[47] Y. Li, C. Sha, X. Huang, and Y. Zhang, "Community detection in attributed graphs: An embedding approach," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[48] L. Du, Z. Lu, Y. Wang, G. Song, Y. Wang, and W. Chen, "Galaxy network embedding: A hierarchical community structure preserving approach.," in *IJCAI*, pp. 2079–2085, 2018.

[49] X. Shen, S. Pan, W. Liu, Y.-S. Ong, and Q.-S. Sun, "Discrete network embedding," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 3549–3555, 2018.

[50] H. Gao and H. Huang, "Deep attributed network embedding," in *Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[51] Z. Zhang, P. Cui, H. Li, X. Wang, and W. Zhu, "Billion-scale network embedding with iterative random projection," in *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 787–796, IEEE, 2018.

[52] H. Wang, E. Chen, Q. Liu, T. Xu, D. Du, W. Su, and X. Zhang, "A united approach to learning sparse attributed network embedding," in *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 557–566, IEEE, 2018.

[53] H. Yang, S. Pan, P. Zhang, L. Chen, D. Lian, and C. Zhang, "Binarized attributed network embedding," in *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 1476–1481, IEEE, 2018.

[54] X. Huang, J. Li, and X. Hu, "Label informed attributed network embedding," in *Proceedings of the tenth ACM international conference on web search and data mining*, pp. 731–739, 2017.

[55] A. Tsitsulin, D. Mottin, P. Karras, and E. Müller, "Verse: Versatile graph embeddings from similarity measures," in *Proceedings of the 2018 world wide web conference*, pp. 539–548, 2018.

[56] T. Huang, L. Zhou, L. Wang, G. Du, and K. Lü, "Attributed network embedding with community preservation," in *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 334–343, IEEE, 2020.

[57] F. Jiang, L. He, Y. Zheng, E. Zhu, J. Xu, and P. S. Yu, "On spectral graph embedding: A non-backtracking perspective and graph approximation," in *Proceedings of the 2018 SIAM International Conference on Data Mining*, pp. 324–332, SIAM, 2018.

[58] X. Huang, J. Li, and X. Hu, "Accelerated attributed network embedding," in *Proceedings of the 2017 SIAM international conference on data mining*, pp. 633–641, SIAM, 2017.

[59] W. Xiao, H. Zhao, V. W. Zheng, and Y. Song, "Vertex-reinforced random walk for network embedding," in *Proceedings of the 2020 SIAM International Conference on Data Mining*, pp. 595–603, SIAM, 2020.

[60] H. Zhu and P. Koniusz, "Refine: Random range finder for network embedding," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 3682–3686, 2021.

[61] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[62] L. Ribeiro, P. Saverese, and D. Figueiredo, "struc2vec: Learning node representations from structural identity," in *KDD*, 2017.

[63] T. He, L. Gao, J. Song, X. Wang, K. Huang, and Y. Li, "Sneq: semi-supervised attributed network embedding with attention-based quantisation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 4091–4098, 2020.

[64] Y. Wang, G. Pan, Y. Yao, H. Tong, H. Yang, F. Xu, and J. Lu, "Bringing order to network embedding: A relative ranking based approach," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 1585–1594, 2020.

[65] H. Chen, S. F. Sultan, Y. Tian, M. Chen, and S. Skiena, "Fast and accurate network embeddings via very sparse random projection," in *Proceedings of the 28th ACM international conference on information and knowledge management*, pp. 399–408, 2019.

[66] T. Lyu, Y. Zhang, and Y. Zhang, "Enhancing the network embedding quality with structural similarity," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 147–156, 2017.

[67] S. Chanpuriya and C. Musco, "Infinitewalk: Deep network embeddings as laplacian embeddings with a nonlinearity," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1325–1333, 2020.

[68] S. Chanpuriya, C. Musco, K. Sotiropoulos, and C. Tsourakakis, "Node embeddings and exact low-rank representations of complex networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13185–13198, 2020.

[69] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *Proceedings of the eleventh ACM international conference on web search and data mining*, pp. 459–467, 2018.

[70] X. Zhang, K. Xie, S. Wang, and Z. Huang, "Learning based proximity matrix factorization for node embedding," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2243–2253, 2021.

[71] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, and W. Zhu, "Arbitrary-order proximity preserved network embedding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2778–2786, 2018.

[72] J. Qiu, Y. Dong, H. Ma, J. Li, C. Wang, K. Wang, and J. Tang, "Netsmf: Large-scale network embedding as sparse matrix factorization," in *The World Wide Web Conference*, pp. 1509–1520, 2019.

[73] A. Epasto and B. Perozzi, "Is a single embedding enough? learning node representations that capture multiple social contexts," in *The world wide web conference*, pp. 394–404, 2019.

[74] R. Al-Rfou, B. Perozzi, and D. Zelle, "Ddgk: Learning graph representations for deep divergence graph kernels," in *The World Wide Web Conference*, pp. 37–48, 2019.

[75] R. Brochier, A. Guille, and J. Velcin, "Global vectors for node representations," in *The World Wide Web Conference*, pp. 2587–2593, 2019.

[76] A. K. Bhowmick, K. Meneni, M. Danisch, J.-L. Guillaume, and B. Mitra, "Louvainne: Hierarchical louvain method for high quality and scalable network embedding," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, pp. 43–51, 2020.

[77] R. A. Rossi, N. K. Ahmed, E. Koh, S. Kim, A. Rao, and Y. Abbasi-Yadkori, "A structural graph representation learning framework," in *Proceedings of the 13th international conference on web search and data mining*, pp. 483–491, 2020.

[78] Z. Meng, S. Liang, H. Bao, and X. Zhang, "Co-embedding attributed networks," in *Proceedings of the twelfth ACM international conference on web search and data mining*, pp. 393–401, 2019.

[79] C. Yang, M. Sun, Z. Liu, and C. Tu, "Fast network embedding enhancement via high order proximity approximation.," in *IJCAI*, vol. 17, pp. 3894–3900, 2017.

[80] Z. Huang, A. Silva, and A. Singh, "A broader picture of random-walk based graph embedding," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 685–695, 2021.

[81] X. Liu, T. Murata, K.-S. Kim, C. Kotarasu, and C. Zhuang, "A general view for network embedding as matrix factorization," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pp. 375–383, 2019.

[82] F. Heidari and M. Papagelis, "Evolving network representation learning based on random walks," *Applied network science*, vol. 5, no. 1, pp. 1–38, 2020.

[83] S. Mahdavi, S. Khoshraftar, and A. An, "Dynamic joint variational graph autoencoders," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 385–401, Springer, 2019.

[84] P. Goyal, S. R. Chhetri, and A. Canedo, "dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowledge-Based Systems*, vol. 187, p. 104816, 2020.

[85] Y. Zuo, G. Liu, H. Lin, J. Guo, X. Hu, and J. Wu, "Embedding temporal network via neighborhood formation," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2857–2866, 2018.

[86] L. Zhou, Y. Yang, X. Ren, F. Wu, and Y. Zhuang, "Dynamic network embedding by modeling triadic closure process," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[87] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *International Conference on Learning Representations*, 2019.

[88] H. Huang, Z. Fang, X. Wang, Y. Miao, and H. Jin, "Motif-preserving temporal network embedding," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 1237–1243.

[89] L. Du, Y. Wang, G. Song, Z. Lu, and J. Wang, "Dynamic network embedding: An extended approach for skip-gram based network embedding.," in *IJCAI*, vol. 2018, pp. 2086–2092, 2018.

[90] J. Ma, Q. Zhang, J. Lou, L. Xiong, and J. C. Ho, "Temporal network embedding via tensor factorization," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 3313–3317, 2021.

[91] Z. Qiu, W. Hu, J. Wu, W. Liu, B. Du, and X. Jia, "Temporal network embedding with high-order nonlinear information," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 5436–5443, 2020.

[92] M. Beladev, L. Rokach, G. Katz, I. Guy, and K. Radinsky, "tdgraphembed: Temporal dynamic graph-level embedding," in *Proceedings of the 29th ACM*

International Conference on Information & Knowledge Management, pp. 55–64, 2020.

[93] Z. Liu, C. Huang, Y. Yu, P. Song, B. Fan, and J. Dong, "Dynamic representation learning for large-scale attributed networks," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 1005–1014, 2020.

[94] Z. Zhang, P. Cui, J. Pei, X. Wang, and W. Zhu, "Timers: Error-bounded svd restart on dynamic networks," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[95] Y. Lu, X. Wang, C. Shi, P. S. Yu, and Y. Ye, "Temporal network embedding with micro-and macro-dynamics," in *Proceedings of the 28th ACM international conference on information and knowledge management*, pp. 469–478, 2019.

[96] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu, "Attributed network embedding for learning in a dynamic environment," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 387–396, 2017.

[97] U. Sharan and J. Neville, "Temporal-relational classifiers for prediction in evolving domains," in *2008 Eighth IEEE International Conference on Data Mining*, pp. 540–549, IEEE, 2008.

[98] U. Singer, I. Guy, and K. Radinsky, "Node embedding over temporal graphs," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 4605–4612, 2019.

[99] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2672–2681, 2018.

[100] J. Zhu, Q. Xie, and E. J. Chin, "A hybrid time-series link prediction framework for large social network," in *International Conference on Database and Expert Systems Applications*, pp. 345–359, Springer, 2012.

[101] L. Yao, L. Wang, L. Pan, and K. Yao, "Link prediction based on common-neighbors for dynamic social network," *Procedia Computer Science*, vol. 83, pp. 82–89, 2016.

[102] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[103] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow, "Gated graph sequence neural networks," in *Proceedings of ICLR'16*, 2016.

[104] F. Gu, H. Chang, W. Zhu, S. Sojoudi, and L. El Ghaoui, "Implicit graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11984–11995, 2020.

[105] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

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

[107] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[108] D. Beaini, S. Passaro, V. Létourneau, W. Hamilton, G. Corso, and P. Liò, "Directional graph networks," in *International Conference on Machine Learning*, pp. 748–758, PMLR, 2021.

[109] X. Liu, W. Jin, Y. Ma, Y. Li, H. Liu, Y. Wang, M. Yan, and J. Tang, "Elastic graph neural networks," in *International Conference on Machine Learning*, pp. 6837–6849, PMLR, 2021.

[110] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International conference on machine learning*, pp. 6861–6871, PMLR, 2019.

[111] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[112] G. Wang, R. Ying, J. Huang, and J. Leskovec, "Multi-hop attention graph neural networks," in *IJCAI*, 2021.

[113] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning*, pp. 1263–1272, PMLR, 2017.

[114] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[115] M. Brockschmidt, "Gnn-film: Graph neural networks with feature-wise linear modulation," in *International Conference on Machine Learning*, pp. 1144–1152, PMLR, 2020.

[116] D. Zambon, C. Alippi, and L. Livi, "Graph random neural features for distance-preserving graph representations," in *International Conference on Machine Learning*, pp. 10968–10977, PMLR, 2020.

[117] V. G. Satorras, E. Hoogeboom, and M. Welling, "E (n) equivariant graph neural networks," in *International Conference on Machine Learning*, pp. 9323–9332, PMLR, 2021.

[118] H. Zhu, F. Feng, X. He, X. Wang, Y. Li, K. Zheng, and Y. Zhang, "Bilinear graph neural network with neighbor interactions," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pp. 1452–1458, 2021.

[119] K. Zhou, Q. Song, X. Huang, D. Zha, N. Zou, and X. Hu, "Multi-channel graph neural networks," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pp. 1352–1358, 2021.

[120] B. Yan, C. Wang, G. Guo, and Y. Lou, "Tinygnn: Learning efficient graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1848–1856, 2020.

[121] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *International Conference on Learning Representations*, 2018.

[122] R. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, "Relational pooling for graph representations," in *International Conference on Machine Learning*, pp. 4663–4673, PMLR, 2019.

[123] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 4602–4609, 2019.

[124] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," *Advances in neural information processing systems*, vol. 32, 2019.

[125] Z. Chen, S. Villar, L. Chen, and J. Bruna, "On the equivalence between graph isomorphism testing and function approximation with gnns," *Advances in neural information processing systems*, vol. 32, 2019.

[126] W. Azizian *et al.*, "Expressive power of invariant and equivariant graph neural networks," in *International Conference on Learning Representations*, 2020.

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

[128] M. Balcilar, P. Héroux, B. Gauzere, P. Vasseur, S. Adam, and P. Honeine, "Breaking the limits of message passing graph neural networks," in *International Conference on Machine Learning*, pp. 599–608, PMLR, 2021.

[129] R. Sato, M. Yamada, and H. Kashima, "Random features strengthen graph neural networks," in *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pp. 333–341, SIAM, 2021.

[130] P. A. Papp, K. Martinkus, L. Faber, and R. Wattenhofer, "Dropgnn: random dropouts increase the expressiveness of graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[131] H. Wang, H. Yin, M. Zhang, and P. Li, "Equivariant and stable positional encoding for more powerful graph neural networks," in *International Conference on Learning Representations*, 2021.

[132] A. Wijesinghe and Q. Wang, "A new perspective on" how graph neural networks go beyond weisfeiler-lehman?"," in *International Conference on Learning Representations*, 2021.

[133] M. Zhang and P. Li, "Nested graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[134] J. You, J. M. Gomes-Selman, R. Ying, and J. Leskovec, "Identity-aware graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 10737–10745, 2021.

[135] G. Dasoulas, L. D. Santos, K. Scaman, and A. Virmaux, "Coloring graph neural networks for node disambiguation," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pp. 2126–2132, 2021.

[136] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *International Conference on Learning Representations*, 2018.

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

[138] L. Zhao and L. Akoglu, "Pairnorm: Tackling oversmoothing in gnns," in *International Conference on Learning Representations*, 2019.

[139] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *International Conference on Learning Representations*, 2019.

[140] K. Zhou, X. Huang, Y. Li, D. Zha, R. Chen, and X. Hu, "Towards deeper graph neural networks with differentiable group normalization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4917–4928, 2020.

[141] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, "Graph random neural networks for semi-supervised learning on graphs," *Advances in neural information processing systems*, vol. 33, pp. 22092–22103, 2020.

[142] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *International Conference on Machine Learning*, pp. 1725–1735, PMLR, 2020.

[143] A. Hasanzadeh, E. Hajiramezanali, S. Boluki, M. Zhou, N. Duffield, K. Narayanan, and X. Qian, "Bayesian graph neural networks with adaptive connection sampling," in *International conference on machine learning*, pp. 4094–4104, PMLR, 2020.

[144] M. Eliasof, E. Haber, and E. Treister, "Pde-gcn: Novel architectures for graph neural networks motivated by partial differential equations," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[145] H. Zeng, M. Zhang, Y. Xia, A. Srivastava, A. Malevich, R. Kannan, V. Prasanna, L. Jin, and R. Chen, "Decoupling the depth and scope of graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[146] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

[147] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," in *International Conference on Learning Representations*, 2018.

[148] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," *Advances in neural information processing systems*, vol. 32, 2019.

[149] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *International Conference on Learning Representations*, 2019.

[150] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *International Conference on Machine Learning*, pp. 942–950, PMLR, 2018.

[151] M. Chen, Z. Wei, B. Ding, Y. Li, Y. Yuan, X. Du, and J.-R. Wen, "Scalable graph neural networks via bidirectional propagation," *Advances in neural information processing systems*, vol. 33, pp. 14556–14566, 2020.

[152] G. Li, M. Müller, B. Ghanem, and V. Koltun, "Training graph neural networks with 1000 layers," in *International conference on machine learning*, pp. 6437–6449, PMLR, 2021.

[153] M. Ding, K. Kong, J. Li, C. Zhu, J. Dickerson, F. Huang, and T. Goldstein, "Vq-gnn: A universal framework to scale up graph neural networks using vector quantization," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[154] K.-L. Yao and W.-J. Li, "Blocking-based neighbor sampling for large-scale graph neural networks," in *International Joint Conference on Artificial Intelligence*, 2021.

[155] T. Chen, Y. Sui, X. Chen, A. Zhang, and Z. Wang, "A unified lottery ticket hypothesis for graph neural networks," in *International Conference on Machine Learning*, pp. 1695–1706, PMLR, 2021.

[156] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, "Beyond homophily in graph neural networks: Current limitations and effective designs," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7793–7804, 2020.

[157] E. Chien, J. Peng, P. Li, and O. Milenkovic, "Adaptive universal generalized pagerank graph neural network," in *International Conference on Learning Representations*, 2020.

[158] S. Suresh, V. Budde, J. Neville, P. Li, and J. Ma, "Breaking the limit of graph neural networks by improving the assortativity of graphs with local mixing patterns," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1541–1551, 2021.

[159] L. Yang, M. Li, L. Liu, C. Wang, X. Cao, Y. Guo, *et al.*, "Diverse message passing for attribute with heterophily," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[160] J. Zhu, R. A. Rossi, A. Rao, T. Mai, N. Lipka, N. K. Ahmed, and D. Koutra, "Graph neural networks with heterophily," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 11168–11176, 2021.

[161] D. Jin, Z. Yu, C. Huo, R. Wang, X. Wang, D. He, and J. Han, "Universal graph convolutional networks," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[162] M. Liu, Z. Wang, and S. Ji, "Non-local graph neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[163] T. Yang, Y. Wang, Z. Yue, Y. Yang, Y. Tong, and J. Bai, "Graph pointer neural networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[164] T. Wang, R. Wang, D. Jin, D. He, and Y. Huang, "Powerful graph convolutioal networks with adaptive propagation mechanism for homophily and heterophily," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[165] Z. Fang, L. Xu, G. Song, Q. Long, and Y. Zhang, "Polarized graph neural networks," in *Proceedings of the ACM Web Conference 2022*, pp. 1404–1413, 2022.

[166] L. Du, X. Shi, Q. Fu, X. Ma, H. Liu, S. Han, and D. Zhang, "Gbk-gnn: Gated bi-kernel graph neural networks for modeling both homophily and heterophily," in *Proceedings of the ACM Web Conference 2022*, pp. 1550–1558, 2022.

[167] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," in *International Conference on Learning Representations*, 2019.

[168] G. Bouritsas, F. Frasca, S. P. Zafeiriou, and M. Bronstein, "Improving graph neural network expressivity via subgraph isomorphism counting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[169] C. Bodnar, F. Frasca, Y. Wang, N. Otter, G. F. Montufar, P. Lio, and M. Bronstein, "Weisfeiler and lehman go topological: Message passing simplicial networks," in *International Conference on Machine Learning*, pp. 1026–1037, PMLR, 2021.

[170] Q. Long, Y. Jin, G. Song, Y. Li, and W. Lin, "Graph structural-topic neural network," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1065–1073, 2020.

[171] Z. Chen, L. Chen, S. Villar, and J. Bruna, "Can graph neural networks count substructures?," *Advances in neural information processing systems*, vol. 33, pp. 10383–10395, 2020.

[172] Q. Long, Y. Jin, Y. Wu, and G. Song, "Theoretically improving graph neural networks via anonymous walk graph kernels," in *Proceedings of the Web Conference 2021*, pp. 1204–1214, 2021.

[173] E. Alsentzer, S. Finlayson, M. Li, and M. Zitnik, "Subgraph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 8017–8029, 2020.

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

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

[176] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[177] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[178] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[179] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *Proceedings of the 2018 World Wide Web Conference*, pp. 499–508, 2018.

[180] G. Pan, Y. Yao, H. Tong, F. Xu, and J. Lu, "Unsupervised attributed network embedding via cross fusion," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pp. 797–805, 2021.

[181] Y. Dong, K. Ding, B. Jalaian, S. Ji, and J. Li, "Adagnn: Graph neural networks with adaptive frequency response filter," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 392–401, 2021.

[182] J. B. Lee, R. A. Rossi, X. Kong, S. Kim, E. Koh, and A. Rao, "Graph convolutional networks with motif-based attention," in *Proceedings of the 28th ACM international conference on information and knowledge management*, pp. 499–508, 2019.

[183] Q. Li, X. Zhang, H. Liu, Q. Dai, and X.-M. Wu, "Dimensionwise separable 2-d graph convolution for unsupervised and semi-supervised learning on graphs," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 953–963, 2021.

[184] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *International Conference on Neural Information Processing*, pp. 362–373, Springer, 2018.

[185] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph wavenet for deep spatial-temporal graph modeling.," in *IJCAI*, 2019.

[186] M. Li and Z. Zhu, "Spatial-temporal fusion graph neural networks for traffic flow forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 4189–4196, 2021.

[187] J. Ye, L. Sun, B. Du, Y. Fu, X. Tong, and H. Xiong, "Co-prediction of multiple transportation demands based on deep spatio-temporal neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 305–313, 2019.

[188] W. Ouyang, X. Zhang, L. Li, H. Zou, X. Xing, Z. Liu, and Y. Du, "Deep spatio-temporal neural networks for click-through rate prediction," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2078–2086, 2019.

[189] Y.-a. Geng, Q. Li, T. Lin, L. Jiang, L. Xu, D. Zheng, W. Yao, W. Lyu, and Y. Zhang, "Lightnet: A dual spatiotemporal encoder network model for lightning prediction," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2439–2447, 2019.

[190] H. Lin, R. Bai, W. Jia, X. Yang, and Y. You, "Preserving dynamic attention for long-term spatial-temporal prediction," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 36–46, 2020.

[191] R. Dai, S. Xu, Q. Gu, C. Ji, and K. Liu, "Hybrid spatio-temporal graph convolutional network: Improving traffic prediction with navigation data," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3074–3082, 2020.

[192] L. Han, B. Du, L. Sun, Y. Fu, Y. Lv, and H. Xiong, "Dynamic and multi-faceted spatio-temporal deep learning for traffic speed forecasting," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 547–555, 2021.

[193] Y. Wang, M. Long, J. Wang, Z. Gao, and P. S. Yu, "Predrnn: Recurrent neural networks for predictive learning using spatiotemporal lstms," *Advances in neural information processing systems*, vol. 30, 2017.

[194] J. Su, W. Byeon, J. Kossaifi, F. Huang, J. Kautz, and A. Anandkumar, "Convolutional tensor-train lstm for spatio-temporal learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13714–13726, 2020.

[195] J. Zhang, Y. Zheng, and D. Qi, "Deep spatio-temporal residual networks for citywide crowd flows prediction," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[196] H. Yao, X. Tang, H. Wei, G. Zheng, and Z. Li, "Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 5668–5675, 2019.

[197] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 922–929, 2019.

[198] Z. Diao, X. Wang, D. Zhang, Y. Liu, K. Xie, and S. He, "Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 890–897, 2019.

[199] F. Wu and L. Wu, "Deepeta: a spatial-temporal sequential neural network model for estimating time of arrival in package delivery system," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 774–781, 2019.

[200] Z. Lin, M. Li, Z. Zheng, Y. Cheng, and C. Yuan, "Self-attention convlstm for spatiotemporal prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 11531–11538, 2020.

[201] C. Song, Y. Lin, S. Guo, and H. Wan, "Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 914–921, 2020.

[202] B. N. Oreshkin, A. Amini, L. Coyle, and M. J. Coates, "Fc-gaga: Fully connected gated graph architecture for spatio-temporal traffic forecasting," in *Proc. AAAI Conf. Artificial Intell*, 2021.

[203] X. Zhang, C. Huang, Y. Xu, L. Xia, P. Dai, L. Bo, J. Zhang, and Y. Zheng, "Traffic flow forecasting with spatial-temporal graph diffusion network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 15008–15015, 2021.

[204] D. Kong and F. Wu, "Hst-lstm: A hierarchical spatial-temporal long-short term memory network for location prediction.," in *IJCAI*, vol. 18, pp. 2341–2347, 2018.

[205] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 3634–3640, 2018.

[206] G. Yang, Y. Cai, and C. K. Reddy, "Spatio-temporal check-in time prediction with recurrent neural network based survival analysis," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 2976–2983, 2018.

[207] S. Fang, Q. Zhang, G. Meng, S. Xiang, and C. Pan, "Gstnet: Global spatial-temporal network for traffic flow prediction.," in *IJCAI*, pp. 2286–2293, 2019.

[208] D. Xu, W. Cheng, D. Luo, X. Liu, and X. Zhang, "Spatio-temporal attentive rnn for node classification in temporal attributed graphs.," in *IJCAI*, pp. 3947–3953, 2019.

[209] H. Liu, H. Wu, W. Sun, and I. Lee, "Spatio-temporal gru for trajectory classification," in *2019 IEEE International Conference on Data Mining (ICDM)*, pp. 1228–1233, IEEE, 2019.

[210] X. Chen, Y. Zhang, L. Du, Z. Fang, Y. Ren, K. Bian, and K. Xie, "Tssrgcn: Temporal spectral spatial retrieval graph convolutional network for traffic flow forecasting," in *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 954–959, IEEE, 2020.

[211] M. A. Ali, S. Venkatesan, V. Liang, and H. Kruppa, "Test-gcn: Topologically enhanced spatial-temporal graph convolutional networks for traffic forecasting," in *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 982–987, IEEE, 2021.

[212] M. Zhang, Y. Li, F. Sun, D. Guo, and P. Hui, "Adaptive spatio-temporal convolutional network for traffic prediction," in *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 1475–1480, IEEE, 2021.

[213] N. Lim, B. Hooi, S.-K. Ng, X. Wang, Y. L. Goh, R. Weng, and J. Varadarajan, "Stp-udgat: spatial-temporal-preference user dimensional graph attention network for next poi recommendation," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 845–854, 2020.

[214] B. Lu, X. Gan, H. Jin, L. Fu, and H. Zhang, "Spatiotemporal adaptive gated graph convolution network for urban traffic flow forecasting," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 1025–1034, 2020.

[215] C. Park, C. Lee, H. Bahng, Y. Tae, S. Jin, K. Kim, S. Ko, and J. Choo, "St-grat: A novel spatio-temporal graph attention networks for accurately forecasting dynamically changing road speed," in *Proceedings of the 29th ACM International conference on information & knowledge management*, pp. 1215–1224, 2020.

[216] X. Zhang, C. Huang, Y. Xu, and L. Xia, "Spatial-temporal convolutional graph attention networks for citywide traffic flow forecasting," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 1853–1862, 2020.

[217] Z. Wang, R. Jiang, Z. Cai, Z. Fan, X. Liu, K.-S. Kim, X. Song, and R. Shibasaki, "Spatio-temporal-categorical graph neural networks for fine-grained multi-incident co-prediction," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 2060–2069, 2021.

[218] X. Liang, G. Wang, M. R. Min, Y. Qi, and Z. Han, "A deep spatio-temporal fuzzy neural network for passenger demand prediction," in *Proceedings of the 2019 SIAM international conference on data mining*, pp. 100–108, SIAM, 2019.

[219] P. Yi, F. Huang, and J. Peng, "A fine-grained graph-based spatiotemporal network for bike flow prediction in bike-sharing systems," in *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pp. 513–521, SIAM, 2021.

[220] C. Miao, J. Fu, J. Wang, H. Yu, B. Yao, A. Zhong, J. Chen, and Z. He, "Predicting crowd flows via pyramid dilated deeper spatial-temporal network," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pp. 806–814, 2021.

[221] Y. Luo, Q. Liu, and Z. Liu, "Stan: Spatio-temporal attention network for next location recommendation," in *Proceedings of the Web Conference 2021*, pp. 2177–2185, 2021.

[222] Z. Jia, Y. Lin, J. Wang, R. Zhou, X. Ning, Y. He, and Y. Zhao, "Graphsleepnet: Adaptive spatial-temporal graph convolutional networks for sleep stage classification.," in *IJCAI*, pp. 1324–1330, 2020.

[223] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *International Conference on Learning Representations*, 2018.

[224] L. Wang, A. Adiga, J. Chen, A. Sadilek, S. Venkatramanan, and M. Marathe, "Causalgnn: Causal-based graph neural networks for spatio-temporal epidemic forecasting," *Proceedings of the AAAI conference on artificial intelligence*, 2022.

[225] Q. Zhang, J. Chang, G. Meng, S. Xiang, and C. Pan, "Spatio-temporal graph structure learning for traffic forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 1177–1185, 2020.

[226] C. Chen, K. Li, S. G. Teo, X. Zou, K. Wang, J. Wang, and Z. Zeng, "Gated residual recurrent graph neural networks for traffic prediction," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 485–492, 2019.

[227] X. Wang, Y. Ma, Y. Wang, W. Jin, X. Wang, J. Tang, C. Jia, and J. Yu, "Traffic flow prediction via spatial temporal graph neural network," in *Proceedings of The Web Conference 2020*, pp. 1082–1092, 2020.

[228] Y. Ma, Z. Guo, Z. Ren, J. Tang, and D. Yin, "Streaming graph neural networks," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 719–728, 2020.

[229] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl, and C. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 5363–5370, 2020.

[230] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," *ICLR*, 2020.

[231] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. 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*, pp. 519–527, 2020.

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

[233] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *arXiv preprint arXiv:2006.10637*, 2020.

[234] D. Fu and J. He, "Sdg: A simplified and dynamic graph neural network," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2273–2277, 2021.

[235] E. Hajiramezanali, A. Hasanzadeh, K. Narayanan, N. Duffield, M. Zhou, and X. Qian, "Variational graph recurrent neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[236] M. Liu and Y. Liu, "Inductive representation learning in temporal networks via mining neighborhood and community influences," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2202–2206, 2021.

[237] M. Yang, Z. Meng, and I. King, "Featurenorm: L2 feature normalization for dynamic graph embedding," in *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 731–740, IEEE, 2020.

[238] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[239] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[240] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953.

[241] H. H. Song, T. W. Cho, V. Dave, Y. Zhang, and L. Qiu, "Scalable proximity estimation and link prediction in online social networks," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pp. 322–335, 2009.

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

[243] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering.," in *Nips*, vol. 14, pp. 585–591, 2001.

[244] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *stat*, vol. 1050, p. 1, 2014.

[245] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, "Representation learning for dynamic graphs: A survey.," *Journal of Machine Learning Research*, vol. 21, no. 70, pp. 1–73, 2020.

[246] C. D. Barros, M. R. Mendonça, A. B. Vieira, and A. Ziviani, "A survey on embedding dynamic graphs," *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–37, 2021.

[247] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[248] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Temporal link prediction using matrix and tensor factorizations," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 5, no. 2, pp. 1–27, 2011.

[249] P. R. da Silva Soares and R. B. C. Prudêncio, "Time series based link prediction," in *The 2012 international joint conference on neural networks (IJCNN)*, pp. 1–7, IEEE, 2012.

[250] C. Chen and H. Tong, "Fast eigen-functions tracking on dynamic graphs," in *Proceedings of the 2015 SIAM international conference on data mining*, pp. 559–567, SIAM, 2015.

[251] E. Luzhnica, B. Day, and P. Lio, "Clique pooling for graph classification," *arXiv preprint arXiv:1904.00374*, 2019.

[252] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

[253] A. Leman and B. Weisfeiler, "A reduction of a graph to a canonical form and an algebra arising during this reduction," *Nauchno-Technicheskaya Informatsiya*, vol. 2, no. 9, pp. 12–16, 1968.

[254] L. El Ghaoui, F. Gu, B. Travacca, A. Askari, and A. Tsai, "Implicit deep learning," *SIAM Journal on Mathematics of Data Science*, vol. 3, no. 3, pp. 930–958, 2021.

[255] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[256] D. S. Grebenkov and B.-T. Nguyen, "Geometrical structure of laplacian eigenfunctions," *siam REVIEW*, vol. 55, no. 4, pp. 601–667, 2013.

[257] Y. Ma, X. Liu, T. Zhao, Y. Liu, J. Tang, and N. Shah, "A unified view on graph neural networks as graph signal denoising," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 1202–1211, 2021.

[258] Y.-X. Wang, J. Sharpnack, A. Smola, and R. Tibshirani, "Trend filtering on graphs," in *Artificial Intelligence and Statistics*, pp. 1042–1050, PMLR, 2015.

[259] X. Zheng, B. Zhou, J. Gao, Y. Wang, P. Lió, M. Li, and G. Montufar, "How framelets enhance graph neural networks," in *International Conference on Machine Learning*, pp. 12761–12771, PMLR, 2021.

[260] X. Zheng, B. Zhou, Y. G. Wang, and X. Zhuang, "Decimated framelet system on graphs and fast g-framelet transforms," *Journal of Machine Learning Research*, vol. 23, no. 18, pp. 1–68, 2022.

[261] X. Miao, N. M. Gürel, W. Zhang, Z. Han, B. Li, W. Min, S. X. Rao, H. Ren, Y. Shan, Y. Shao, *et al.*, "Degnn: Improving graph neural networks with graph decomposition," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1223–1233, 2021.

[262] Z. Ma, J. Xuan, Y. G. Wang, M. Li, and P. Liò, "Path integral based convolution and pooling for graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 16421–16433, 2020.

[263] J. Klicpera, S. Weißenberger, and S. Günnemann, "Diffusion improves graph learning," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 13366–13378, 2019.

[264] J. Zhao, Y. Dong, M. Ding, E. Kharlamov, and J. Tang, "Adaptive diffusion in graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

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

[266] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, "Film: Visual reasoning with a general conditioning layer," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[267] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld, "Quantum chemistry structures and properties of 134 kilo molecules," *Scientific data*, vol. 1, no. 1, pp. 1–7, 2014.

[268] X. He and T.-S. Chua, "Neural factorization machines for sparse predictive analytics," in *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 355–364, 2017.

[269] S. Zhang and L. Xie, "Improving attention mechanism in graph neural networks via cardinality preservation," in *IJCAI: proceedings of the conference*, vol. 2020, p. 1395, NIH Public Access, 2020.

[270] K.-H. Lai, D. Zha, K. Zhou, and X. Hu, "Policy-gnn: Aggregation optimization for graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 461–471, 2020.

[271] D. Zha, K.-H. Lai, K. Zhou, and X. Hu, "Experience replay optimization," in *IJCAI*, 2019.

[272] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *stat*, vol. 1050, p. 9, 2015.

[273] Z. Liu, T.-K. Nguyen, and Y. Fang, "Tail-gnn: Tail-node graph neural networks," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1109–1119, 2021.

[274] C. Vignac, A. Loukas, and P. Frossard, "Building powerful and equivariant graph neural networks with structural message-passing," *Advances in Neural Information Processing Systems*, vol. 33, pp. 14143–14155, 2020.

[275] Z. Zhang, F. Wu, and W. S. Lee, "Factor graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 8577–8587, 2020.

[276] G. Nikolentzos and M. Vazirgiannis, "Random walk graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 16211–16222, 2020.

[277] F. Xu, Q. Yao, P. Hui, and Y. Li, "Automorphic equivalence-aware graph neural network," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[278] T. He, Y. S. Ong, and L. Bai, "Learning conjoint attentions for graph neural nets," *Advances in Neural Information Processing Systems*, vol. 34, pp. 2641–2653, 2021.

[279] S. Yun, S. Kim, J. Lee, J. Kang, and H. J. Kim, "Neo-gnns: Neighborhood overlap-aware graph neural networks for link prediction," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

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

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

[282] Z. Ding, R. Zhao, J. Zhang, T. Gao, R. Xiong, Z. Yu, and T. Huang, "Spatio-temporal recurrent networks for event-based optical flow estimation," *Proceedings of the AAAI conference on artificial intelligence*, 2022.

[283] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[284] J. Wang, G. Song, Y. Wu, and L. Wang, "Streaming graph neural networks via continual learning," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 1515–1524, 2020.

[285] N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield, "Efficient graphlet counting for large networks," in *2015 IEEE International Conference on Data Mining*, pp. 1–10, IEEE, 2015.

[286] J. Lin, C. Gan, and S. Han, "Tsm: Temporal shift module for efficient video understanding," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7083–7093, 2019.

[287] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire, *et al.*, "Eta prediction with graph neural networks in google maps," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 3767–3776, 2021.

[288] Z. Liu, B. Ma, Q. Liu, J. Xu, and B. Zheng, "Heterogeneous graph neural networks for large-scale bid keyword matching," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 3976–3985, 2021.

[289] X. Qu, Z. Li, J. Wang, Z. Zhang, P. Zou, J. Jiang, J. Huang, R. Xiao, J. Zhang, and J. Gao, "Category-aware graph neural networks for improving e-commerce review helpfulness prediction," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 2693–2700, 2020.

[290] Q. Liu, R. Xie, L. Chen, S. Liu, K. Tu, P. Cui, B. Zhang, and L. Lin, "Graph neural network for tag ranking in tag-enhanced video recommendation," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 2613–2620, 2020.

[291] Y. Liu, Y. Gu, Z. Ding, J. Gao, Z. Guo, Y. Bao, and W. Yan, "Decoupled graph convolution network for inferring substitutable and complementary items," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 2621–2628, 2020.

[292] X. Niu, B. Li, C. Li, R. Xiao, H. Sun, H. Deng, and Z. Chen, "A dual heterogeneous graph attention network to improve long-tail performance for shop search in e-commerce," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3405–3415, 2020.

[293] W. Luo, H. Zhang, X. Yang, L. Bo, X. Yang, Z. Li, X. Qie, and J. Ye, "Dynamic heterogeneous graph neural network for real-time event prediction," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3213–3223, 2020.

[294] X. Fang, J. Huang, F. Wang, L. Zeng, H. Liang, and H. Wang, "Constgat: Contextual spatial-temporal graph attention network for travel time estimation at baidu maps," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2697–2705, 2020.

[295] J. Huang, H. Wang, Y. Sun, M. Fan, Z. Huang, C. Yuan, and Y. Li, "Hgamn: Heterogeneous graph attention matching network for multilingual poi retrieval at baidu maps," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 3032–3040, 2021.

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

[297] J. Xu, Z. Zhu, J. Zhao, X. Liu, M. Shan, and J. Guo, "Gemini: a novel and universal heterogeneous graph information fusing framework for online recommendations," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3356–3365, 2020.

[298] M. Wang, Y. Lin, G. Lin, K. Yang, and X.-m. Wu, "M2grl: A multi-task multi-view graph representation learning framework for web-scale recommender systems," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2349–2358, 2020.

[299] P. Barceló, F. Geerts, J. Reutter, and M. Ryschkov, "Graph neural networks with local graph parameters," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[300] Z. Huang, Y. Wang, C. Li, and H. He, "Going deeper into permutation-sensitive graph neural networks," *ICML*, 2022.

[301] A. Wijesinghe and Q. Wang, "A new perspective on "how graph neural networks go beyond weisfeiler-lehman?"," *ICML*, 2022.

[302] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[303] K. Oono and T. Suzuki, "Graph neural networks exponentially lose expressive power for node classification," in *International Conference on Learning Representations*, 2019.

[304] D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, and C. A. Raffel, "Mixmatch: A holistic approach to semi-supervised learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[305] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 338–348, 2020.

[306] B. Chamberlain, J. Rowbottom, M. I. Gorinova, M. Bronstein, S. Webb, and E. Rossi, "Grand: Graph neural diffusion," in *International Conference on Machine Learning*, pp. 1407–1418, PMLR, 2021.

[307] D. He, R. Guo, X. Wang, D. Jin, Y. Huang, and W. Wang, "Inflation improves graph neural networks," in *Proceedings of the ACM Web Conference 2022*, pp. 1466–1474, 2022.

[308] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," *Advances in neural information processing systems*, vol. 30, 2017.

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

[310] A. Bojchevski, J. Klicpera, B. Perozzi, A. Kapoor, M. Blais, B. Rózemberczki, M. Lukasik, and S. Günnemann, "Scaling graph neural networks with approximate pagerank," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2464–2473, 2020.

[311] Z. Huang, S. Zhang, C. Xi, T. Liu, and M. Zhou, "Scaling up graph neural networks via graph coarsening," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 675–684, 2021.

[312] M. Fey, J. E. Lenssen, F. Weichert, and J. Leskovec, "GNNAutoScale: Scalable and expressive graph neural networks via historical embeddings," in *International Conferences on Machine Learning (ICML)*, 2021.

[313] M. Yoon, T. Gervet, B. Shi, S. Niu, Q. He, and J. Yang, "Performance-adaptive sampling strategy towards fast and accurate graph neural networks," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2046–2056, 2021.

[314] J. Peng, Y. Shen, and L. Chen, "Graphangel: Adaptive and structure-aware sampling on graph neural networks," in *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 479–488, IEEE, 2021.

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

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

[317] J. Liu, K. Kawaguchi, B. Hooi, Y. Wang, and X. Xiao, "Eignn: Efficient infinite-depth graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[318] D. Lukovnikov and A. Fischer, "Improving breadth-wise backpropagation in graph neural networks helps learning long-range dependencies.," in *International Conference on Machine Learning*, pp. 7180–7191, PMLR, 2021.

[319] H. Liu, Y. Yang, and X. Wang, "Overcoming catastrophic forgetting in graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 8653–8661, 2021.

[320] F. Zhou and C. Cao, "Overcoming catastrophic forgetting in graph neural networks with experience replay," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 4714–4722, 2021.

[321] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annual review of sociology*, vol. 27, no. 1, pp. 415–444, 2001.

[322] X. Li, R. Zhu, Y. Cheng, C. Shan, S. Luo, D. Li, and W. Qian, "Finding global homophily in graph neural networks when meeting heterophily," *ICML*, 2022.

[323] V. Nanda, "Computational algebraic topology lecture notes," 2021.

[324] E. Thiede, W. Zhou, and R. Kondor, "Autobahn: Automorphism-based graph neural nets," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[325] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," in *International Conference on Learning Representations*, 2020.

[326] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," *International Conference on Learning Representations*, 2022.

[327] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graph neural architecture search.," in *IJCAI*, vol. 20, pp. 1403–1409, 2020.

[328] J. You, Z. Ying, and J. Leskovec, "Design space for graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 17009–17021, 2020.

[329] Z. Wang, S. Di, and L. Chen, "Autogel: An automated graph neural network with explicit link information," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[330] Z. Huan, Y. Quanming, and T. Weiwei, "Search to aggregate neighborhood for graph neural network," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 552–563, IEEE, 2021.

[331] A. Dalmia and M. Gupta, "Towards interpretation of node embeddings," in *Companion Proceedings of the The Web Conference 2018*, pp. 945–952, 2018.

[332] A. Gogoglou, C. B. Bruss, and K. E. Hines, "On the interpretability and evaluation of graph representation learning," *arXiv preprint arXiv:1910.03081*, 2019.

[333] S. Park, J. Bak, and A. Oh, "Rotated word vector representations and their interpretability," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 401–411, 2017.

[334] Y. Adi, E. Kermany, Y. Belinkov, O. Lavi, and Y. Goldberg, "Fine-grained analysis of sentence embeddings using auxiliary prediction tasks," in *ICLR*, 2017.

[335] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, "Heterogeneous network embedding via deep architectures," in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 119–128, 2015.

[336] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[337] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," in *Social network data analytics*, pp. 115–148, Springer, 2011.

[338] X. Yu, X. Ren, Y. Sun, Q. Gu, B. Sturt, U. Khandelwal, B. Norick, and J. Han, "Personalized entity recommendation: A heterogeneous information

network approach," in *Proceedings of the 7th ACM international conference on Web search and data mining*, pp. 283–292, 2014.

[339] N. Kaji and H. Kobayashi, "Incremental skip-gram model with negative sampling," *arXiv preprint arXiv:1704.03956*, 2017.

[340] C. May, K. Duh, B. Van Durme, and A. Lall, "Streaming word embeddings with the space-saving algorithm," *arXiv preprint arXiv:1704.07463*, 2017.

[341] P. Goyal, N. Kamra, X. He, and Y. Liu, "Dyngem: Deep embedding method for dynamic graphs," *arXiv preprint arXiv:1805.11273*, 2018.

[342] J. Leskovec and A. Krevl, "{SNAP Datasets}:{Stanford} large network dataset collection," 2015.

[343] "http://konect.uni-koblenz.de/networks/,"

[344] "http://snap.stanford.edu/data/sx-mathoverflow.html,"

[345] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: extraction and mining of academic social networks," in *Proceedings of the 14th ACM SIGKDD*, pp. 990–998, ACM, 2008.

[346] R. Lippman, R. Cunningham, D. Fried, I. Graf, K. Kendall, S. Webster, and M. Zissman, "Results of the darpa 1998 offline intrusion detection evaluation," in *Slides presented at RAID 1999 Conference*, 1999.

[347] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[348] L. Tang and H. Liu, "Leveraging social media networks for classification," *Data Mining and Knowledge Discovery*, vol. 23, no. 3, pp. 447–478, 2011.

[349] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[350] R. Trivedi, M. Farajtbar, P. Biswal, and H. Zha, "Representation learning over dynamic graphs," *arXiv preprint arXiv:1803.04051*, 2018.

[351] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[352] X. Du and C. Cardie, "Identifying where to focus in reading comprehension for neural question generation," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 2067–2073, 2017.

[353] G. Chen, J. Yang, C. Hauff, and G.-J. Houben, "Learningq: a large-scale dataset for educational question generation," in *Twelfth International AAAI Conference on Web and Social Media*, 2018.

[354] Z. Wang, A. S. Lan, W. Nie, A. E. Waters, P. J. Grimaldi, and R. G. Baraniuk, "Qg-net: a data-driven question generation model for educational content," in *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, p. 7, ACM, 2018.

[355] Y. Zhao, X. Ni, Y. Ding, and Q. Ke, "Paragraph-level neural question generation with maxout pointer and gated self-attention networks," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3901–3910, 2018.

[356] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, *et al.*, "Moses: Open source toolkit for statistical machine translation," in *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, pp. 177–180, 2007.

[357] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[358] S. Ranshous, S. Shen, D. Koutra, S. Harenberg, C. Faloutsos, and N. F. Samatova, "Anomaly detection in dynamic networks: a survey," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 7, no. 3, pp. 223–247, 2015.

[359] Q. Ding, N. Katenka, P. Barford, E. Kolaczyk, and M. Crovella, "Intrusion as (anti) social communication: characterization and detection," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 886–894, ACM, 2012.

[360] J. Xu and C. R. Shelton, "Intrusion detection using continuous time bayesian networks," *Journal of Artificial Intelligence Research*, vol. 39, pp. 745–774, 2010.

[361] A. Ghoting, M. E. Otey, and S. Parthasarathy, "Loaded: Link-based outlier and anomaly detection in evolving data sets," in *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pp. 387–390, IEEE, 2004.

[362] T. Zhang, X. Zhuang, S. Pande, and W. Lee, "Anomalous path detection with hardware support," in *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pp. 43–54, ACM, 2005.

[363] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data mining and knowledge discovery*, vol. 29, no. 3, pp. 626–688, 2015.

[364] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[365] J. Tang, J. Zhang, L. Yao, and J. Li, "Extraction and mining of an academic social network," in *Proceedings of the 17th WWW*, pp. 1193–1194, ACM, 2008.

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

[367] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[368] Z. He, C.-Y. Chow, and J.-D. Zhang, "Stcnn: A spatio-temporal convolutional neural network for long-term traffic prediction," in *2019 20th IEEE International Conference on Mobile Data Management (MDM)*, pp. 226–233, IEEE, 2019.

[369] H. Yu, Z. Wu, S. Wang, Y. Wang, and X. Ma, "Spatiotemporal recurrent convolutional networks for traffic prediction in transportation networks," *Sensors*, vol. 17, no. 7, p. 1501, 2017.

[370] A. Mohamed, K. Qian, M. Elhoseiny, and C. Claudel, "Social-stgcnn: A social spatio-temporal graph convolutional neural network for human trajectory prediction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14424–14432, 2020.

[371] A. P. Mukherjee, P. Xu, and S. Tirthapura, "Mining maximal cliques from an uncertain graph," in *2015 IEEE 31st International Conference on Data Engineering*, pp. 243–254, IEEE, 2015.

[372] G. Goel and J. Gustedt, "Bounded arboricity to determine the local structure of sparse graphs," in *International Workshop on Graph-Theoretic Concepts in Computer Science*, pp. 159–167, Springer, 2006.

[373] D. Eppstein and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," in *International Symposium on Experimental Algorithms*, pp. 364–375, Springer, 2011.

[374] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," *Journal of Experimental Algorithmics (JEA)*, vol. 18, pp. 3–1, 2013.

[375] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical computer science*, vol. 363, no. 1, pp. 28–42, 2006.

[376] D. R. Lick and A. T. White, "k-degenerate graphs," *Canadian Journal of Mathematics*, vol. 22, no. 5, pp. 1082–1096, 1970.

[377] S. J. Prince, *Computer vision: models, learning, and inference*. Cambridge University Press, 2012.

[378] A. Farasat, A. Nikolaev, S. N. Srihari, and R. H. Blair, "Probabilistic graphical models in modern social network analysis," *Social Network Analysis and Mining*, vol. 5, no. 1, pp. 1–18, 2015.

[379] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[380] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1269–1278, 2019.

[381] www.https://github.com/snap stanford/CAW

[382] X. Yan, X. Fan, P. Yang, Z. Wu, S. Pan, L. Chen, Y. Zang, and C. Wang, "Contig: Continuous representation learning on temporal interaction graphs," *arXiv preprint arXiv:2110.06088*, 2021.

[383] S. Tian, T. Xiong, and L. Shi, "Streaming dynamic graph neural networks for continuous-time temporal graph modeling," in *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 1361–1366, IEEE, 2021.

[384] F. Bloch, M. O. Jackson, and P. Tebaldi, "Centrality measures in networks," *Available at SSRN 2749124*, 2019.

[385] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," *arXiv preprint arXiv:2002.05287*, 2020.

[386] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-scale attributed node embedding," *arXiv preprint arXiv:1909.13021*, 2019.

[387] W. R. Knight, "A computer method for calculating kendall's tau with ungrouped data," *Journal of the American Statistical Association*, vol. 61, no. 314, pp. 436–439, 1966.

[388] K. Xu, M. Zhang, S. Jegelka, and K. Kawaguchi, "Optimization of graph neural networks: Implicit acceleration by skip connections and more depth," in *International Conference on Machine Learning*, pp. 11592–11602, PMLR, 2021.

[389] G. Yehudai, E. Fetaya, E. Meirom, G. Chechik, and H. Maron, "From local structures to size generalization in graph neural networks," in *International Conference on Machine Learning*, pp. 11975–11986, PMLR, 2021.

[390] J. Brandstetter, D. E. Worrall, and M. Welling, "Message passing neural pde solvers," in *International Conference on Learning Representations*, 2022.

[391] T. Dash, S. Chitlangia, A. Ahuja, and A. Srinivasan, "A review of some techniques for inclusion of domain-knowledge into deep neural networks," *Scientific Reports*, vol. 12, no. 1, pp. 1–15, 2022.

[392] T. Dash, A. Srinivasan, and A. Baskar, "Inclusion of domain-knowledge into gnns using mode-directed inverse entailment," *Machine Learning*, vol. 111, no. 2, pp. 575–623, 2022.

[393] T. Dash, A. Srinivasan, and L. Vig, "Incorporating symbolic domain knowledge into graph neural networks," *Machine Learning*, vol. 110, no. 7, pp. 1609–1636, 2021.

[394] M. Ringsquandl, H. Sellami, M. Hildebrandt, D. Beyer, S. Henselmeyer, S. Weber, and M. Joblin, "Power to the relational inductive bias: Graph neural networks in electrical power grids," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 1538–1547, 2021.

[395] J. Zhang, A.-T. Kuo, J. Zhao, Q. Wen, E. Winstanley, C. Zhang, and Y. Ye, "Rxnet: Rx-refill graph neural network for overprescribing detection," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 2537–2546, 2021.

[396] R. Guan, Y. Liu, X. Feng, and X. Li, "Vpalg: Paper-publication prediction with graph neural networks," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 617–626, 2021.